TECNOLOGÍAS DE LA INFORMACIÓN I y II.

TEMARIO.

TEMARIO 1º BACHILLERATO.

- Programación: App Inventor.
- Ofimática: Excel.
- Robótica: Robomind.

TEMARIO 2º BACHILLERATO.

- Programación: Python.
- Redes de ordenadores.
- Ofimática: Bases de datos.
- Robótica: Gearsbot.

PARTE I:
DESARROLLO DE APPS PARA DISPOSITIVOS ANDROID.

INTRODUCCIÓN.

¿QUÉ ES APP INVENTOR?

El objetivo de estos apuntes es enseñar los fundamentos de la programación usando la herramienta App Inventor (versión 2). **App Inventor** es un entorno de programación que permite desarrollar aplicaciones, comúnmente llamadas "apps", para teléfonos móviles y tabletas Android.

Para poder construir aplicaciones para móviles y tabletas a nivel profesional se requiere de un conocimiento avanzado de lenguajes de programación tales como Java, C++, o Kotlin. Por el contrario, App Inventor es una herramienta pedagógica especialmente diseñada para formar a programadores principiantes que nunca han desarrollado aplicaciones móviles previamente. En App Inventor usamos un **Diseñador** para añadir los componentes que constituyen la pantalla de nuestra app y especificar sus propiedades. A continuación, utilizamos el **Editor de Bloques** para programar el comportamiento de esos componentes. En este editor no usamos Java, C++, ni ningún lenguaje de programación textual, sino que programamos visualmente utilizando bloques de código.

Con App Inventor utilizamos un ordenador personal para crear nuestras aplicaciones. Además, nos podemos conectar a un teléfono o a una tableta inalámbricamente o mediante un cable USB, para comprobar cómo funciona la aplicación que estamos construyendo en el dispositivo vinculado.

Y si no tenemos un dispositivo Android que conectar al ordenador, App Inventor también proporciona un emulador Android. El **emulador** es un teléfono virtual simulado en el ordenador. Conforme desarrollamos la aplicación, podemos usar este emulador para verificar su correcto funcionamiento. Aunque el emulador es limitado (por ejemplo, no dispone de sensor GPS para reportar la localización), incluye muchas de las características básicas de un teléfono móvil real.

¿QUÉ ES ANDROID?

Ya hemos mencionado la palabra *Android* unas cuantas veces en esta introducción. **Android** es un **sistema operativo** que se ejecuta en muchos dispositivos móviles. El sistema operativo es el programa encargado de gestionar el hardware del dispositivo (esto es, la parte física del dispositivo que podemos ver y tocar, incluyendo los chips, circuitos integrados, cámaras, pantallas táctiles, etc.).

Windows, Mac, y Linux son sistemas operativos que se usan principalmente en ordenadores y portátiles. Por su parte, Android, iOS, y Windows Phone son los principales sistemas operativos que podemos encontrar en los dispositivos móviles. App Inventor nos permite programar cualquier dispositivo que trabaje con Android, pero no funciona con iPhones ni con dispositivos Windows Phone.

CONFIGURAR APP INVENTOR.

Para poder usar App Inventor en nuestro ordenador necesitamos tener una cuenta personal en **Google**. (Si tenemos correo electrónico de **Gmail** ya disponemos de una cuenta Google, pero si no, debemos crear una). Para crear una nueva cuenta de Google abrimos el navegador de Internet, acudimos a la web www.google.com/accounts/newaccount, y seguimos los pasos que se indican. Debemos asegurarnos de recordar el nombre de usuario y la contraseña elegidos al registrarnos, porque los necesitaremos para poder entrar en App Inventor.

Una vez disponemos de una cuenta Google, debemos configurar App Inventor. Vamos a la página web http://appinventor.mit.edu/explore/, pinchamos en "Get Started", y seleccionamos el enlace "Setup Instructions". A continuación veremos que hay tres opciones disponibles: (1) Construir aplicaciones usando

un dispositivo Android y una conexión WiFi, (2) Usar el emulador (si no disponemos de un dispositivo Android), y (3) Construir aplicaciones usando un dispositivo Android y un cable USB (si tenemos un dispositivo Android, pero no disponemos de conexión a internet vía WiFi). En cada una de estas opciones, llegaremos a una página con instrucciones que deberemos seguir.

La opción recomendada por la propia web de App Inventor es la primera, y es la que (siempre que sea posible) utilizaremos aquí. En cualquiera de los tres casos, necesitaremos hacer algunas configuraciones previas en el ordenador y en nuestro dispositivo Android:

1) En primer lugar, debemos descargar la app gratuita "**MIT AI2 Companion**" de **Google Play Store** e instalarla en nuestro dispositivo Android. Esta aplicación nos permitirá conectarnos al ordenador y lanzar pruebas en vivo en nuestro móvil o tableta de las apps que estamos construyendo, para comprobar que el diseño y el funcionamiento son los esperados.

2) En segundo lugar, necesitamos tener instalado un lector de **códigos QR** en nuestro dispositivo Android. Algunos dispositivos ya vienen con escáneres QR preinstalados por defecto, pero de no ser así debemos descargar e instalar un lector gratuito desde Play Store. Como veremos, el lector será necesario para poder descargar a nuestro dispositivo la versión final de la aplicación que hayamos desarrollado, y poder tenerla en funcionamiento aun cuando nos hayamos desconectado del ordenador.

3) Si no podemos conectarnos a nuestro dispositivo Android a través de una red WiFi, deberemos hacerlo mediante un cable USB. En ese caso, tendremos que instalar en el ordenador el software de configuración de App Inventor disponible en la web http://appinventor.mit.edu/explore/ai2/windows.html (asumiendo que nuestro sistema operativo sea Windows). Dentro de este software está el programa "**aiStarter**", que le permite al navegador de Internet comunicarse con un dispositivo Android a través del cable USB (y con el emulador, en el caso de que no dispongamos de un dispositivo Android).

Estas configuraciones previas son necesarias para el correcto funcionamiento de App Inventor, y conviene traer las dos primeras hechas desde casa. De esta forma evitaremos que el profesor tenga que perder el tiempo configurando los dispositivos de todos los alumnos.

Habiendo configurado adecuadamente App Inventor, ya podemos empezar a crear nuestras apps. Para ello nos conectamos a la página de inicio de App Inventor, http://appinventor.mit.edu/explore/, y pinchamos en el botón "Create Apps!" localizado arriba y a la derecha de la página. Una opción más rápida es conectarnos directamente a http://ai2.appinventor.mit.edu/. Para entrar en el entorno de desarrollo on-line de App Inventor, deberemos iniciar sesión con nuestra cuenta de Google.

¿QUÉ ES UN PROGRAMA?

Antes de empezar a trabajar con App Inventor propiamente dicho, vamos a presentar algunos conceptos básicos sobre **programación**. Los conceptos que discutiremos aquí aplican a cualquier tipo de programación, independientemente de si la máquina a programar es un ordenador personal, un superordenador, un dispositivo móvil, o cualquier otro aparato programable.

Pero para poder responder adecuadamente a la pregunta "¿Qué es un programa?", primero debemos tener claro qué es un **ordenador**. Para aprender programación no es necesario conocer los detalles del funcionamiento de un ordenador, y nos basta con una definición básica:

Un ordenador es un dispositivo que obedece instrucciones.

Un ordenador no sabe hacer nada por sí mismo. Los ordenadores únicamente obedecen las órdenes que les damos en forma de **instrucciones**. Ahora bien, un ordenador no es capaz de obedecer cualquier tipo de instrucción. Por ejemplo, no podemos decirle a un ordenador que nos prepare el desayuno y nos lo sirva en la cama. Eso no es el tipo de instrucciones que un *ordenador* puede entender; solo es el tipo de instrucciones que un *humano* puede entender. El propósito más habitual de los ordenadores es operar datos: Los ordenadores solo pueden hacer tareas simples como sumar y multiplicar números, mostrar resultados por pantalla, almacenar datos para que poder recuperarlos más adelante, y cosas por el estilo. Una vez sabido esto, podemos ampliar nuestra definición de lo que es un ordenador diciendo:

Un ordenador es un dispositivo que obedece instrucciones para manipular y almacenar datos.

Ahora, cuando se diseña y construye un ordenador, se le equipa con un conjunto de operaciones que puede realizar con los datos. La mayoría de estas operaciones son muy básicas. Algunos ejemplos son:

- Sumar dos números.
- Restar un número a otro número.
- Multiplicar dos números.
- Dividir un número entre otro número.
- Mover un número de una posición de memoria a otra.
- Determinar si un número es igual, mayor, o menor que otro número.
- Etc.

Una instrucción es simplemente una orden para decirle a un ordenador que realice una de las operaciones que es capaz de efectuar.

Aunque hay una instrucción para cada una de las operaciones que un ordenador puede realizar, las distintas instrucciones individuales no son muy útiles por sí solas. Como las operaciones que pueden hacer los ordenadores son tan básicas, toda tarea que queramos implementar en un ordenador suele requerir la realización conjunta de muchas de estas instrucciones básicas. Por ejemplo, si queremos que un ordenador calcule los intereses que nos rendirán este año los ahorros que tenemos en el banco, el ordenador deberá realizar una gran cantidad de instrucciones, y en el orden adecuado.

Con todo esto en mente, ya podemos definir qué es un **programa**:

Un programa es el conjunto de instrucciones que sigue un ordenador para poder llevar a cabo una tarea específica.

Por consiguiente, si queremos que un ordenador lleve a cabo una cierta tarea debemos tener un programa, que no es más que un conjunto de instrucciones. Además, las instrucciones de un programa deben escribirse siguiendo una secuencia lógica en un orden determinado, para que la tarea que implementa pueda efectuarse de forma correcta. Cuando un ordenador está siguiendo las instrucciones de un programa, decimos que el ordenador está **ejecutando** ese programa.

ALGORITMOS Y LENGUAJES DE PROGRAMACIÓN.

Los **programadores** son aquellas personas que saben cómo programar un ordenador. El trabajo de los programadores es sumamente importante, ya que sin programas los ordenadores no sabrían hacer absolutamente nada.

Cuando un programador empieza a escribir un programa, una de las primeras cosas que debe hacer es desarrollar un algoritmo. Un **algoritmo** es el conjunto de pasos lógicos que deben efectuarse en un cierto orden para resolver una determinada tarea.

Por ejemplo, si estamos escribiendo un programa para calcular el sueldo mensual en bruto de un empleado, los pasos lógicos que deberíamos seguir son los siguientes:

1) Obtener el número de horas que el empleado ha trabajado este mes, y almacenarlo en memoria.
2) Obtener el salario que ese empleado cobra por hora, y almacenarlo en memoria.
3) Multiplicar el número de horas trabajadas ese mes por el precio de la hora trabajada, y almacenar el resultado en memoria.
4) Mostrar por pantalla un mensaje que muestre el sueldo bruto mensual del empleado. Este mensaje debe incluir el resultado del cálculo efectuado en el paso 3.

Notar que los pasos de este algoritmo están ordenados secuencialmente. El paso 1 debería realizarse antes que el paso 2, el paso 2 antes que el paso 3, y así sucesivamente. Es importante que cada uno de estos pasos lógicos se efectúe en el orden adecuado.

Los pasos que hemos mostrado más arriba están escritos en lenguaje llano. Aunque cualquier persona podría entenderlo, este algoritmo no puede ser implementado por un ordenador tal y como está ahora mismo. Para que un ordenador pueda entender un algoritmo, las instrucciones deben traducirse a **lenguaje máquina**, que es el único lenguaje que un ordenador puede entender. En lenguaje máquina, cada instrucción se representa mediante un número binario. Un **número binario** es simplemente una cadena de bits, esto es, una secuencia de ceros y unos, como por ejemplo:

101101000001110110

Cuando una persona observa este número solo ve un conjunto de 1's y 0's. Pero para un ordenador este número es una instrucción, esto es, una orden para realizar una operación. Un programa que esté listo para ser ejecutado por un ordenador es simplemente una cadena de números binarios que representan instrucciones.

Como podemos imaginar, el proceso de traducir un algoritmo de lenguaje humano a lenguaje máquina es muy difícil y tedioso. Por lo tanto, para que el trabajo de los programadores sea más sencillo se han inventado los lenguajes de programación. Los **lenguajes de programación** usan palabras en lugar de números binarios para representar las instrucciones. De esta forma, los programadores pueden escribir sus programas mediante un lenguaje de programación, que es mucho más fácil de usar y de entender que el lenguaje máquina. A continuación, un **compilador** o **intérprete** traduce ese programa (escrito en base a un cierto lenguaje de programación) a lenguaje máquina.

A lo largo de los años se han creado una gran variedad de lenguajes de programación. Algunos ejemplos de son Java, Python, C++, y Visual Basic. Éstos son solo unos cuantos ejemplos de los lenguajes de programación que utilizan los programadores profesionales para crear software comercial. Cada uno de estos lenguajes tiene su propio conjunto de palabras que el programador debe aprender para poder programar con ese lenguaje. Las palabras que representan las instrucciones de un cierto lenguaje de programación se denominan **palabras clave** o **palabras reservadas**. Por ejemplo, la palabra `print` es la palabra reservada que usa el lenguaje Python para mostrar un mensaje por pantalla:

```
print ('Hello world!')
```

Esta instrucción le dice al ordenador que muestre el mensaje `Hello world!` por pantalla. Si comparamos esta instrucción con el código binario que vimos antes, entenderemos que los programadores prefieran usar lenguajes de programación a usar lenguaje máquina. La tarea de escribir programas usando lenguajes de programación es infinitamente más sencilla que hacerlo en lenguaje máquina.

Además de las palabras reservadas, los lenguajes de programación también disponen de **operadores** que permiten efectuar varias operaciones con los datos. Por ejemplo, todos los lenguajes de programación

tienen operadores matemáticos para realizar operaciones aritméticas básicas. En Python y en otros muchos lenguajes, el operador + permite sumar dos números:

```
12 + 75
```

Y junto con las palabras reservadas y los operadores, cada lenguaje tiene su propia **sintaxis**, que es el conjunto de reglas que deben obedecerse estrictamente a la hora de escribir un programa. Las reglas de sintaxis indican la forma en la que deben usarse las palabras clave, los operadores, y los distintos caracteres de puntuación para escribir programas en un cierto lenguaje. Cuando aprendemos un lenguaje de programación en particular, debemos aprendernos las reglas sintácticas que gobiernan su uso

Al escribir un programa en un lenguaje de programación tradicional, convertimos un algoritmo en una serie de sentencias. Una **sentencia** consiste en una combinación de palabras reservadas, operadores, caracteres de puntuación, y otros elementos, organizados en una secuencia correcta para efectuar una operación.

Por ejemplo, el siguiente código Python utiliza una sentencia condicional para comprobar la contraseña de acceso que introduce el usuario. Si la contraseña es correcta, el programa imprime por pantalla el mensaje `Access granted.` y llama a una función (no detallada) que abre las puertas (de una instalación secreta, de una caja fuerte, etc.). En caso contrario se imprime por pantalla el mensaje `Access denied.` y se llama a una función (no detallada) que hace sonar una alarma:

```
pass = input()
if  pass == secretPassword:
      print('Access granted.')
      openLocks()
else:
      print('Access denied.')
      soundAlarm()
```

Los programadores llaman a estas sentencias **código**. Normalmente escribiremos nuestras sentencias de programación en un editor de texto, guardaremos el código en un archivo, y después usaremos el compilador para convertir este código en un programa ejecutable. Un **programa ejecutable** es un archivo que contiene las instrucciones en lenguaje máquina que el ordenador ejecutará directamente.

PROGRAMAR EN APP INVENTOR.

Una de las cosas que hace que aprender a programar con App Inventor sea muy sencillo es que elimina muchos de los errores que los programadores noveles suelen cometer. Con un lenguaje de programación tradicional (como Python, Java, o C++) los principiantes suelen cometer errores al escribir el código, como por ejemplo, palabras reservadas mal escritas, omisión de caracteres de puntuación, etc. Estos errores se conocen comúnmente como *errores sintácticos*. Si un programa contiene un solo error sintáctico, el compilador no puede traducirlo a lenguaje máquina para convertirlo en un archivo ejecutable. Debido a ello, tanto los estudiantes como los programadores profesionales invierten un montón de tiempo rastreando y corrigiendo errores sintácticos.

En App Inventor los errores sintácticos no ocurren nunca, porque es un lenguaje que no emplea sentencias escritas. En vez de eso, el programador arrastra y suelta **bloques de código** en el Editor. Estos bloques de código son bloques gráficos que representan las instrucciones y los datos, y que pueden conectarse unos a otros como piezas de un puzle para crear sentencias de programación plenamente funcionales. Como estos bloques ya están construidos y no hay que escribirlos, evitan perder el tiempo localizando y corrigiendo errores sintácticos. En su lugar, podemos invertir nuestro tiempo en planificar las acciones que queremos que realice nuestra app, y en conectar los bloques necesarios para ello en el orden adecuado.

PROGRAMACIÓN ORIENTADA A EVENTOS.

Las aplicaciones que construiremos con App Inventor son **programas dirigidos por eventos**. Esto significa que cuando una app se está ejecutando espera a que se produzcan ciertos eventos específicos, y cuando ocurren, responde a ellos.

¿A qué nos referimos por evento? Un **evento** es la ocurrencia de un hecho o acción, como cuando un usuario presiona un botón, o desliza su dedo sobre la pantalla. La recepción de un mensaje de texto también constituye un evento, así como la acción de inclinar o sacudir el teléfono móvil. Cuando creamos una app decidimos a qué eventos responderá la app, y escribimos el código que se ejecuta cuando esos eventos se producen.

Un ejemplo de app muy sencilla sería aquella que muestre un mensaje en la pantalla del móvil cuando el usuario presiona un botón. El evento que controla el funcionamiento de la app es que el usuario pulse el botón, y la acción que desencadena la ocurrencia de ese evento es la impresión del mensaje por pantalla.

1. INTRODUCCIÓN A APP INVENTOR.

1.1. EL ENTORNO DE DESARROLLO DE APP INVENTOR.

Para comenzar a programar con App Inventor abrimos un navegador y nos conectamos a la web http://ai2.appinventor.mit.edu. El navegador abrirá la versión más reciente de App Inventor, que actualmente es App Inventor 2 (de ahora en adelante, simplemente **App Inventor**).

El entorno de programación de App Inventor tiene tres partes principales:

- El **Diseñador de Componentes** (figura 1.1), que se usa para seleccionar los **componentes** que constituirán la pantalla de nuestra aplicación (botones, etiquetas, sonidos, sensores, etc.).
- El **Editor de Bloques** (figura 1.2), que sirve para especificar cómo se comportarán los componentes de la aplicación (por ejemplo, qué ocurre cuando el usuario pulsa un botón).
- Un dispositivo Android en el que podemos ejecutar e ir probando nuestra aplicación conforme la vamos construyendo. Si no disponemos de un dispositivo Android, también podemos probar la aplicación en el **emulador Android** integrado en App Inventor (figura 1.3).

La primera vez que nos conectemos a http://ai2.appinventor.mit.edu veremos la página de Proyectos. (En vuestro caso esta página estará en blanco porque todavía no habéis creado ningún proyecto; en mi caso, la página está llena de las apps que ya he desarrollado). Para crear un nuevo proyecto pinchamos en el botón "Start new project", escribimos el nombre del proyecto (por ejemplo, "prueba"), y pinchamos en "OK". Tras unos pocos segundos, la primera ventana que se abre es el Diseñador. Podemos acceder al Editor de Bloques pinchando en el botón "Blocks" en la esquina superior derecha.

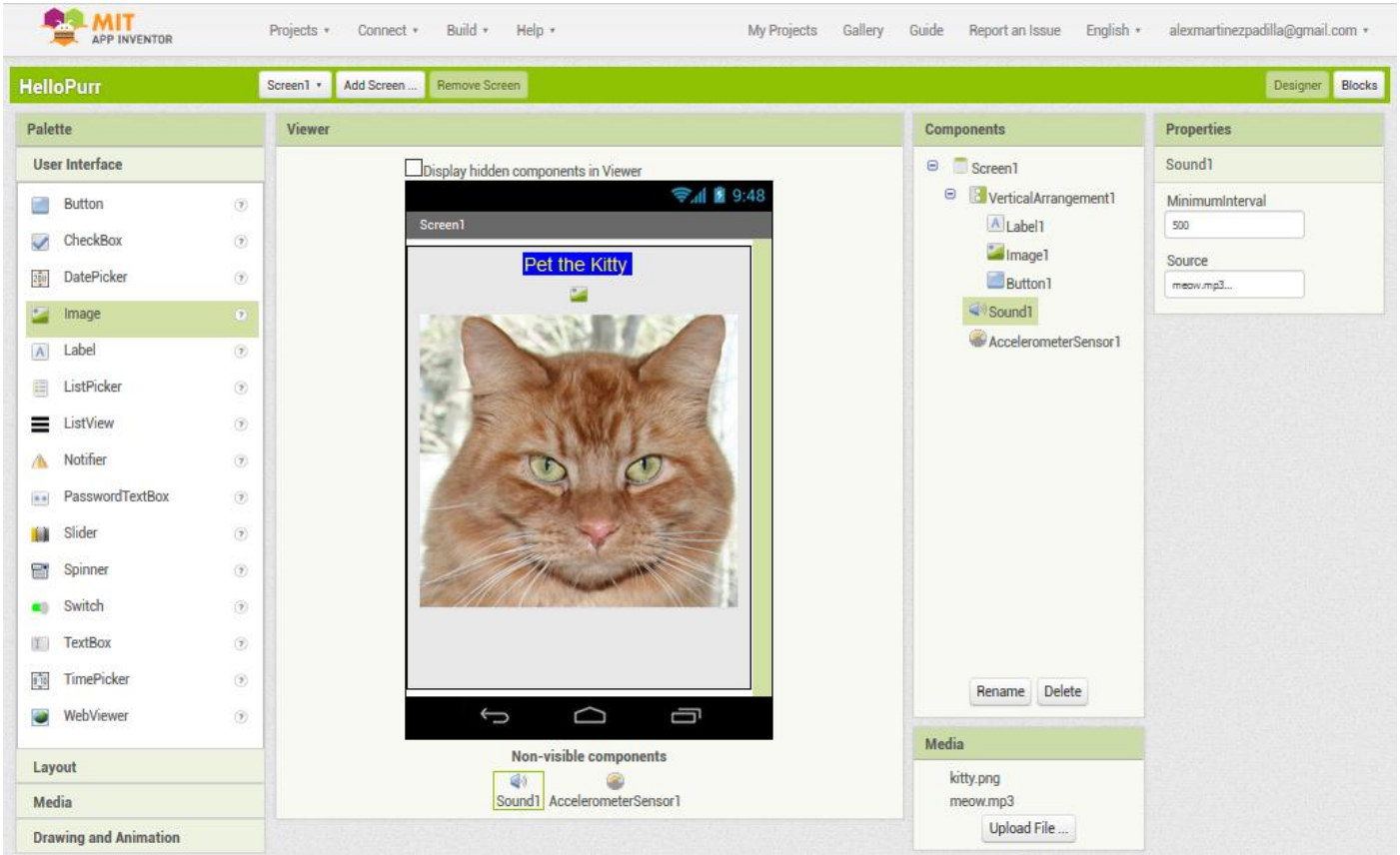


Figura 1.1. El Diseñador de Componentes.

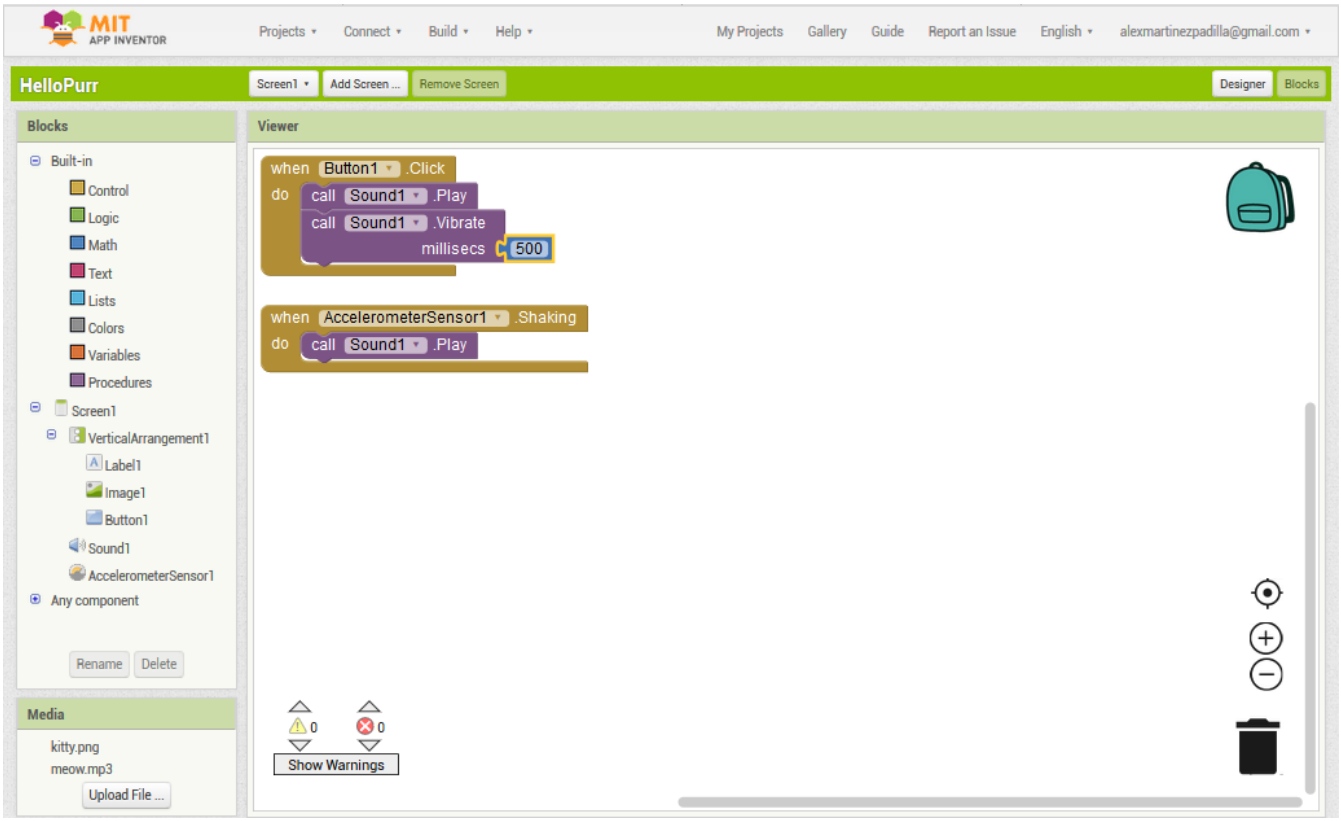


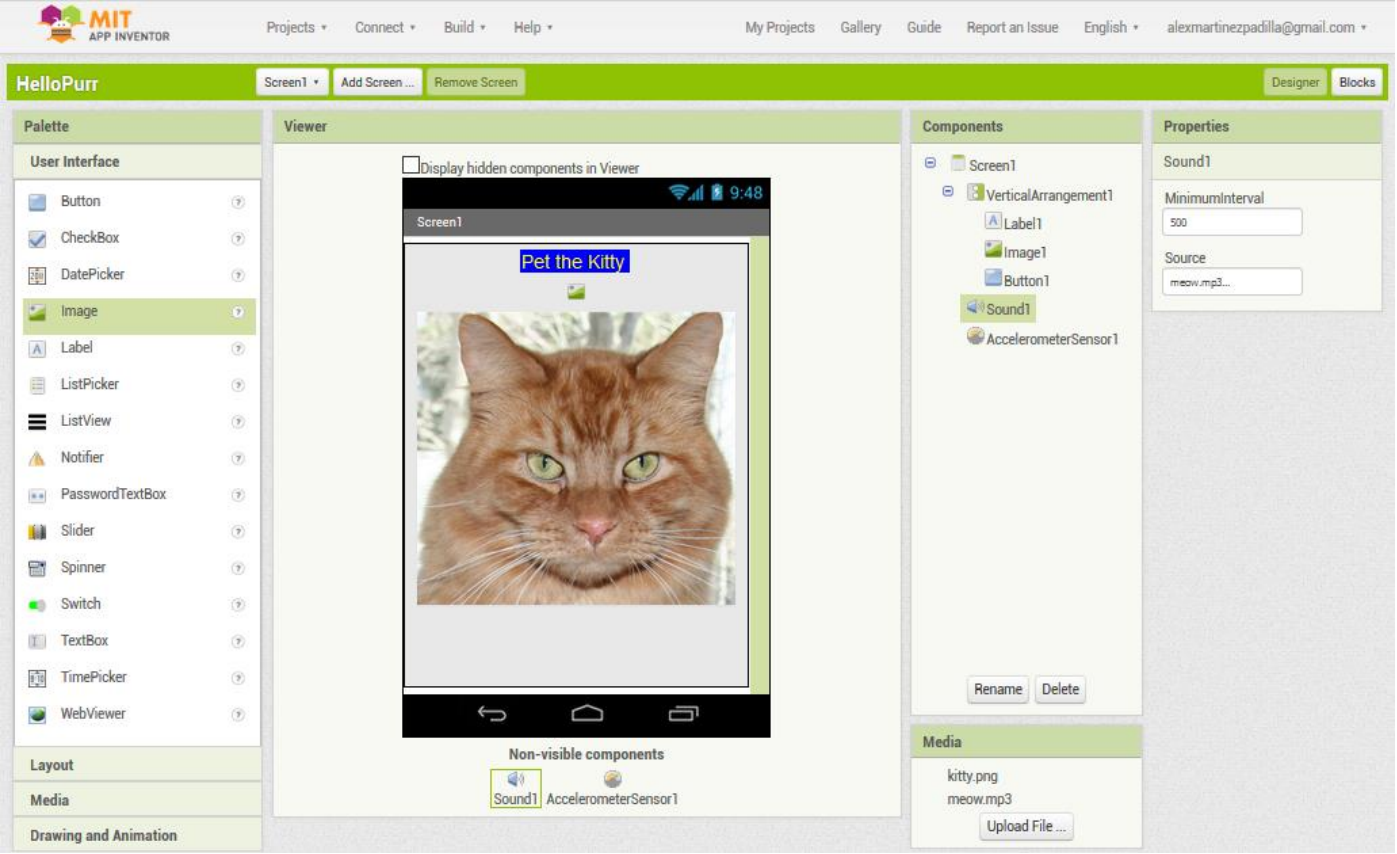
Figura 1.2. El Editor de Bloques.

Figura 1.3. El emulador Android de App Inventor.

En las siguientes secciones vamos a analizar con detalle estas tres partes principales.

1.2. EL DISEÑADOR DE COMPONENTES.

El primer paso para crear una app es diseñarla. Diseñar una app significa decidir el aspecto que tendrá la pantalla de la app, y elegir los **componentes** (botones, etiquetas, sonidos, etc.) que necesitaremos añadir para que la app funcione. El lugar donde hacemos todas estas cosas es el **Diseñador de Componentes**.



El Diseñador está dividido en varias secciones:

- La zona en blanco en el centro se denomina **Visor** (Viewer). Aquí es donde ubicamos y organizamos los componentes que incorporará nuestra app. El Visor sólo muestra de forma aproximada el aspecto que tendrá la app. Por ejemplo, una línea de texto podría romperse en varias líneas en nuestro dispositivo Android, pero no hacerlo en el Visor. Para comprobar el aspecto real de una app, debemos probarla en nuestro dispositivo Android o en el emulador.

- El menú a la izquierda del Visor es la **Paleta** (Palette), que es el conjunto de todos los componentes que podemos usar en nuestra app (más de 50 componentes en la versión 2 de App Inventor). La Paleta se divide en varias categorías, comúnmente llamadas **bandejas** (drawers). Por defecto, la Paleta muestra los componentes de la bandeja "User Interface" (Interfaz de Usuario), pero también podemos acceder a los componentes de otras bandejas clicando en los encabezados etiquetados como "Layout" (Composición), "Media" (Multimedia), "Drawing and Animation", etc.

  Algunos de los componentes de App Inventor son muy sencillos, como el componente "Label" (etiqueta), que simplemente muestra un texto por pantalla, o el componente "Button" (botón), sobre el que pulsamos para iniciar una acción. Otros componentes son más elaborados, como el componente "Canvas" (lienzo), sobre el que podemos dibujar, y el cual puede contener objetos animados (sprites); el "AccelerometerSensor" (acelerómetro), que es un sensor de movimiento que detecta si movemos o

agitamos el dispositivo; o los componentes que crean y reciben mensajes de texto, reproducen música y video, obtienen información de páginas web, etc.

En las apps podemos distinguir dos tipos de componentes: Los **componentes visibles** son aquellos que podemos ver en la pantalla de la app, como los componentes "Label", "Button", "Canvas", o "TextBox". Por otro lado, también hay **componentes no visibles**, como el componente "Sound", que es capaz de reproducir sonidos, o el componente "LocationSensor", que proporciona información de localización (longitud, latitud, altitud, velocidad, etc.). Los componentes no visibles son los objetos que hacen cosas en la app, pero que no se muestran en la interfaz visual del usuario. Como los componentes no visibles no se muestran por pantalla, siempre aparecen en la zona de componentes no visibles del Visor.

- A la derecha del Visor está la lista de componentes que ya hemos añadido a nuestro proyecto. Cualquier componente que arrastremos y coloquemos en el Visor aparecerá en esta lista. Cuando el proyecto está recién empezado, esta lista solo muestra el componente "Screen1", que representa la pantalla inicial de la app.

- En el extremo de la derecha hay una zona que muestra las **propiedades** de los componentes. Cuando pinchemos sobre un componente en el Visor, veremos listadas todas sus propiedades en esta zona. Las propiedades son los detalles de un componente que podemos cambiar. (Por ejemplo, cuando pinchamos sobre un componente "Label" podemos ver las propiedades relacionadas con el color, el texto, el tipo de fuente, etc., de la etiqueta). Al crear un nuevo proyecto solo se muestran las propiedades de la pantalla "Screen1", que incluyen un color de fondo, una imagen de fondo, un título, etc.

- Finalmente, y bajo la lista de componentes, hay una zona llamada "Media" que muestra todos los archivos multimedia (imágenes y sonidos) cargados para el proyecto.

A modo de ejemplo, imaginar que queremos construir una app para jugar a las tres en raya. ¿Cómo deberíamos diseñar la pantalla de esta app? Bueno, probablemente queramos tener una cuadrícula con 9 casillas (3×3). También necesitaremos un mecanismo para permitir al usuario tocar en uno de las nueve casillas de la cuadrícula, y para poner una X o una O en esa casilla (siempre que esté libre). Probablemente también querremos reproducir algunos sonidos y llevar la puntuación.

Para hacer todas estas cosas en App Inventor empezaríamos un nuevo proyecto llamado "TicTacToe". A continuación pondríamos como fondo de pantalla una cuadrícula, junto con algunos botones para detectar qué casilla de la cuadrícula ha tocado el usuario, una etiqueta para la puntuación, y algunos sonidos. Para añadir los botones y la etiqueta acudiríamos a la Paleta, seleccionaríamos estos componentes, y los arrastraríamos al Visor hasta dejarlos en la posición deseada. Para incorporar al proyecto los sonidos de esta app cargaríamos los respectivos archivos de audio en la sección "Media", y para poder reproducirlos añadiríamos uno o varios componentes de sonido, que aparecerían en la zona de componentes no visibles del Visor.

1.3. EL EDITOR DE BLOQUES.

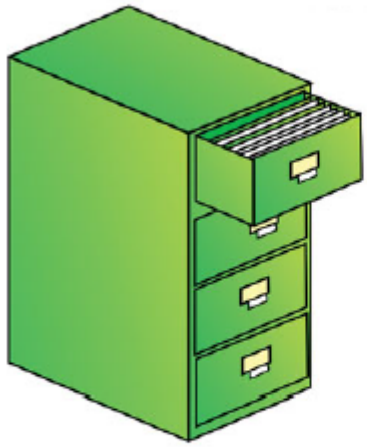
Imaginar que ya hemos diseñado la interfaz de nuestra app, la cual incluye botones, etiquetas, imágenes, etc. ¿Qué hace esta app? Tal y como está ahora mismo, la app no hace nada en absoluto. Nuestra siguiente tarea es decirle a la app lo que debe hacer usando el **Editor de Bloques**.

En el juego de las tres en raya, deberíamos escribir un programa que detecte qué casilla de la cuadrícula ha tocado el usuario, mostrar una X en esa casilla (si el usuario está jugando con las X), permitir un movimiento del jugador contrario, mostrar una O en la casilla correspondiente, y así sucesivamente. También deberíamos detectar si alguien gana o si hay un empate, y reproducir algunos sonidos. Todos estos comportamientos se programan en el Editor de Bloques de App Inventor.

El Editor de Bloques es una interfaz de tipo *arrastra y suelta* (drag and drop), lo que significa que podemos empezar a programar arrastrando y soltando **bloques de código** en el área de trabajo, lo que nos evita tener que aprender una gran cantidad de instrucciones en formato texto. Así pues, en el ejemplo de las tres en raya podríamos arrastrar y soltar bloques de código para detectar cuándo se presiona un botón sin tener que escribir un montón de código. Por supuesto, la programación mediante bloques de código visuales es mucho más sencilla y natural para el programador que la programación textual.

La pantalla del Editor de Bloques se divide en dos pates principales: Una zona a la izquierda con acceso todos los bloques disponibles, y un gran área de trabajo vacía a la derecha.

Como vemos en la figura, el Editor de Bloques es como un armario archivador con bandejas donde residen los distintos bloques de código. En la figura hemos abierto la bandeja "Built-in" → "Math", lo que nos ha dado acceso a una gran cantidad de bloques matemáticos que podemos arrastrar y soltar en el área de trabajo.



Algunas de las bandejas de bloques (como la bandeja "Math") son muy extensas, y tras acceder a ella debemos desplazarnos hacia abajo para ver todos los bloques que incorpora.

En App Inventor distinguimos dos categorías de bloques: Los bloques integrados ("built-in"), y los bloques específicos de componentes ("component-specific"):

- Los **bloques integrados** son estáticos, lo que significa que siempre están disponibles independientemente de la app que estemos creando.
- Los **bloques específicos de componentes** son dinámicos, y aparecen conforme añadimos componentes a nuestra app en el Diseñador. Por ejemplo, si en el Diseñador añadimos un botón, en el Editor de Bloques veremos aparecer una nueva bandeja de bloques relacionados con ese botón.

BLOQUES INTEGRADOS.

He aquí un breve resumen de cada una de las bandejas presente en la categoría de bloques integrados:

Block drawer	Blocks you'll find in there	Explanation and example
Control	`Decision` blocks that tell the program what to do next based on some kind of test `Loops`, which are a powerful way of repeating tasks	`Decision`: If you clicked the Happy button, the phone sends a ☺ tweet. If you clicked Sad, it sends a ☹ tweet. `Loop`: When you text a group of contacts, the phone executes a loop from the first contact to the last, sending a single text each time. Doing this manually would be tedious!
Logic	The logic values `True` and `False` Logical tests: `AND`, `OR`, `NOT`, `=`	`True` and `False` can set properties. For example, setting a button's `Visible` property to `False` makes the button disappear. Logical tests are used for more complicated `Decision` blocks. For example, you might want to make sure the user's age is more than 8 `AND` less than 12.
Math	Blocks that create and compare numbers, plus all the functions you would find on a calculator	For example, in a game you could use `Math` blocks to add up the number of coins collected and convert this to a score. You could also use `Math` blocks to work out whether this score was greater than the current high score.
Text	Blocks that create, change, add, join, split, replace, and compare text	Any piece of text you want to display is created in a `Text` block. For example, you could join the name a user typed and a text message—something like, "Hello Amy, how are you today?"
Lists	Blocks that store, display, and work with lists of things	For example you might want to give the user a menu of choices that they can click.
Colors	A choice of standard colors to use[a]	You can set or change the color of anything you see on screen, like text, a button, or the background.
Variables	Blocks to create variables	A *variable* is a named storage space for a small piece of data like your name or age or a game score.
Procedures	Blocks to create and run procedures	A *procedure* is a kind of mini program that other blocks can start. For example, you might have a procedure that resets the screen whenever a game is over.

a. También podemos mezclar nuestros propios colores usando la carta de colores en:
http://appinventor.mit.edu/explore/ai2/support/blocks/colors.html

BLOQUES ESPECÍFICOS DE COMPONENTES.

Los contenidos del Editor de Bloques cambian dependiendo de los componentes que hayamos añadido en el Diseñador. Por ejemplo, imaginar que hemos creado una interfaz sencilla para una app con solo dos componentes: un botón llamado "PressButton" y un sonido llamado "BeepSound". Cuando despleguemos la bandeja raíz "Screen1" (la cual contiene todas las demás bandejas de bloques específicos de componentes), dentro de ella encontraremos las bandejas "BeepSound" y "PressButton". Estas bandejas incluirán todos los bloques de código relacionados con el componente de sonido "BeepSound" y con el botón "PressButton".

NOTA: En el Editor de Bloques siempre tendremos disponible la bandeja "Screen1". Esto es así porque este componente siempre existe por defecto para todas las apps: Es la pantalla en blanco que se abre al arrancar la app.

1.4. EJECUTAR Y PROBAR APLICACIONES.

En App Inventor podemos probar cada parte de nuestro diseño y de nuestro código para asegurarnos de que funciona como esperábamos, sin necesidad de tener finalizada toda la app. Conforme vamos añadiendo objetos en el Diseñador o vamos definiendo comportamientos en el Editor de Bloques, veremos en tiempo real cómo va quedando la app en nuestro dispositivo móvil o en el emulador.

DISPOSITIVO VS. EMULADOR.

Probablemente todos nosotros dispongamos de un dispositivo Android (un teléfono móvil o una tableta) para hacer pruebas en vivo en App Inventor. Pero también habrá ocasiones en las que no tendremos disponible nuestro teléfono (tal vez porque nos hayamos quedado sin batería) o en las que querremos mostrar una app a alguien que no tenga un dispositivo Android. Para estos casos, App Inventor pone a nuestro servicio el emulador.

El **emulador** es un programa que se ejecuta en nuestro PC, Mac, o máquina Linux, y que simula un teléfono Android real. La principal desventaja es que hay algunas características específicas de los dispositivos Android que no funcionan en el emulador. Algunas de estas características son las siguientes:

- La pantalla táctil (aunque podemos usar el ratón para realizar algunas acciones).
- El GPS.
- Los mensajes y las llamadas.
- La cámara.
- El escáner de códigos de barras y QR.
- El acceso a internet.
- La detección de lo que está ocurriendo físicamente en el dispositivo móvil (como moverlo, inclinarlo, agitarlo, etc.).

1.5. NUESTRA PRIMERA APP: "HELLOWORLD1".

Vamos a crear nuestra primera app. Comenzaremos con algo muy sencillo: Una app que simplemente muestra por pantalla un mensaje emergente que diga "Hello world!". (Es una especie de tradición que los programadores escriban su propia versión del programa "HelloWorld" siempre que empiezan a estudiar un nuevo lenguaje).

Comencemos a construir la app. Tras haber configurado App Inventor en nuestro ordenador, nos conectamos a http://ai2.appinventor.mit.edu para empezar a trabajar. Inmediatamente se abrirá la ventana de proyectos, que en nuestro caso aparecerá vacía. Para crear un nuevo proyecto, pinchamos en el botón "Start new project", escribimos el nombre del proyecto ("HelloWorld1"), y pinchamos en "OK". A continuación se abre la ventana del Diseñador con el proyecto ya creado, pero en blanco.

Recordemos que el Diseñador es el lugar en el que añadimos componentes a nuestra app, y donde diseñamos la apariencia de la pantalla. (Por cierto, que en cualquier momento podemos volver a la lista de proyectos seleccionando "Projects" → "My projects" en el menú superior del Diseñador).

1.6. DISEÑAR LOS COMPONENTES DE "HELLOWORLD1".

AÑADIR UN COMPONENTE NOTIFICADOR.

Los **notificadores** (componentes "Notifier") están presentes en multitud de programas y de apps. Son pequeñas ventanas emergentes que nos dan informaciones o avisos en nuestros teléfonos, consolas, ordenadores, y televisores. En ocasiones solo disponen de un botón de OK, otras veces nos permiten elegir entre OK o Cancel, y a veces nos piden más información mostrando una lista en la que debemos elegir alguna opción, o una caja de texto en la que debemos escribir la información solicitada.

El objetivo de esta app es que, cuando el usuario presione un botón, aparezca una ventana emergente (un notificador) que muestre el mensaje "Hello World!". Para añadir el componente notificador abrimos la bandeja "User Interface", y arrastramos un componente "Notifier" al Visor. Los notificadores son componentes invisibles, por lo que independientemente de dónde lo soltemos, aparecerá en la parte inferior del Visor (área "Non visible components"). Además, veremos que en la lista de componentes el icono del notificador "Notifier1" aparece colgando debajo del icono del componente "Screen1". (Recordemos que "Screen1" es el componente que representa la pantalla de nuestra app, y que siempre está creada por defecto para todas las apps).

1.7. AÑADIR COMPORTAMIENTOS A LOS COMPONENTES DE "HELLOWORLD1".

MOSTRAR EL NOTIFICADOR AL ARRANCAR LA APP.

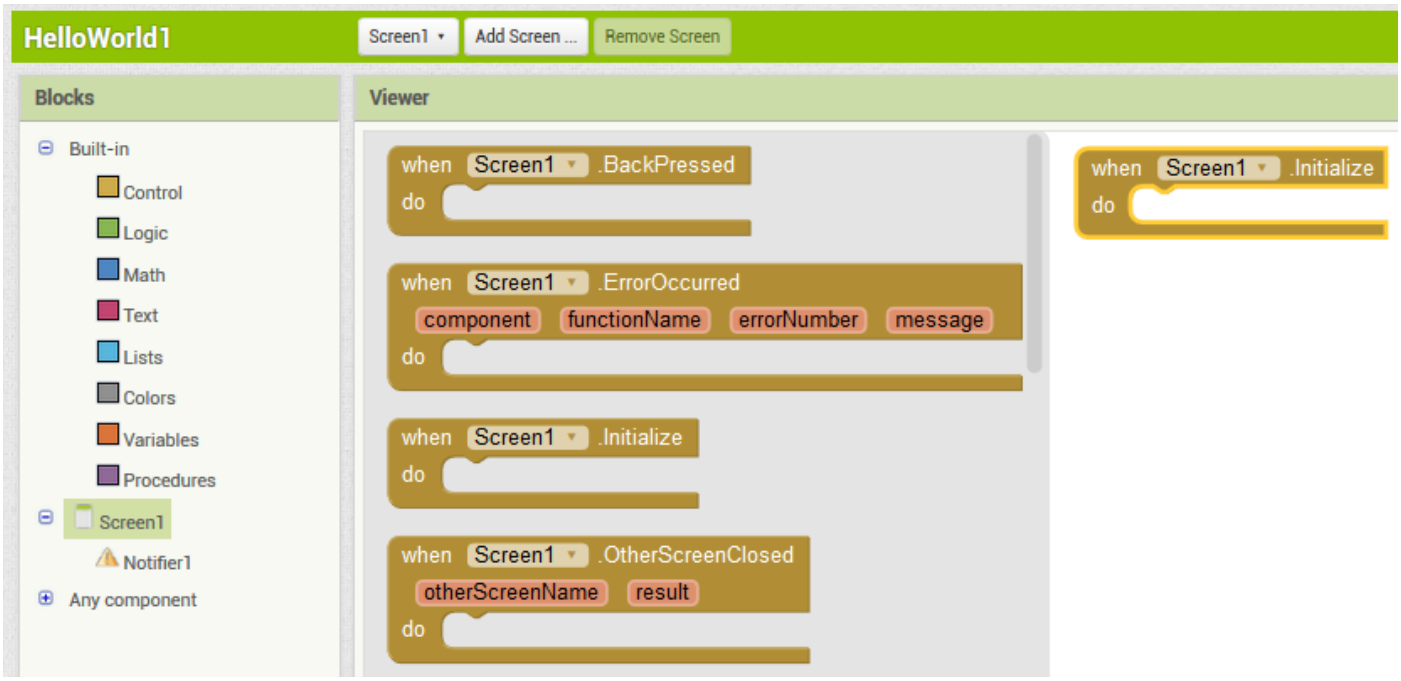
Normalmente las apps no hacen nada hasta que el usuario realiza alguna acción, como presionar un botón. A esto se le llama **evento**. Y a los bloques que detectan la ocurrencia de un evento y hacen algo en respuesta a ello se les denomina **manejadores de eventos**. Los bloques manejadores de eventos son fáciles de reconocer porque comienzan y terminan las palabras reservadas "when" y "do", respectivamente (ver figura).

Después del "when" siempre veremos dos palabras separadas por un punto: La primera palabra es el nombre del componente sobre el que se produce el evento, y la segunda palabra es el evento que el manejador está esperando. A modo de ejemplo, consideremos el manejador de evento "when (Button1).Click do":

Este manejador esperará a que el usuario pulse o clique sobre el botón "Button1", y cuando lo haga, ejecutará los bloques que haya dentro de su sección "do".

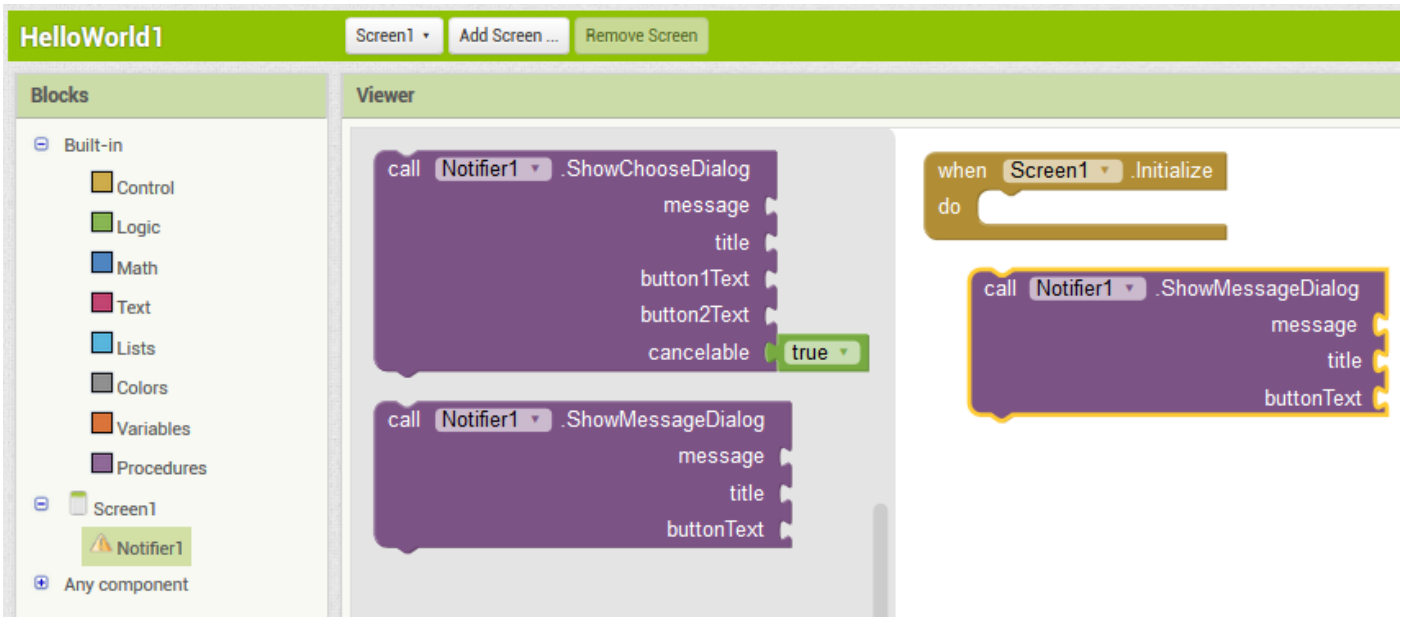
En nuestro caso, queremos que aparezca un notificador nada más arrancar la app. Para ello usaremos un manejador de evento especial llamado "when (Screen1).Initialize". Con este manejador la app ejecutará lo que haya dentro de la sección "do" en cuanto empiece la app, incluso aunque el usuario no haya presionado un botón. Para programar este comportamiento, hacemos lo siguiente:

1) Nos vamos del Diseñador al Editor de Bloques pulsando el botón "Blocks" situado arriba y a la derecha del Diseñador.
2) Acudimos a la bandeja de bloques del componente "Screen1", seleccionamos el manejador de evento "when (Screen1).Initialize" y lo soltamos en el área de trabajo.

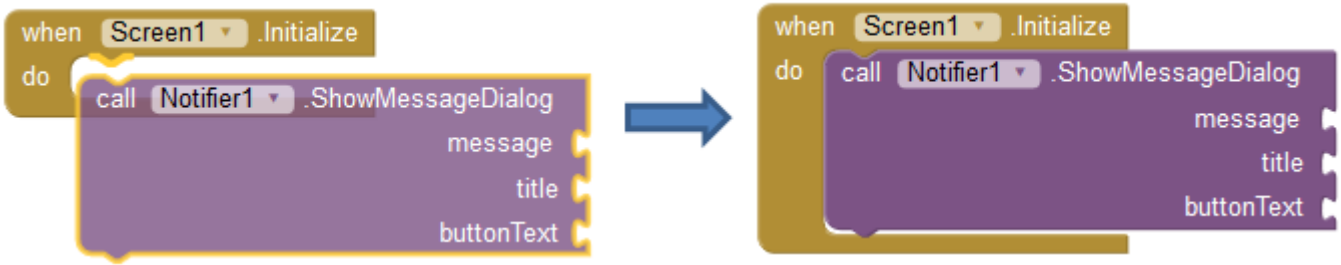


Ahora queremos que la app muestre un notificador que diga "Hello world!" nada más ejecutarse la app. Si seleccionamos la bandeja del componente "Notifier1", veremos que un notificador puede realizar muchas acciones distintas. Pero en este caso usaremos un bloque de función que le permita al notificador mostrar un mensaje de alerta que estará visible hasta que el usuario presione un botón de OK. Este bloque se llama "call Notifier1.ShowMessageDialog". Por consiguiente:

3) Acudimos a la bandeja del componente "Notifier1", seleccionamos el bloque de función "call Notifier1.ShowMessageDialog", y lo arrastramos al área de trabajo:

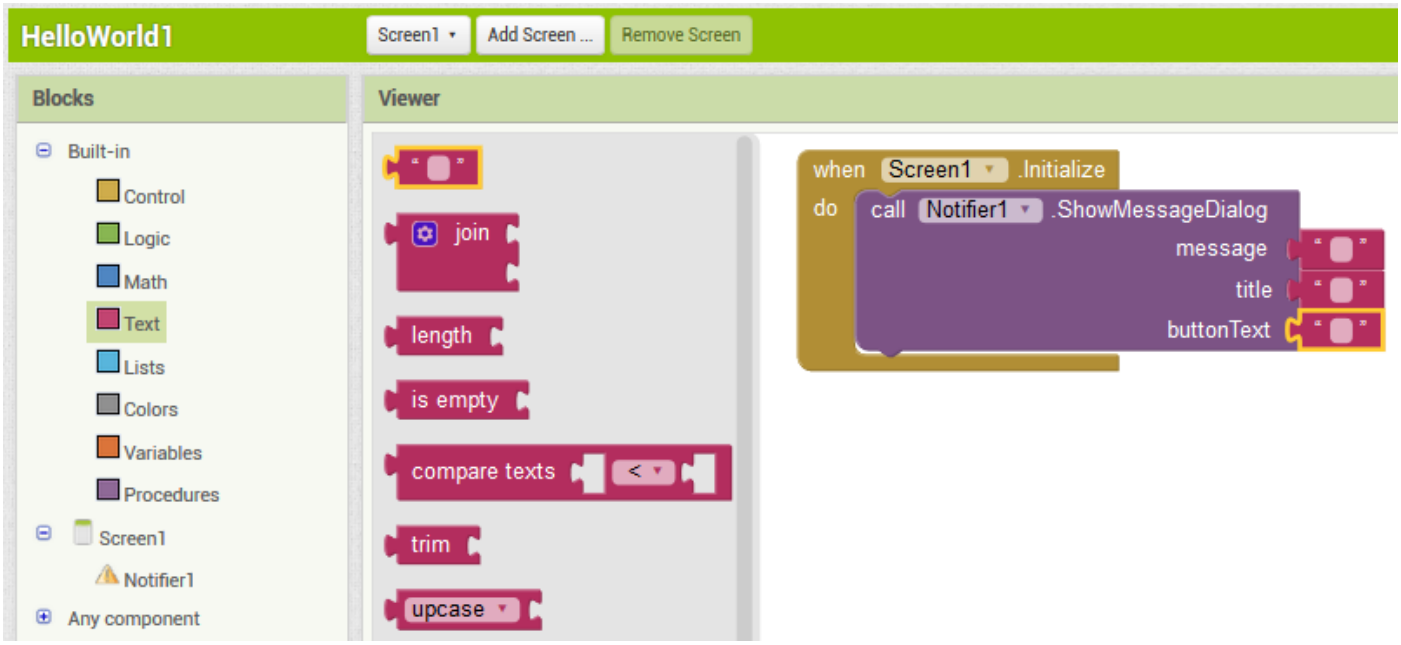


4) Ahora conectamos el bloque "call Notifier1.ShowMessageDialog" dentro de la sección "do" del manejador "when (Screen1).Initialize".

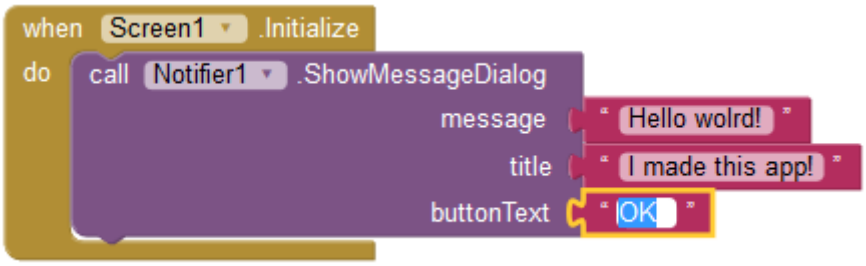


5) El bloque "call Notifier1.ShowMessageDialog" tiene tres ranuras vacías en su lado derecho. Se trata de tres **parámetros** llamados "message", "title", y "button text" cuyo valor debemos indicar para que este bloque funcione correctamente. Los valores de estos tres parámetros son simples textos (también llamados **cadenas**) que especifican el mensaje que mostrará el notificador ("Hello world!"), el título del notificador ("I made this app!"), y el mensaje que aparecerá sobreimpreso en el botón del notificador ("OK"), respectivamente. Para construir estos tres textos, hacemos lo siguiente:

6) En la bandeja "Text" de bloques integrados, arrastramos tres bloques de texto vacíos al área de trabajo, y los conectamos en las ranuras abiertas a la derecha del bloque "call Notifier1.ShowMessageDialog":



7) Ahora editamos los bloques de texto vacíos y escribimos los textos que queremos que muestren, como ilustra la figura:



Y con esto ya hemos terminado. Ahora vamos a probar el funcionamiento de nuestra app.

1.8. PRUEBAS EN VIVO.

Es sorprendentemente fácil cometer un error al programar: Podríamos haber usado un bloque erróneo, olvidado un paso, o incluso no haber conectado dos bloques por unos pocos milímetros. Afortunadamente, con App Inventor podemos probar nuestra aplicación conforme la vamos creando. Si disponemos de un dispositivo Android y de conexión a internet a través de una red WiFi, podemos hacer **pruebas en vivo** (live testing) sin necesidad de descargar ningún software en nuestro ordenador. Si no tenemos un dispositivo Android, necesitaremos hacer algunas configuraciones para usar el emulador. En el caso de que dispongamos de dispositivo, pero no de conexión a una red WiFi, también podemos conectar el dispositivo al ordenador a través de un cable USB. Esta opción también requiere algunas configuraciones. (Las instrucciones para estas configuraciones adicionales podemos encontrarlas en http://appinventor.mit.edu/explore/ai2/setup.html).

OPCIÓN 1.

Si disponemos de un dispositivo Android y de conexión WiFi, debemos hacer lo siguiente:

1) En nuestro dispositivo, descargamos e instalamos la app "MIT AI2 Companion" desde Google Play Store.
2) Conectamos el ordenador y el dispositivo a la misma red WiFi.
3) En App Inventor, seleccionamos "Connect" en el menú superior, y a continuación, elegimos la opción "AI Companion". Una ventana emergente mostrará un código QR (ver figura).
4) En nuestro dispositivo lanzamos la aplicación "MIT AI2 Companion" recién instalada. En la pantalla mostrada en la figura, seleccionamos "Scan QR Code", y después mantenemos el dispositivo sobre el código QR que muestra la pantalla del ordenador, para que el móvil pueda escanearlo.

Si todo ha ido bien, deberíamos ver la aplicación "HelloWorld1" ejecutándose en nuestro dispositivo. A partir de ahora podemos hacer cambios en el Diseñador de App Inventor o en el Editor de Bloques, y esos cambios también se reflejarán en el dispositivo.[1]

OPCIÓN 2.

Si no disponemos de un dispositivo Android debemos usar el emulador. Para iniciarlo, hacemos lo siguiente:

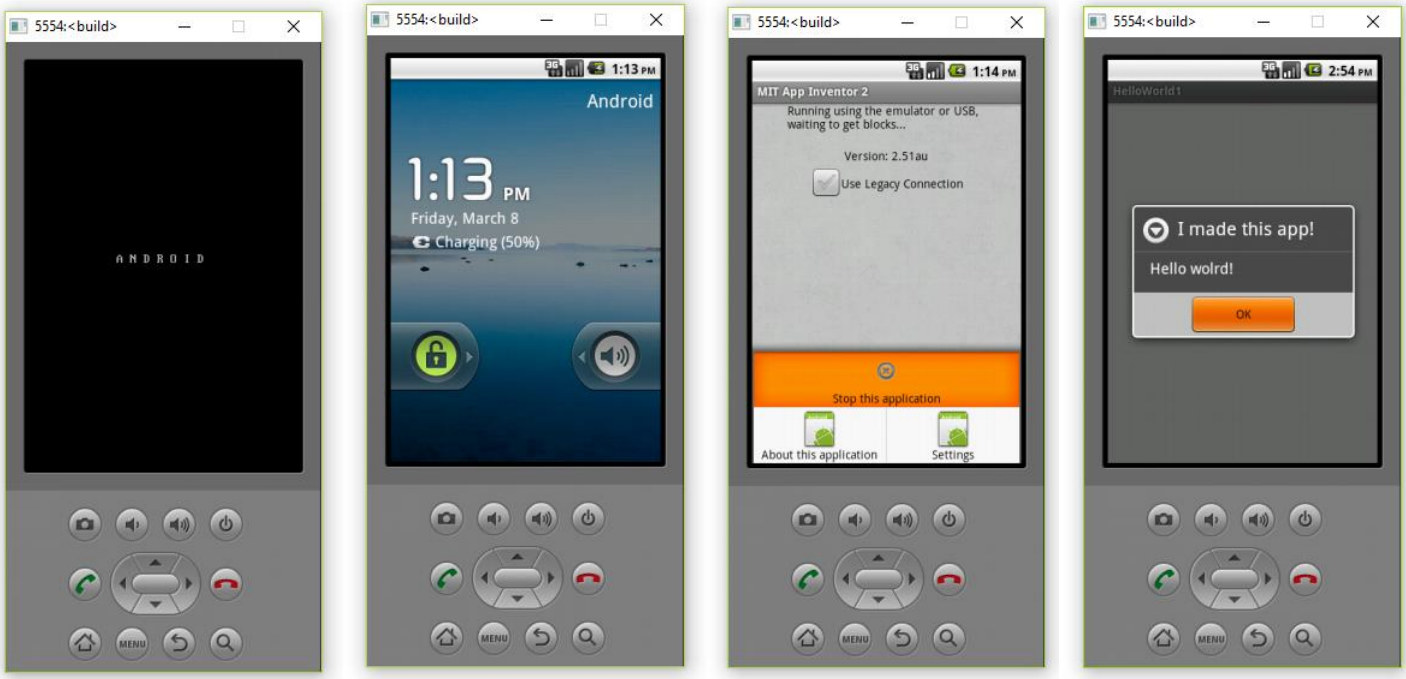
1) Nos conectamos a la web http://appinventor.mit.edu/explore/ai2/setup.html, y clicamos en el enlace "Instructions" de la opción 2. Ello nos redirigirá a la web http://appinventor.mit.edu/explore/ai2/setup-emulator.html, donde debemos seleccionar el Sistema Operativo de nuestro ordenador. En el caso habitual de que nuestro ordenador use Windows, llegaremos a la web http://appinventor.mit.edu/explore/ai2/windows.html, en la que debemos descargar el instalador del software de configuración de App Inventor (por ejemplo, "MIT_App_Inventor_Tools_2.3.0_win_setup.exe"), y seguir las instrucciones para su instalación.

[1] Si tenemos problemas para inicializar las pruebas en nuestro dispositivo Android, visitar http://appinventor.mit.edu/explore/ai2/setup.html.

2) Una vez instalado este software, lanzamos el programa "aiStarter" en el ordenador. Este programa es el software que le permite al navegador comunicarse con el emulador (y con un dispositivo Android a través del cable USB). Al lanzar este programa, veremos que se abre una ventana de consola de comandos.

3) Volvemos a App Inventor, y en el menú "Connect" seleccionamos la opción "Emulator". Una ventana emergente nos informará de que App Inventor se está conectando al emulador. Arrancar el emulador puede llevar un par de minutos. Si abrimos la nueva ventana de consola de comandos que ha aparecido, veremos que se va llenando de instrucciones de conexión.

4) A continuación aparece el emulador, inicialmente con la pantalla en negro. Poco a poco, el emulador se va cargando: Inicializa la pantalla, y prepara su tarjeta SD. Una vez conectado, el emulador lanzará y mostrará por pantalla la app que tenemos abierta en App Inventor (ver figura).



OPCIÓN 3.

Por último, si disponemos de un dispositivo Android pero no de conexión a una red WiFi, debemos conectar el ordenador al dispositivo a través de un cable USB:

1) Nos conectamos a la web http://appinventor.mit.edu/explore/ai2/setup.html, y clicamos en el enlace "Instructions" de la opción 3. Ello nos redirigirá a la web http://appinventor.mit.edu/explore/ai2/setup-device-usb.html. El primer paso es instalar el software de configuración de App Inventor, que incluye la aplicación "aiStarter" que le permite al navegador comunicarse con un dispositivo Android a través del cable USB. El segundo paso es instalar en el dispositivo Android la aplicación "MIT AI2 Companion" desde Play Store.
2) Lanzamos a aplicación "aiStarter" en el ordenador. Al lanzar este programa, veremos que se abre una ventana de consola de comandos.
3) Configuramos nuestro dispositivo para la comunicación vía USB. Para ello, debemos acudir a los Ajustes del Sistema de nuestro dispositivo Android, y allí activamos las Opciones de Desarrollador, asegurándonos de que la Depuración de USB está habilitada.

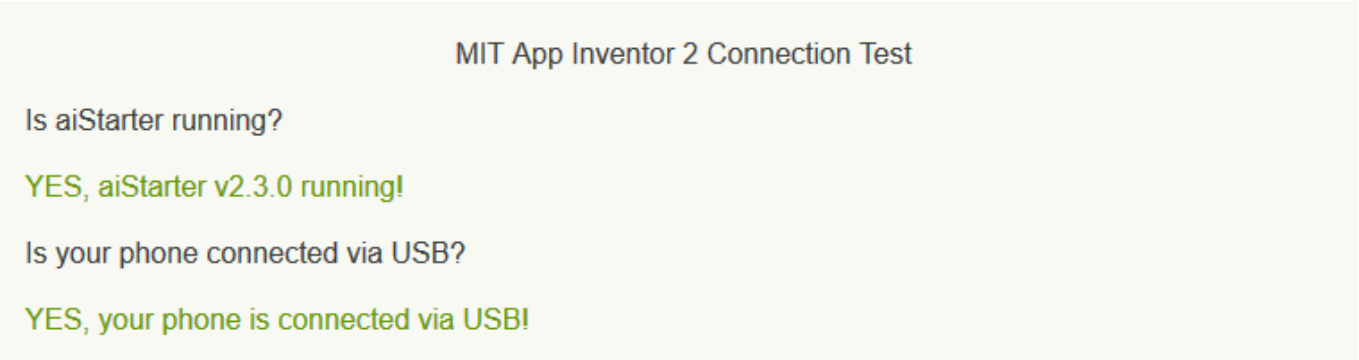
   En la mayoría de dispositivos Android 3.2 o anteriores, podemos encontrar estas opciones en "Ajustes" → "Aplicaciones" → "Desarrollo". En dispositivos Android 4.0 y posteriores, están en "Ajustes" → "Opciones de desarrollador". (Sin embargo, en dispositivos Android 4.2 y posteriores, las "Opciones de

desarrollador" están ocultas por defecto. Para hacerlas visibles, vamos a "Ajustes" → "Acerca del teléfono", y tocamos en el "número de compilación" de serie siete veces. Volvemos a la pantalla previa para encontrar las "Opciones de desarrollador", incluyendo la Depuración de USB).

4) Conectamos el dispositivo al ordenador mediante el cable USB, asegurándonos que el dispositivo se conecta como un "dispositivo de almacenamiento masivo" (y no como un "dispositivo multimedia"), y que no se monta como una unidad adicional en el ordenador. Esto significa que, en el ordenador, debemos ir a "Mi PC" o "Equipo", y desconectar cualquier unidad que se haya montado cuando conectamos el dispositivo. (En algunos dispositivos basta con no permitir que el ordenador tenga acceso a los datos del teléfono en la ventana emergente que aparece nada más conectar el teléfono).

   En dispositivos Android 4.2.2 y posteriores, el dispositivo abrirá una ventana con el mensaje "¿Permitir depuración USB?" la primera vez que lo conectemos. Seleccionamos "OK", lo cual autentificará el ordenador al dispositivo, permitiéndole al ordenador comunicarse con él.

5) Comprobamos la conexión. Para ello vamos a la página http://appinventor.mit.edu/test, y comprobamos si el ordenador puede detectar el dispositivo. Si este test falla, acudimos a http://appinventor.mit.edu/explore/ai2/connection-help.html y consultamos la ayuda sobre conexión USB para nuestro ordenador (Windows o Mac). (Esto podría implicar tener que instalar unos ciertos drivers en el ordenador).



6) Una vez comprobado que la conexión se ha establecido correctamente, volvemos a App Inventor, y seleccionamos "Connect" → "USB". Tras unos segundos, la aplicación aparecerá en la pantalla de nuestro dispositivo.

NOTA IMPORTANTE:

En ocasiones ocurre que el tester de conexión de App Inventor (http://appinventor.mit.edu/test) falla, y ni siquiera detecta que la aplicación "aiStarter" esté funcionando, aunque la hayamos lanzado previamente. En tales casos nos intentamos conectar por USB ignorando este hecho, y la mayoría de las veces la conexión funcionará sin más problemas. Sin embargo, habrá otras ocasiones en las que la conexión por USB falla, y enntences debemos proceder como indicamos:

- Ejecutamos la aplicación "aiStarter" en nuestro ordenador.
- Abrimos el administrador de tareas, e intentamos localizar el proceso "adb.exe".
- De no estar presente, o de no estar funcionando, forzamos su ejecución. Para ello, acudimos a la carpeta donde se localiza la aplicación "aiStarter.exe" en nuestro ordenador (probablemente, C:/Archivos de programa (x86)/AppInventor). Dentro de ella, habrá otra subcarpeta llamada "commands-for-appinventor". Allí encontraremos el archivo "adbrestart.bat", que debemos lanzar para forzar la ejecución de este proceso.
- Tras esto, volvemos a App Inventor, e intentamos conectarnos al dispositivo vía USB. El problema debería haberse solucionado.

Cuando finalmente (y por cualquiera de las tres opciones) nuestra app aparezca en el dispositivo o en el emulador, deberíamos ver que el notificador con el mensaje "Hello world!" aparece en pantalla.

Notar que al pulsar en el botón de OK del notificador, la pantalla se queda en blanco. Esto es así porque el notificador sabe que debe desaparecer al pulsar en este botón, y no le hemos dicho a la app que haga nada más cuando esto ocurre. Para quitar la app, pulsamos en el botón de menús del móvil (o del emulador), seleccionamos la opción "Stop this application", y después elegimos "Stop and Exit" (ver figuras). Si queremos ejecutar la aplicación "HelloWorld1" por segunda vez, debemos quitar la app del dispositivo o del emulador, resetear la conexión (seleccionando la opción "Connect" → "Reset conecction" en el menú superior del App Inventor), y reiniciar todo el proceso de pruebas en vivo. Esto es ciertamente engorroso e ineficiente. ¿Cómo podemos subsanar este problema? Como ya hemos comentado antes, App Inventor es un lenguaje de programación orientado a eventos. Esto significa que la app está constantemente esperando a que ocurra un evento. Las apps pueden estar vigilando la ocurrencia de múltiples eventos distintos, por lo que podríamos escribir una app que compruebe todo el tiempo si el usuario ha pulsado un botón, si ha agitado el teléfono, o si el teléfono ha recibido un mensaje de texto, y la app podría responder de forma distinta a cada uno de estos eventos.

El problema con la app "HelloWorld1" es que el evento "Screen1.Initialize" que hemos usado para hacer funcionar a la app solo ocurre una vez: nada más arrancar la app. Para solventar este defecto, vamos a crear una segunda versión de la app en la que el usuario deba presionar un botón para que la app muestre el notificador con el mensaje "Hello world!".

1.9. COMENZAR CON LA APP "HELLOWORLD2".

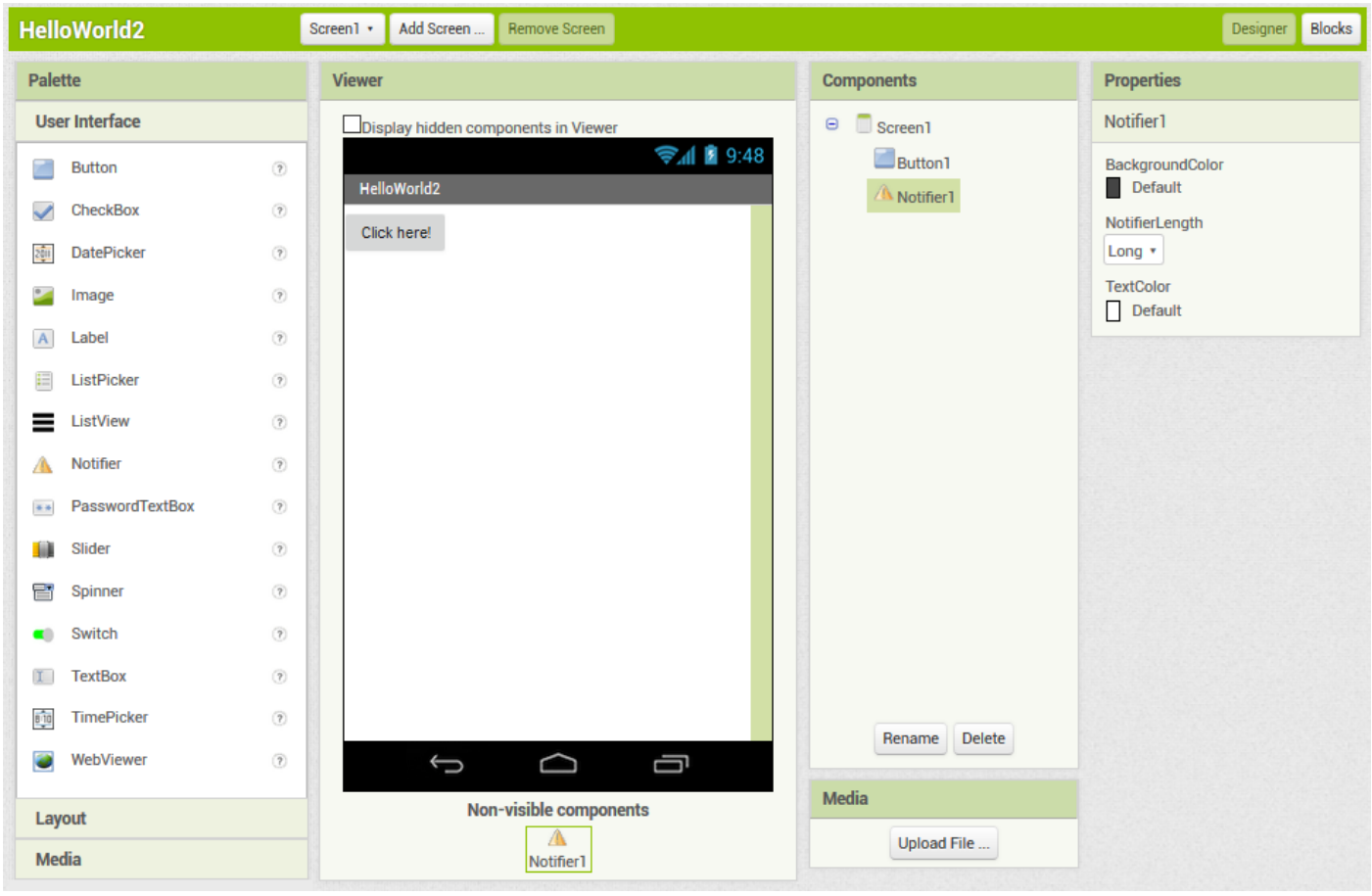
Añadir un botón a la app "HelloWorld1" es muy sencillo. Pero como esta segunda versión de la app será muy parecida a la primera, es muy recomendable guardar una copia de la app "HelloWorld1" para empezar a trabajar basándonos en ella (en vez de construirlo todo de nuevo partiendo de cero). Para ello, vamos al menú superior de App Inventor y seleccionamos la opción "Projects" → "Save Project as…". En la ventana emergente guardamos la copia de la app "HelloWorld1" como "HelloWorld2", y al presionar en el botón OK, se abrirá esta segunda versión del proyecto (de momento, una nueva versión idéntica a la primera).

1.10. DISEÑAR LOS COMPONENTES DE "HELLOWORLD2".

AÑADIR UN BOTÓN.

Para añadir un botón a la app "HelloWorld2", seguimos estos pasos:

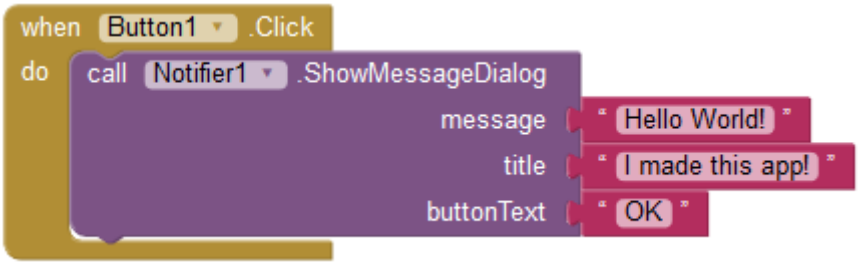
1) En la vista de Diseñador acudimos a la Paleta de componentes, y en la bandeja "User Interface" seleccionamos y arrastramos un componente "Button" al Visor.
2) Veremos que el botón incluye el texto "Text for Button1". Queremos cambiar este texto a "Click Here!". Para ello seleccionamos el botón (en el Visor o en la lista de componentes), y accedemos a su lista de propiedades. Para cambiar el texto sobreimpreso en el botón, editamos su propiedad "Text" escribiendo el texto deseado en la caja de texto destinada a tal efecto.
3) En el Visor, deberíamos ver que el texto impreso sobre el botón cambia a "Click here!".



1.11. AÑADIR COMPORTAMIENTOS A LOS COMPONENTES DE "HELLOWORLD2".

En la app previa ya escribimos los bloques necesarios para mostrar el notificador. Lo único que debemos cambiar es el evento que dispara esta acción. Para ello, en vez de usar el evento "Screen1.Initialize", vamos a usar el evento "Button1.Click" asociado al botón recién agregado. Para ello:

1) Nos vamos al Editor de Bloques.
2) Seleccionamos la bandeja "Button1" para acceder a los bloques de código asociados a este botón.
3) Tomamos y arrastramos el manejador de evento "when (Button1).Click" y lo soltamos en el área de trabajo.
4) Desconecamos el bloque "call (Notifier1).ShowMessageDialog" del manejador "when (Screen1).Initialize", y lo conectamos al manejador "when (Button1).Click". Una vez hecho esto, borramos el bloque "when (Screen1).Initialize" que ha quedado vacío. El programa final debería quedar como muestra la figura:



Y con esto ya hemos terminado. Ahora vamos a probar la app: Si pulsamos en el botón "Click here!" emerge el notificador con el mensaje "Hello World!". Si ahora pulsamos en el botón de OK del notificador, volvemos a la pantalla con el botón "Click here!", que nos permite volver a lanzar el notificador.

1.12. MODIFICACIONES A "HELLOWORLD2".

Después de construir las apps propuestas en cada capítulo, probablemente se nos ocurrirán varias formas de mejorarlas. Al final de cada app, en ocasiones se nos propondrán algunas ideas que podemos implementar. El hecho de personalizar las aplicaciones guiadas nos obligará a explorar los distintos componentes y bloques disponibles, lo cual nos permitirá aprender a programar por nosotros mismos sin instrucciones detalladas que nos constriñan.

Algunas mejoras que podemos probar con esta app son las siguientes:

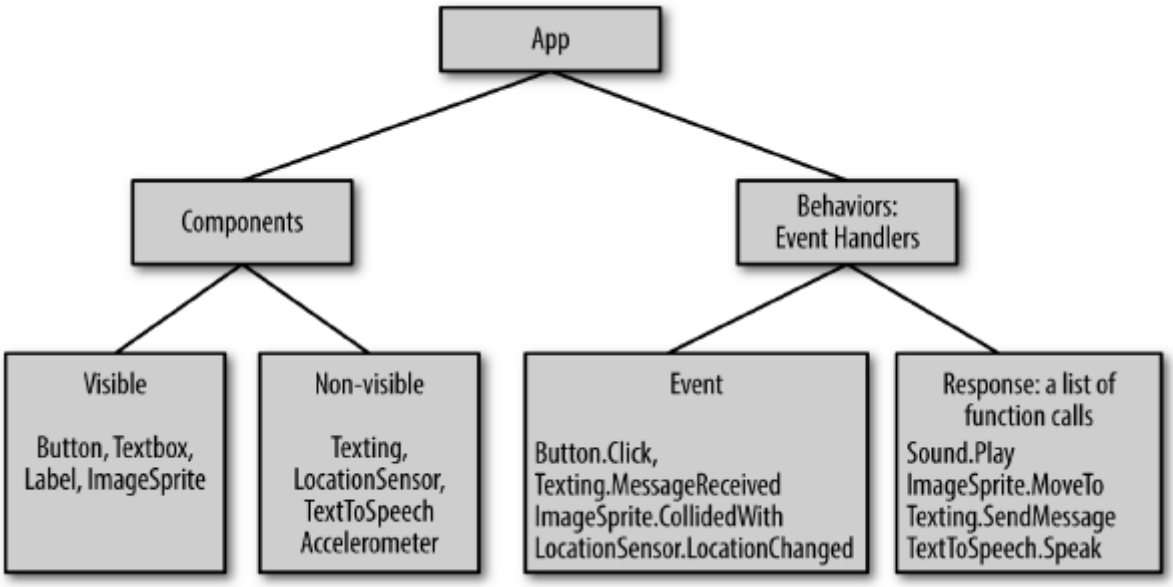
- Añadir un nuevo botón y otro notificador a la app. Hacemos que el segundo notificador diga "Goodbye world!" si el usuario pulsa ese segundo botón.
- Probar a cambiar las propiedades de color de "Screen1" y "Button1". También podemos probar a cambiar las propiedades "Font" y "Size" de "Button1".

1.13. LA ARQUITECTURA DE UNA APP.

Una vez construida nuestra primera app, vamos a concluir este capítulo examinando la estructura de una aplicación desde el punto de vista del programador. Como veremos, entender cómo funciona una app desde la perspectiva del programador es mucho más comlicado que desde el punto de vista del usuario, porque las apps tienen una estructura interna que debemos comprender para poder crearlas de manera correcta.

Una forma habitual de describir la estructura de una app es dividirla en dos partes: sus **componentes** y sus **comportamientos**. Esta división se corresponde aproximadamente con las dos ventanas principales de App Inventor: En App Inventor usamos el Diseñador de Componentes para especificar los objetos (los componentes) que constituyen la app, y después usamos el Editor de Bloques para programar la forma en la que la app responde a las acciones del usuario y a eventos externos (los comportamientos).

La figura muestra un resumen de la arquitectura general de una app. En esta sección exploraremos esta arquitectura en detalle.



COMPONENTES.

Como ya hemos comentado, existen dos tipos de componentes en una app: los visibles y los no visibles.

Los **componentes visibles** son aquellos que podemos ver cuando ejecutamos la app. Algunos ejemplos son los botones, las cajas de texto, las etiquetas, etc. A veces nos referimos a ellos como la *interfaz de usuario*.

Los **componentes no visibles** son aquellos que no podemos ver, por lo que no forman parte de la interfaz de usuario. Su misión principal es proporcionar acceso a las funcionalidades internas del dispositivo. Por ejemplo, el componente "Texting" envía y procesa mensajes de texto SMS, el componente "LocationSensor" determina la localización del dispositivo, y el componente "TextToSpeech" convierte textos en voz.

Tanto los componentes visibles como los no visibles están defindos por un conjunto de propiedades. Las **propiedades** son como posiciones de memoria para almacenar información acerca de un componente específico. Los componentes visibles como los botones y las etiquetas tienen propiedades tales como "Width" (anchura), "Height" (altura), y "Alignment" (alineación) que definen la apariencia del componente en pantalla.

Podemos imaginar que las propiedades de un componente son como las celdas de una hoja de cálculo: Podemos especificar el valor que contienen en el Diseñador para definir la apariencia *inicial* de ese componente, pero también podemos cambiar sus valores en el Editor de Bloques mediante código.

COMPORTAMIENTO.

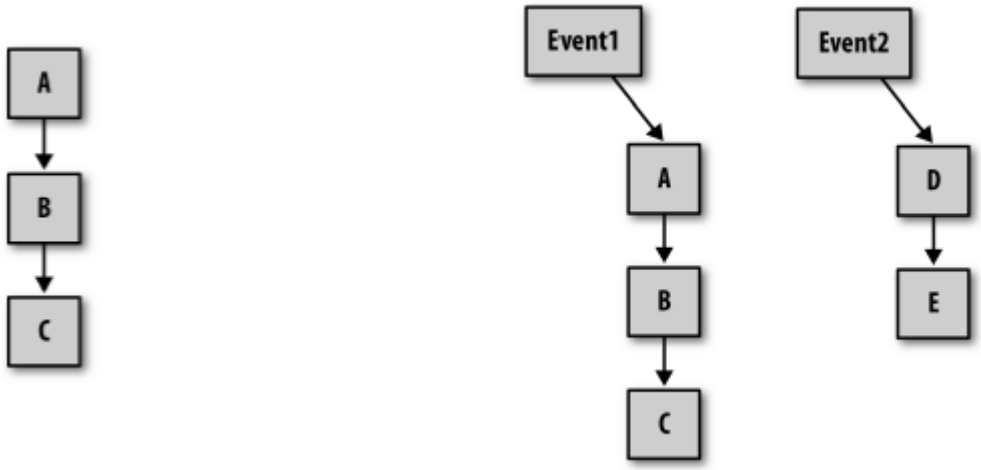
Los componentes de una app son fáciles de entender: Una caja de texto sirve para que el usuario pueda insertar información, un botón sirve para pulsarlo, etc. Por otro lado, el comportamiento de la app es conceptualmente más complicado y difícil de especificar. El comportamiento define la forma en la que la app debe responder a la ocurrencia de **eventos**, tanto los inicializados por el usuario (por ejemplo, la

pulsación de un botón) como aquellos con un origen externo (por ejemplo, la recepción de un SMS en el dispositivo). La razón por la que la programación es más complicada reside en la dificutad de especificar este comportamiento interactivo.

Afortunadamente, App Inventor proporciona bloques de código de alto nivel para especificar comportamientos. Al contrario de lo que ocurre en la programación tradicional, que implica tener que aprender y escribir grandes cantidades de código, la programación basada en bloques hace que programar comportamientos sea parecido a encajar piezas de un puzle. Y además, App Inventor está diseñado para hacer que la programación de respuestas a la ocurrencia de eventos sea especialmente sencilla. Las siguientes secciones proporcionan un modelo para tratar de explicar el comportamiento de una app, y cómo podemos programarlo en App Inventor.

PROGRAMACIÓN SECUENCIAL.

Un programa tradicional sigue una secuencia lineal de instrucciones que el ordenador debe ejecutar, como muestra la figura de la izquierda. Por ejemplo, un software típico podría comenzar con una transacción bancaria (A), seguir con una serie de cálculos y una modificación de la cuenta de un cliente (B), y finalizar mostrando por pantalla el nuevo balance de la cuenta (C).



Programación secuencial VS. programación orientada a eventos.

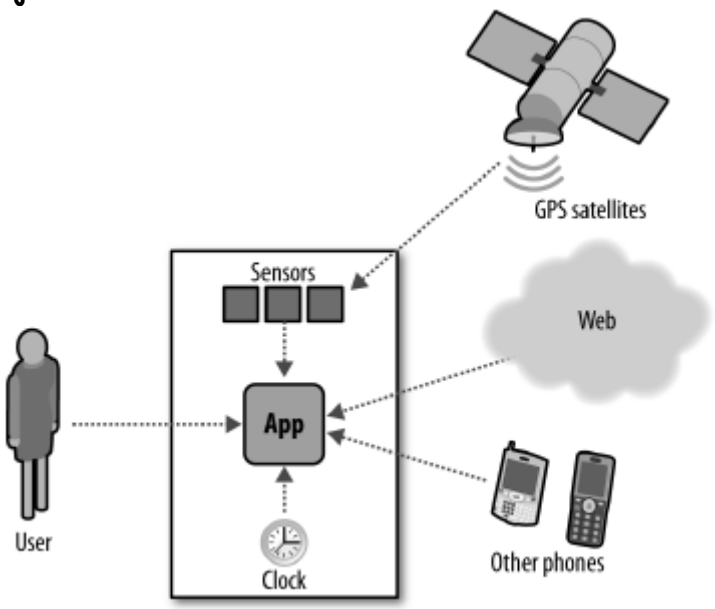
PROGRAMACIÓN ORIENTADA A EVENTOS.

La programación secuencial es el paradigma bajo el que funcionaban los programas de los primeros ordenadores, pero no es adecuada para las apps de los teléfonos móviles, para Internet, y en general, para la mayor parte del software que se desarrolla hoy en día. En efecto, los programas actuales no se limitan a realizar un conjunto de instrucciones en un orden predeterminado. En vez de eso, los programas reaccionan ante **eventos** (habitualmente, iniciados por el usuario de ese software). Por ejemplo, si el usuario presiona un botón, el programa responde realizando alguna operación (por ejemplo, enviar un mensaje de texto). Pensemos en los dispositivos táctiles actuales: La acción de arrastrar nuestro dedo a través de la pantalla constituye un evento, y una app podría responder a este evento dibujando una línea desde el punto donde nuestro dedo tocó la pantalla inicialmente hasta el punto donde dejó de tocarla.

Las apps modernas pueden verse como máquinas de respuesta a eventos. Estas apps también incluyen secuencias de instrucciones, pero cada una de estas secuencias solo se ejecuta como respuesta a un cierto evento, como muestra la figura.

Conforme van ocurriendo eventos, la app reacciona llamando a una secuencia de funciones. Las **funciones** son aquellas cosas que podemos hacer con (o hacerle a) un componente. Estas cosas podrían ser operaciones como enviar un mensaje SMS, o cambiar los valores de las propiedades de un componente (por ejemplo,

cambiar el texto de una etiqueta). A la acción de ejecutar una función se le denomina *llamar* (o *invocar*) a la función. Para definir a qué funciones se llaman en respuesta a un cierto evento, los programadores utilizan unos bloques llamados **manejadores de eventos**.

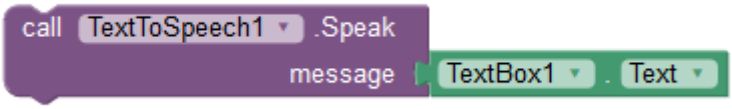


Muchos eventos son inicializados por el usuario final de la app, pero hay otros que no. Una app puede reaccionar ante eventos que ocurren dentro del teléfono (como cambios en su sensor de orientación o en su reloj interno), o puede responder ante eventos originados fuera del teléfono (como la recepción de un mensaje de texto o de una llamada de voz desde otro teléfono, o la descarga de datos desde la web), ver figura.

Una de las razones por las que la programación en App Inventor es tan intuitiva es que está basada en el paradigma de respuesta ante eventos. En la programación orientada a eventos definimos un comportamiento arrastrando bloques manejadores de eventos, con la forma "when <event> do". Por ejemplo, consideremos una app llamada "SpeakIt" que responde a la pulsación de un botón convirtiendo a voz el texto que el usuario haya escrito en una caja de texto. Esta aplicación podría programarse con un solo manejador de eventos como muestra la figura:

Estos bloques especifican que cuando el usuario pulse el botón llamado "SpeakItButton", el componente "TextToSpeech1" debería leer en voz alta el texto que el usuario ha escrito en la caja de texto llamada "TextBox1". La respuesta al evento "SpeakItButton.Click" es la llamada a la función "TextToSpeech1.Speak".

En App Inventor toda actividad siempre ocurre como respuesta a un evento. Nuestra app no debería contener bloques fuera de un manejador de evento. Por ejemplo, los bloques mostrados en la figura no hacen absolutamente nada cuando los dejamos solos:



TIPOS DE EVENTOS.

Los distintos eventos que pueden ocurrir en una app se clasifican en las categorías listadas en la tabla:

Event type	Example
User-initiated event	*When* the user clicks `button1`, *do*...
Initialization event	*When* the app launches, *do*...
Timer event	*When* 20 milliseconds passes, *do*...
Animation event	When two objects collide, *do*...
External event	*When* the phone receives a text, *do*...

Eventos iniciados por el usuario.

Los eventos iniciados por el usuario son los más comunes. Por ejemplo, en los formularios de registro de datos lo normal es que el usuario presione un botón para desencadenar una respuesta en la app (por ejemplo, la grabación de los datos insertados en una base de datos). Por otro lado, las apps gráficas responden a toques o deslizamientos de los dedos del usuario sobre la pantalla táctil.

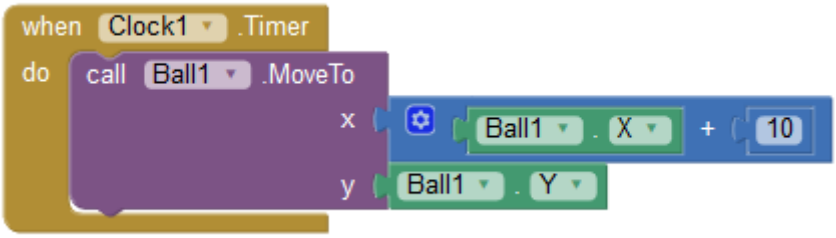
Eventos de inicialización.

En ocasiones necesitaremos que la app realice ciertas operaciones nada más arrancar, y no en respuesta a acciones del usuario u otro tipo de eventos. Pero, ¿cómo encaja esto con el paradigma de programación orientada a eventos?

Los lenguajes de programación orientados a eventos como App Inventor consideran la ejecución de la app como un evento más. Si queremos ejecutar ciertas operaciones nada más abrir la app, debemos arrastrar un bloque manejador de evento "when Screen1.Initialize", y ubicar dentro de él los bloques de llamada a las funciones pertinentes.

Eventos de temporización

En algunas apps hay ciertas operaciones deben ejecutarse conforme pasa el tiempo. Por ejemplo, hay algunas animaciones que se implementan mediante objetos que se mueven controlados por un temporizador. App Inventor incluye un componente "Clock" que podemos usar para activar eventos asociados al disparo de un temporizador. Por ejemplo, si queremos que una pelota se mueva horizontamente 10 píxeles cada vez que pase una cierta cantidad de tiempo especificada en un componente "Clock", deberíamos escribir un programa similar al de la figura:



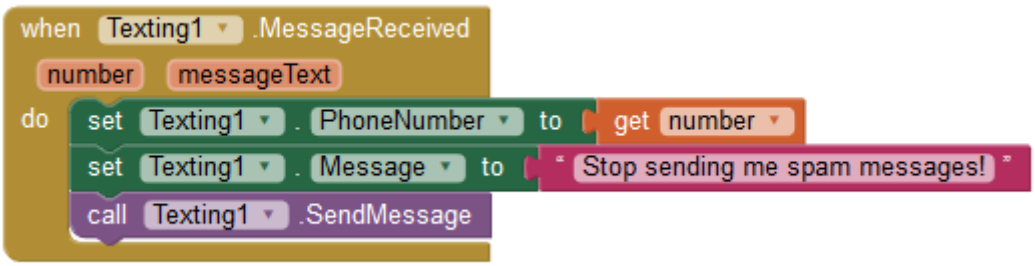
Eventos de animación.

Hay ciertos eventos que son activados por los objetos gráficos (sprites) contenidos en los lienzos (canvas) de las apps. Estos eventos nos permiten programar videojuegos y otras animaciones interactivas especificando que debería pasar cuando ocurren eventos como la llegada del sprite al borde del lienzo en el que se mueve, o cuando dos sprites colisionan uno contra otro, ver figura:

Eventos externos.

Cuando nuestro teléfono recibe información de localización actualizada en su sensor GPS se dispara un evento. Análgamente, cuando nuestro dispositivo recibe un mensaje de texto, también se dispara otro evento, ver figura.

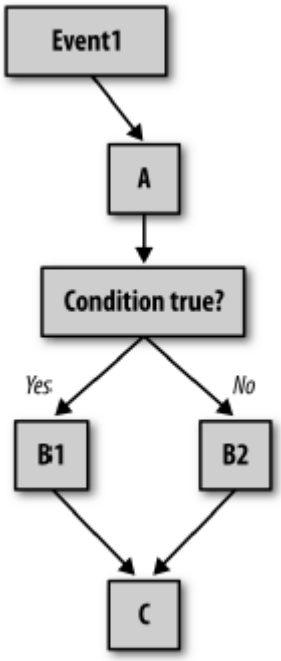
Este tipo de recepciones de información externas al dispositivo son eventos igual de válidos que la pulsación de un botón por parte de un usuario.



En general, las apps que nosotros crearemos aquí contendrán varios manejadores de eventos: uno para inicializar las cosas, otros para responder a entradas o acciones del usuario, algunos activados por temporizadores, y otros tantos disparados por eventos externos. Nuestra tarea como programadores será diseñar nuestra app según este enfoque, y definir su respuesta ante la ocurrencia de estos eventos.

LOS MANEJADORES DE EVENTOS PUEDEN HACER PREGUNTAS.

Las respuestas a los eventos no son siempre secuencias lineales de instrucciones; en ocasiones también formulan preguntas y repiten la ejecución de ciertas operaciones. Por "formular una pregunta" nos referimos a que la app consulta algunos datos que ha almacenado previamente y en función de su valor determina qué conjunto de instrucciones ejecuta a continuación (bifurcación). La figura ilustra un ejemplo de bifurcación.



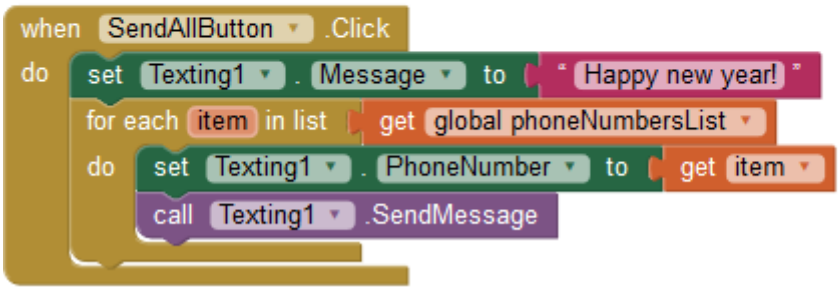
En el diagrama vemos que, cuando ocurre el evento 1, la app realiza la operación A y a continuación comprueba una condición. Si la condición es cierta se ejecuta la operación $B1$, y si es falsa se ejecuta la operación $B2$. En cualquier caso, la app continúa ejecutando la operación C.

Las comprobaciones que hacen las apps son preguntas del tipo "¿La puntuación ha llegado a 100 puntos?", o "¿El mensaje que he recibido lo ha enviado Ana?". Estas comprobaciones también pueden ser fórmulas más complejas que incluyan mútiples operadores de comparación (mayor que, menor que, igual que) y operadores lógicos (and, or, y not).

En App Inventor podemos especificar estos comportamientos condicionales usando los bloques "if" e "if – else". Por ejemplo, el programa de la figura muestra el mensaje "You win!" si la puntuación del jugador llega a los 100 puntos.

LOS MANEJADORES DE EVENTOS PUEDEN REPETIR LA EJECUCIÓN DE CIERTOS BLOQUES.

Además de hacer preguntas y bifurcar el flujo de ejecución en función de la respuesta, las apps también pueden reaccionar a un evento repitiendo múltiples veces la ejecuación de ciertas operaciones. App Inventor incorpora algunos bloques para realizar estas repeticiones, incluyendo los bloques "for each" y "while do". Ambos bloques deben contener otros bloques dentro de ellos. Todos los bloques dentro de un bloque "for each" se ejecutan una vez por cada uno de los elementos de una lista. Por ejemplo, si quisiéramos enviar el mismo mensaje a una lista de números de teléfono, podríamos usar el programa mostrado en la figura:



En este caso, la ejecución de los bloques dentro del "for each" se repite tres veces, porque la lista "phoneNumbersList" tiene tres elementos. En este ejemplo, la app envía el mensaje "Happy new year!" a los tres números de teléfono de la lista.

LOS MANEJADORES DE EVENTO PUEDEN RECORDAR COSAS.

Como un manejador de evento ejecuta ciertos bloques en respuesta a un evento, a menudo necesitará guardar información. La información puede almacenarse en posiciones de memoria llamadas **variables**, que podemos definir en el Editor de Bloques. Las variables son como las propiedades de los componentes, con la diferencia de que no están asociadas a un componente en particular. A modo de ejemplo, en un videojuego podríamos definir una variable llamada "score" (puntuación), y nuestros manejadores de eventos deberían modificar su valor cuando el usuario haga algo relacionado con ella. Las variables almacenan datos temporalmente mientras la aplicación está ejecutándose; cuando cerramos la app, los datos se pierden y dejan de estar disponibles.

En ocasiones, nuestra app necesitará recordar datos no solo en tiempo de ejecución, sino también cuando la cerremos y volvamos a abrirla. Si queremos guardar las puntuaciones más altas de un videojuego, deberíamos almacenar estos datos de forma que estén disponibles la próxima vez que alguien juegue una partida. Los datos que se guardan incluso después de cerrar la app se llaman **datos persistentes**, y no se

pueden almacenar en variables (porque se borran al cerrar la app), sino que deben grabarse en algún tipo de base de datos.

LOS MANEJADORES DE EVENTOS PUEDEN INTERACTUAR CON LA WEB.

Algunas apps solo utilizan la información presente dentro del dispositivo. Pero otras muchas también se comunican con la Web, ya sea mostrando una página web dentro de la app, o bien enviando solicitudes a APIs de servicios web.[2]

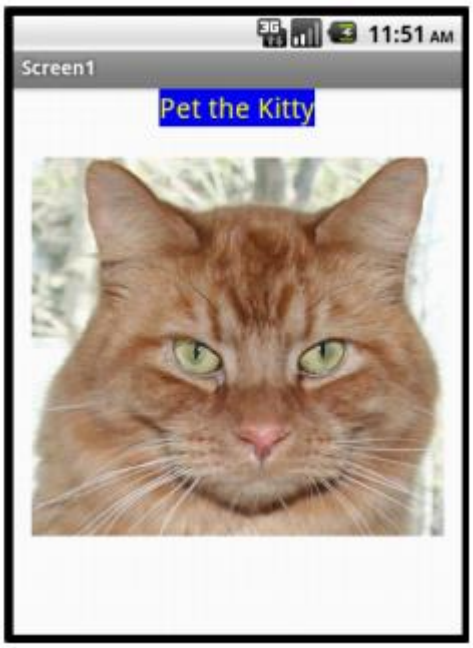
Twitter es un ejemplo de servicio web con el que App Inventor puede interaccionar. Por ejemplo, podríamos escribir una app que solicite y muestre los tweets previos de nuestros amigos, y que también actualice nuestro estado.

[2] Una API (Application Programming Interface) es un conjunto de funciones y procedimientos que cumplen una cierta funcionalidad, y cuya finalidad es ser utilizados por otro software. Esto nos permite utilizar las funcionalidades de la API en nuestro proyecto sin tener que programarlas previamente.

2. ETIQUETAS, BOTONES, Y SONIDOS.

El objetivo de este capítulo es familiarizarnos con algunos de los componentes más básicos de App Inventor. Para ello vamos a programar una aplicación muy sencilla, en la que la imagen de un gato maúlla y ronronea cuando la tocamos, y en la que el gato también maúlla al agitar el dispositivo Android.

2.1. COMENZAR CON LA APP "HELLOPURR".



Para comenzar a programar con App Inventor abrimos un navegador y nos conectamos a http://ai2.appinventor.mit.edu.

Lo primero que vemos al conectarnos a App Inventor es la página de Proyectos. Para crear un nuevo proyecto, pinchamos en el botón "Start new project", escribimos el nombre del proyecto ("HelloPurr"), y pinchamos en "OK". Tras unos segundos, se abre la ventana del Diseñador.

> NOTA: App Inventor es una herramienta *en la nube*, los que significa que nuestra app se almacena en un servidor online. Por consiguiente, si cerramos App Inventor nuestra app seguirá allí cuando volvamos a abrirlo; no hace falta que guardemos nada en nuestro ordenador como haríamos, por ejemplo, con un archivo de Microsoft Word.[3]

2.2. DISEÑAR LOS COMPONENTES DE "HELLOPURR".

Para nuestra app "HelloPurr" necesitaremos dos componentes visibles: Un componente "Label" con el texto "Pet the Kitty", y un componente "Button" con la imagen de un gato en él. También usaremos dos componentes no visibles: Un componente "Sound" para reproducir sonidos y hacer vibrar el dispositivo, y un componente "AccelerometerSensor" para detectar cuándo agitamos el dispositivo.

CONSTRUIR UNA ETIQUETA.

El primer componente a añadir es un componente "Label" (etiqueta):

1) Acudimos a la Paleta, abrimos la bandeja "User Interface", pinchamos en el componente "Label", y lo arrastramos al Visor. Veremos que sobre el Visor aparece una forma rectangular que contiene las palabras "Text for Label1".
2) Acudimos a las propiedades de esta etiqueta. Una de ellas se denomina "Text", y dispone de una caja para escribir el texto que queremos ver impreso en la etiqueta. Cambiamos el texto a "Pet the Kitty" y presionamos ENTER. Veremos el cambio del texto en el Visor.
3) Cambiamos el color de fondo de la etiqueta (propiedad "BackgroundColor"), que actualmente está a "None": Para ello, desplegamos la lista de colores de fondo y elegimos el azul. Cambiamos también el color del texto de la etiqueta (propiedad "TextColor") a amarillo, y finalmente, cambiamos el tamaño de la letra (propiedad "FontSize") a 20.

[3] También hay versiones (no oficiales) de App Inventor que funcionan offline, y que podemos instalar en nuestros ordenadores para desarrollar aplicaciones sin necesidad de disponer de una conexión a Internet. Ver anexo A para más información.

AÑADIR UN BOTÓN.

Ahora vamos a añadir la imagen del gato. Tal vez hayamos pensado que para ello lo más apropiado es el componente "Image" de la bandeja "User Interface". Sin embargo, esta elección tiene un serio inconveniente: Queremos que el gato maúlle al tocar sobre la imagen del gato, pero los componentes "Image" no pueden saber si se les ha tocado. De hecho, se trata de componentes que no son capaces de detectar ninguna acción del usuario; son simples imágenes que solo se quedan en su posición, sin hacer nada.

Por consiguiente, no es "Image" el componente que necesitamos para incorporar la imagen del gato. Hay múltiples opciones de objetos capaces de detectar acciones del usuario, como ser tocados o arrastrados. En nuestro caso, la opción más sencilla es implementar la imagen como un componente de tipo "Button".

Para ello creamos un botón normal, y después cambiamos la imagen del botón a la del gato. Para hacer el botón básico, vamos a la Paleta (bandeja "User Interface") y clicamos en "Button". Arrastramos el objeto al Visor, y lo ubicamos bajo la etiqueta. Veremos que en el Visor aparece un botón rectangular.

Ahora que ya disponemos de un botón lo usaremos para activar el sonido cuando el usuario lo toque. Pero antes queremos que el botón adopte la apariencia de la imagen de un gato, en lugar de parecer un típico botón rectangular. Para ello hacemos lo siguiente:

1) Primero necesitamos la imagen del gato. Esta imagen la tenemos disponible en los archivos de los alumnos con el nombre *kitty.png*.
2) Pinchamos sobre el botón (cuyo nombre es "Button1") para mostrar las propiedades de este componente. En la propiedad "Image", el valor actual es "None" (esto es, el botón no tiene asociada ninguna imagen).
3) Pinchando en la caja de edición de la propiedad "Image", seleccionamos la opción "Upload File…", y subimos la imagen *kitty.png* buscándola en nuestro ordenador y pinchando en "OK". Al cargarla, el botón adquiere la forma de la imagen de un gato. También veremos que el archivo *kitty.png* aparece listado (junto al valor "None") como una opción para la propiedad "Image" del botón. Además, veremos que el archivo *kitty.png* se añade a la lista de archivos multimedia en el área "Media" de la ventana del Diseñador.
4) Notar que el botón (ahora con la imagen del gato) aún conserva las palabras "Text for Button 1". Como no queremos que este texto aparezca en nuestra app, borramos esta frase de la propiedad "Text" del componente "Button1".

AÑADIR UN SONIDO.

En nuestra app queremos que el gato maúlle al tocar en el botón del gato. Para conseguirlo, necesitamos añadir un sonido de maullido y programar el comportamiento del botón para reproducir ese sonido cuando el usuario lo presione. Para ello:

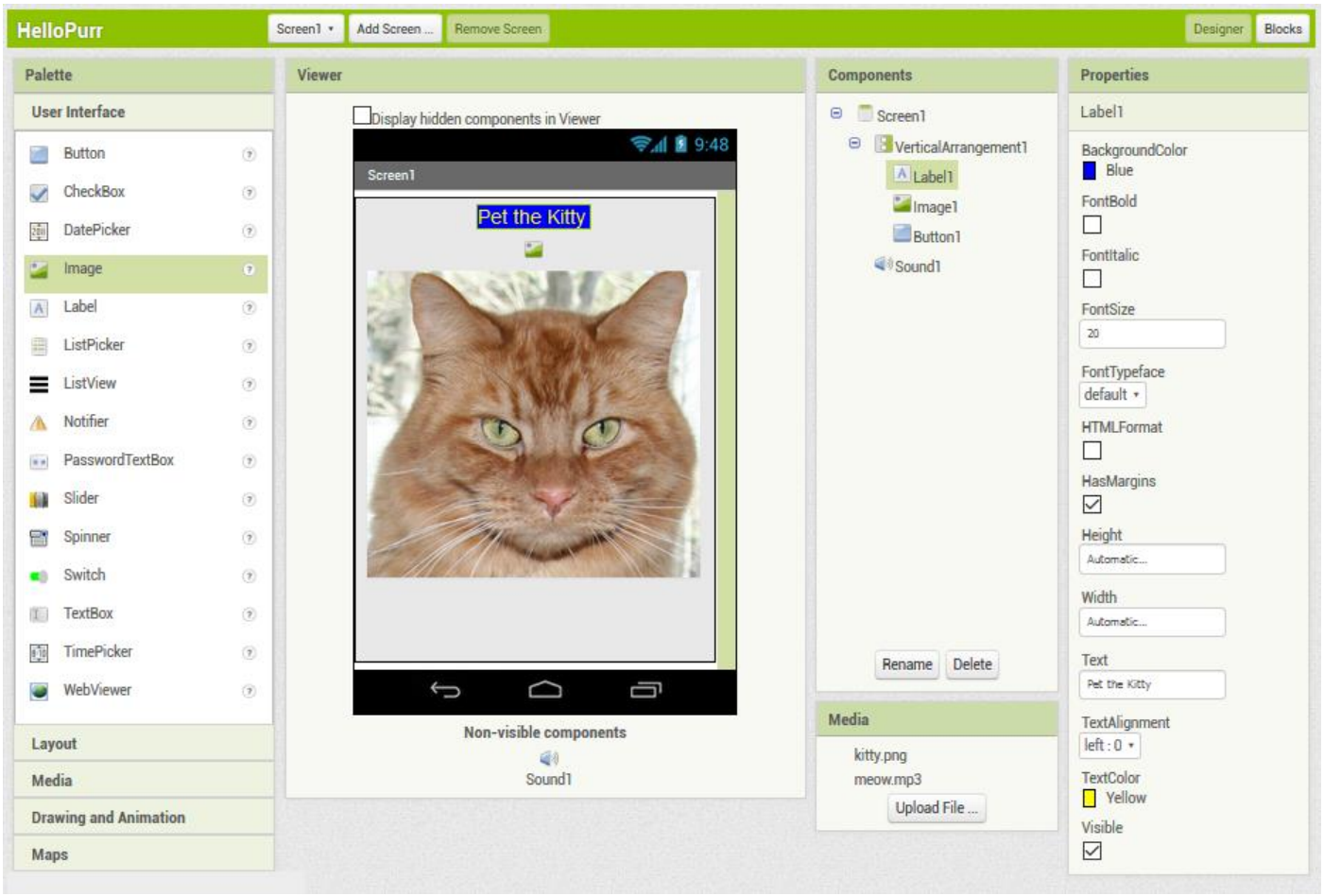
1) En la Paleta clicamos en la bandeja llamada "Media" para expandir la lista de componentes relacionados con el multimedia. Arrastramos un componente "Sound" hasta el Visor. Da igual donde lo soltemos en el Visor, que este componente siempre aparecerá en la zona inferior del Visor llamada "Non-visible components".
2) En el Visor seleccionamos el objeto "Sound1" para mostrar sus propiedades. A continuación, pinchamos en la propiedad "Source" (fuente) y seguimos los pasos necesarios para cargar desde nuestro ordenador el archivo *meow.mp3* (también disponible en los archivos de los alumnos). Al hacerlo, deberíamos ver que tanto *kitty.png* como *meow.mp3* están listados como archivos multimedia en la zona "Media" del Diseñador.

ORGANIZAR LOS COMPONENTES EN LA PANTALLA.

Para tener mejor organizado el diseño de la pantalla, es muy útil poder distribuir los componentes en una cuadrícula. Para ello, podemos centrar la etiqueta en la parte superior de la pantalla, y centrar el botón un poco más abajo de ella. Esto podemos hacerlo con un componente organizador "VerticalArrangement":[4]

1) Añadimos al Visor un componente "VerticalArrangement" desde la bandeja "Layout".
2) Arrastramos los componentes "Label1" y "Button1" hasta dejarlos en la parte superior e inferior del organizador, respectivamente.
3) Para forzar una separación entre la etiqueta y el botón, añadimos un tercer componente dentro organizador: Tomamos un componente "Image" de la bandeja "User Interface" y lo ubicamos dentro del "VerticalArrangement1", entre la etiqueta y el botón. Esta imagen la dejamos vacía, esto es, sin asociarle ningún archivo de imagen (propiedad "Picture" fijada a "None"). Hacemos esto porque la finalidad de este componente no es mostrar una imagen estática por pantalla, sino insertar una separación entre la etiqueta y el botón.
4) Ahora establecemos las propiedades de los componentes para organizar la pantalla:
   - Seleccionamos el componente "VerticalArrangement1" y fijamos sus propiedades "Height" (altura) y "Width" (anchura) a "Fill parent", para que se extiendan a lo largo de las dimensiones de su componente padre, que en este caso es "Screen1". Además, fijamos su propiedad "AlignHorizontal" a "Center", para que los componentes que incluye estén centrados dentro del organizador.
   - Seleccionamos el componente "Image1" y fijamos su propiedad "Height" a 20 píxeles, para establecer una separación entre la etiqueta y el botón igual a esta cantidad.

Con todo lo hecho hasta ahora, el Diseñador debería mostrar un aspecto similar al ilustrado en la figura:



[4] Hablaremos más sobre los componentes organizadores "HorizontalArrangement", "VerticalArrangement", y "TableArrangement" en el próximo capítulo.

Al lanzar las pruebas en vivo de la app en el dispositivo o en el emulador, probamos a tocar el botón del gato. Por supuesto no pasará nada, porque no le hemos dicho al botón qué es lo que debe hacer cuando el usuario lo toque. Éste es un tema importante a tener claro en App Inventor: Para casi cada componente que añadamos en el Diseñador, debemos acudir al Editor de Bloques y crear el código que le diga a ese componente cómo debe comportarse en la app.

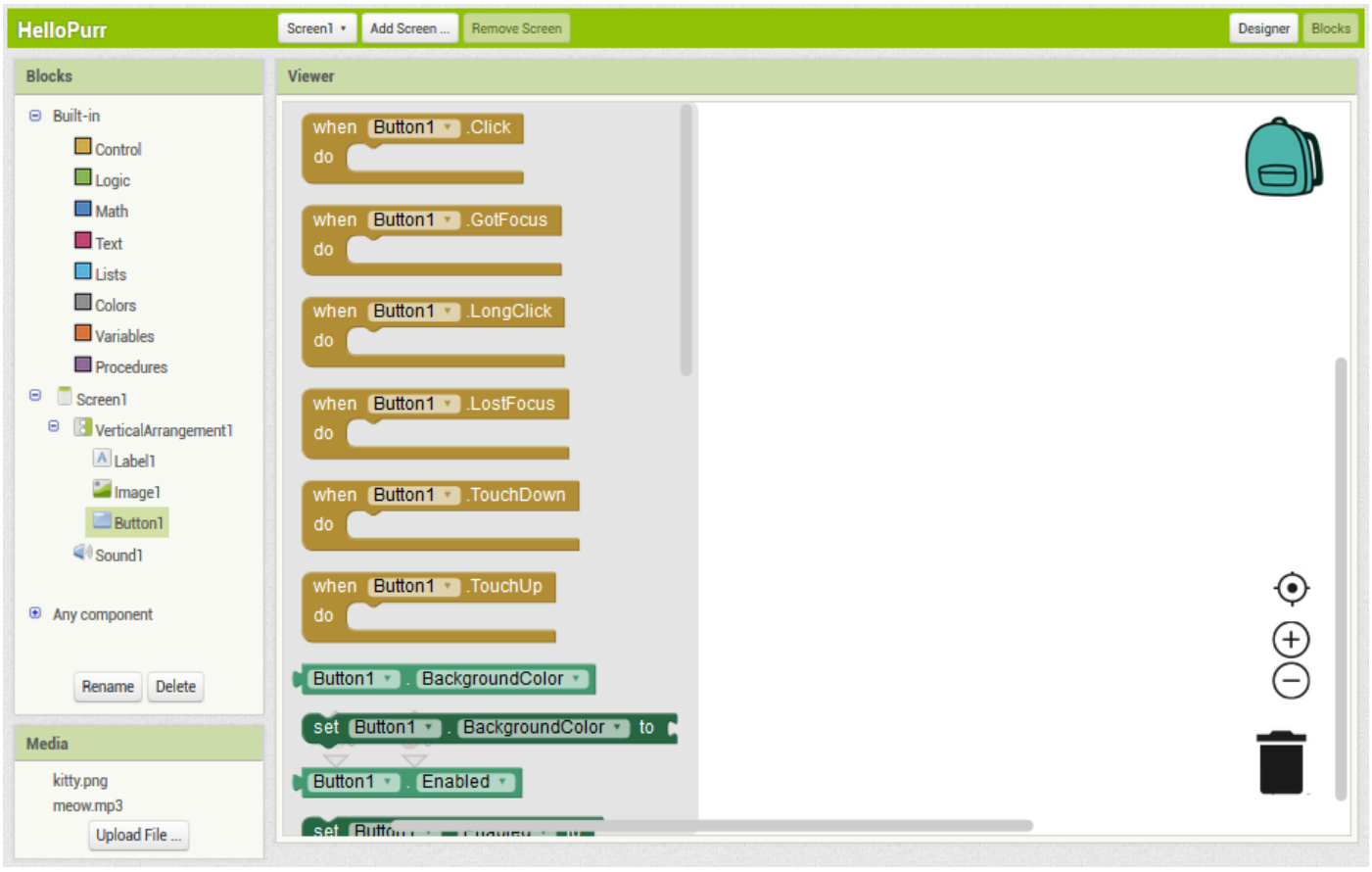
2.3. AÑADIR COMPORTAMIENTOS A LOS COMPONENTES DE "HELLOPURR".

Hemos añadido los componentes "Button", "Label", "VerticalArrangement", "Image", y "Sound" como elementos constitutivos de nuestra primera app. Ahora debemos hacer que el gato maúlle cuando el usuario presione el botón del gato. Este comportamiento lo programamos con el **Editor de Bloques**.

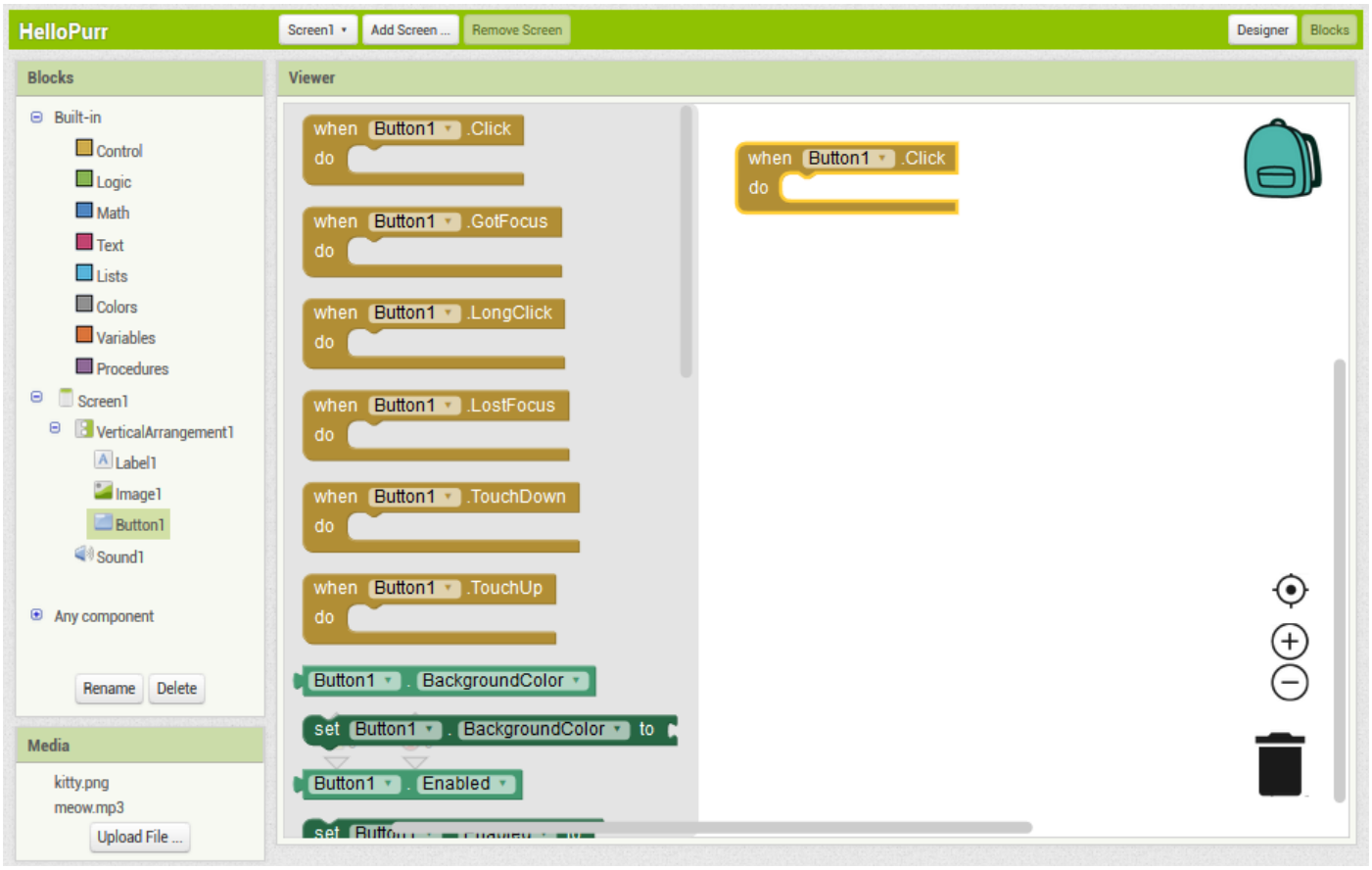
Para acceder al Editor de Bloques, pinchamos en el botón "Blocks" situado arriba y a la derecha de la ventana del Diseñador. En la ventana del Editor de Bloques es donde les decimos a los componentes qué queremos que hagan. Vamos a decirle al botón del gato que reproduzca un sonido cuando el usuario lo toque.

HACER MAULLAR AL GATO.

En el extremo izquierdo de la ventana del Editor de Bloques, bajo la cabecera "Blocks", veremos una columna que contiene varias bandejas: Una bandeja llamada "Built-in" (con los bloques integrados de App Inventor), y una bandeja para cada componente que hayamos añadido en el Diseñador para nuestra app, a saber, "Screen1", "VerticalArrangement1", "Label1", "Image1", "Button1", y "Sound1". Al clicar en cualquiera de estas bandejas aparecerán los bloques relacionados con ese componente. Así, si pinchamos en la bandeja del componente "Button1", se abrirá una selección de bloques que podemos usar para definir el comportamiento del botón (ver figura).



Seleccionamos el bloque "when (Button1).Click" y los arrastramos al área de trabajo en blanco. Recordemos que los bloques que empiezan con la palabra "when" se denominan **manejadores de eventos**, y sirven para especificar qué debería hacer la app cuando ocurre algún evento particular en ese componente. En este caso, el evento en el que estamos interesados es cuando el usuario pincha (click) en el botón con la imagen del gato, como muestra la figura. Para especificar qué hace la app en respuesta a un toque del botón, añadiremos algunos bloques más dentro de la sección "do" del manejador "when (Button1).Click".



Seleccionamos la bandeja "Sound1" para abrir los bloques disponibles para el componente de sonido, y sacamos el bloque "call (Sound1).Play". (Recordemos que antes ya editamos la propiedad "Source" de "Sound1" para asociarle el archivo *meow.mp3*). Notar que el bloque "call (Sound1).Play" tiene arriba y abajo unas muescas y salientes que encajan dentro de la sección "do" del bloque "when (Button1).Click". App Inventor está diseñado para que solo ciertos bloques encajen juntos; de esta forma sabremos que estamos conectando bloques que pueden funcionar conjuntamente. Los bloques con la palabra "call" sirven para decirles a los componentes que hagan ciertas cosas. (En lenguaje técnico, los bloques "call" son **llamadas** a las **funciones** propias de un cierto componente). Los dos bloques pueden conectarse para formar la unidad mostrada en la figura. (Además oiremos un chasquido de conexión cuando los combinemos juntos).

Al contrario que en la programación tradicional (donde un programa es un código formado por instrucciones escritas), los bloques de App Inventor expresan el comportamiento que estamos programando de una forma visual que es mucho más fácil de entender. En este caso, básicamente estamos diciendo, "Oye App Inventor, cuando alguien toque el botón del gato, reproduce el sonido *meow.mp3*".

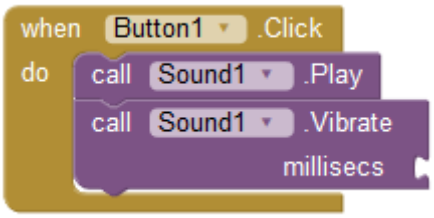
Ahora comprobamos que todo funciona correctamente. Es importante probar nuestra aplicación cada vez que añadimos algo nuevo. Al tocar el botón del gato en el dispositivo (o al pinchar sobre él si estamos usando el emulador), deberíamos oír maullar al gato.

AÑADIR UN RONRONEO.

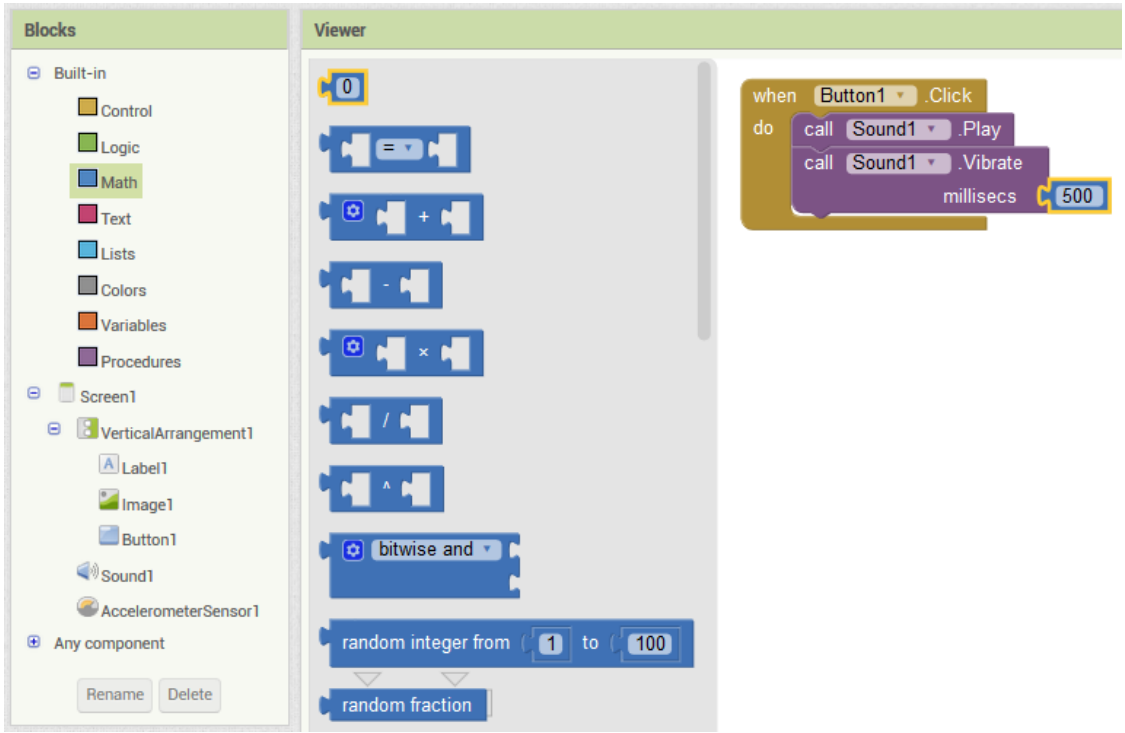
Ahora vamos a hacer que el gato también ronronee al tocar en el botón. El ronroneo lo simularemos haciendo que el dispositivo vibre. Esto puede parecernos muy difícil, pero de hecho es fácil, porque el componente "Sound" que ya hemos usado para reproducir el maullido también puede hacer vibrar al dispositivo. Una de las cosas buenas de App Inventor es que nos permite acceder a este tipo de funciones internas sin necesidad de saber cómo vibra realmente un teléfono móvil.

Para conseguir el ronroneo, basta con añadir un segundo bloque de llamada a una función del componente "Sound1" dentro de la sección "do" del bloque "when (Button1).Click". Para ello:

1) Vamos al Editor de Bloques y pinchamos en la bandeja del componente "Sound1".
2) Seleccionamos el bloque "call (Sound1).Vibrate" y lo colocamos bajo el bloque "call (Sound1).Play" dentro del manejador "when (Button1).Click", como muestra la figura:



3) Notar que el bloque "call (Sound1).Vibrate" incluye el texto "millisecs" (milisegundos) en su esquina inferior derecha, y junto a él hay un conector abierto hacia adentro desde el borde del bloque. Un conector abierto a la derecha de un bloque se denomina **parámetro**, y nos indica que debemos conectarle algo al bloque para especificar alguna cosa más sobre su comportamiento. En este caso, debemos decirle al bloque "Vibrate" cuánto tiempo debe vibrar. Este tiempo lo especificamos en milisegundos, algo bastante habitual en muchos lenguajes de programación. Así pues, para hacer que el dispositivo vibre durante medio segundo, debemos insertar un valor de 500 milisegundos. Y para hacer esto, hemos de usar un bloque numérico. En la bandeja "Built-in" de bloques integrados, seleccionamos la bandeja "Math", y observaremos que aparece una lista de bloques azules, como muestra la figura. Al principio de la lista veremos un pequeño bloque con un 0 dentro. Arrastramos este bloque al área de trabajo, y cambiamos el 0 por un 500. Por último, conectamos el bloque numérico con el 500 al conector abierto a la derecha del bloque "call (Sound1).Vibrate", como muestra la figura:

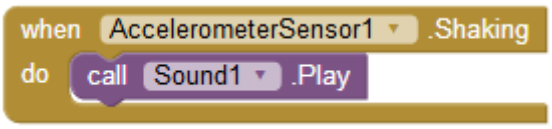


Probamos estas modificaciones en nuestro dispositivo Android. Al presionar el botón del gato, además de escuchar un maullido, deberíamos sentir un ronroneo (en la forma de una vibración del móvil) durante medio segundo.

SACUDIR EL DISPOSITIVO.

Ahora vamos a añadir una característica final a nuestra app: Haremos que el gato maúlle al agitar el dispositivo. Para conseguirlo, usaremos un componente llamado "AccelerometerSensor", el cual es capaz de detectar cuándo sacudimos o movemos el dispositivo:

1) En el Diseñador, en la Paleta de componentes, expandimos la bandeja "Sensors", y arrastramos el componente "AccelerometerSensor" al Visor. Como todo componente no visible no importa en qué lugar del Visor lo soltemos, que se irá directamente a la sección "Non-visible components" en la parte inferior del Visor.
2) En esta app queremos tratar el hecho de que el usuario agite el dispositivo como un evento distinto e independiente al toque del botón. Esto significa que necesitaremos un nuevo manejador de eventos. Acudimos al Editor de Bloques, donde debería aparecer la bandeja para el componente "AccelerometerSensor1". Pinchamos en él para acceder a sus bloques, y arrastramos al área de trabajo el manejador de evento "when (AccelerometerSensor1).Shaking".
3) Como hicimos con el click del botón, tomamos un bloque "call (Sound1).Play" y lo conectamos dentro de la sección "do" del bloque "when (AccelerometerSensor1).Shaking". Agitamos el dispositivo para ver qué ocurre.



2.4. DESCARGAR UNA APP A UN DISPOSITIVO ANDROID.

El método de pruebas en vivo nos permite comprobar el funcionamiento de la app mientras App Inventor está conectado al dispositivo. El único problema es que si desconectamos el dispositivo de App Inventor la app se parará y no la encontraremos por ningún lado en nuestro teléfono. Esto es así porque no habíamos instalado la app; solo se estaba ejecutando dentro de la aplicación "MIT AI2 Companion".

Pero también podemos descargar e instalar una app para que funcione en cualquier dispositivo, incluso aunque no esté conectado al ordenador. Para ello, lo primero que debemos hacer es establecer el icono de la app para que, cuando la instalemos en un dispositivo, el icono represente a la app en la lista de aplicaciones. Esto lo hacemos en el Diseñador seleccionando el componente "Screen1", pinchando en la propiedad "Icon", y cargando un archivo de imagen para el icono (para nuestra app, seleccionaremos la imagen del gato, *kitty.png*).

A continuación nos aseguramos de que nuestro dispositivo permita que las aplicaciones puedan descargarse desde otros lugares, aparte de hacerlo desde la tienda Android. En la mayoría de dispositivos Android, esto puede hacerse entrando en "Ajustes" → "Seguridad", y activando la opción "Instalar apps desconocidas" en las aplicaciones para las que queramos habilitar esta opción (como por ejemplo, el navegador de Internet, la app AI2 Companion, etc.).

Ahora, y de vuelta al Diseñador de App Inventor, pinchamos en el menú "Build", y seleccionamos "App (provide QR code for .apk)". Debería aparecer una ventana con una barra de progreso que puede tardar unos minutos en completarse. Cuando la app esté lista, se mostrará por pantalla un código QR. Escaneamos

este código con una app de escaneo de códigos de barras.[5] Después de escanear el código QR, puede que el dispositivo nos pida que insertemos la contraseña de nuestra cuenta Google. Al introducir la contraseña, la app empezará a descargarse en el dispositivo, y veremos un icono de descarga en las notificaciones del dispositivo. Vamos a las notificaciones, esperamos a que termine la descarga, y elegimos la app para instalarla.

Después de haberla instalado, miramos las apps disponibles en nuestro dispositivo, y veremos la app "HelloPurr" que acabamos de construir. Esta app se ejecuta como cualquier otra aplicación. (Debemos asegurarnos de ejecutar nuestra nueva app, y no la aplicación "MIT AI2 Companion"). Ahora podemos parar la aplicación "MIT AI2 Companion" o desconectar el dispositivo del ordenador, que nuestra nueva app empaquetada seguirá estando ahí.

Es importante entender que a partir de ahora nuestra **aplicación empaquetada** "HelloPurr" es una app independiente del proyecto "HelloPurr" en App Inventor. Esto significa que podemos actualizar o modificar el proyecto "HelloPurr" en App Inventor, y probar en vivo estos cambios en el dispositivo con la aplicación "MIT AI2 Companion", exactamente igual que antes. Pero estos cambios no se reflejarán en la aplicación empaquetada que ya está instalada en el dispositivo. Si hacemos cambios sobre nuestro proyecto en App Inventor, probablemente querremos volver a empaquetar la app resultante, y descargar la nueva versión para reemplazar la versión antigua que ya teníamos instalada en el teléfono.

2.5. COMPARTIR UNA APP.

Podemos compartir nuestra app de varias formas. Para compartir la app ejecutable (el archivo .apk), primero pinchamos en "Build" y elegimos "App (save .apk to my computer)". Con esto crearemos un archivo con una extensión .apk en nuestro ordenador. Podemos compartir este archivo con otras personas enviándoselo como un adjunto en un email, para que lo abran mediante sus gestores de correo en sus dispositivos. También podemos subir el archivo .apk a la nube (por ejemplo, a Dropbox, OneDrive, etc.). Pero debemos asegurarnos de que las personas que necesiten nuestra app sepan que deben permitir la instalación de apps desconocidas en sus dispositivos.

También podemos crear un código QR para la app, de forma que la gente pueda escanearlo desde una web, o incluso desde un póster físico impreso en papel. Hay muchas herramientas que generan códigos QR a partir de URLs (por ejemplo, http://qrcode.kaywa.com). Podemos copiar y pegar el código QR en una página web o en un documento para imprimirlo como un póster.

Otra posibilidad es compartir el código fuente (los bloques) de nuestra app con otro programador de App Inventor. Para ello, pinchamos en "Projects" → "Export selected Project (.aia) to my computer". El archivo creado (con extensión .aia) puede enviarse por email, y el receptor puede guardalo en su ordenador y abrirlo en App Inventor seleccionando "Projects" → "Import project (.aia) from my computer". Esto le permitirá a esa persona disponer de una copia completa de nuestra app, para editarla y personalizarla sin afectar a nuestra propia versión.

App Inventor también dispone de una galería de apps donde cualquiera puede subir y compartir sus aplicaciones, y para acceder a las aplicaciones de otros desarrolladores de todo el mundo.

[5] Hay muchos escáneres de códigos QR para Android. Si no tenemos instalado ningún escáner, acudimos a Play Store e instalamos uno en nuestro dispositivo.

2.6. MODIFICACIONES A "HELLOPURR".

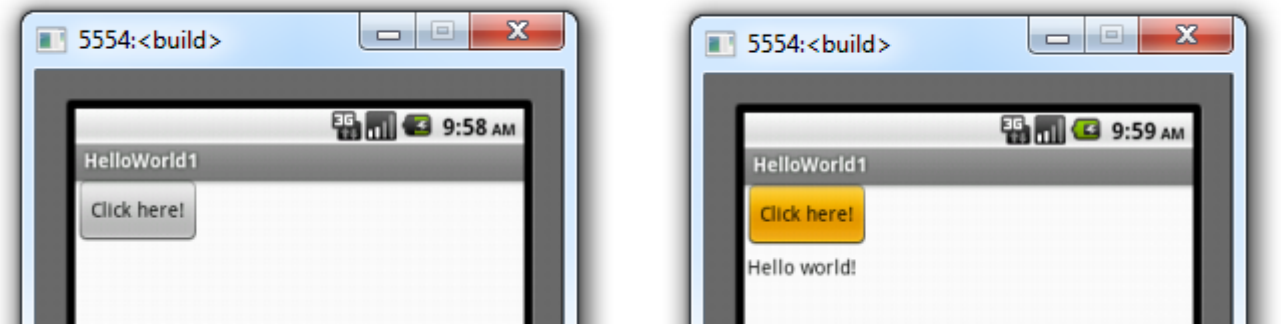
He aquí algunas modificaciones que podemos probar con la app "HelloPurr":

- Al agitar el dispositivo los maullidos suenan extraños, como con eco. Esto ocurre porque el sensor acelerómetro activa el evento "Shaking" (agitar) muchas veces por segundo en respuesta a cada movimiento ascendente o descendente, y los distintos maullidos que se reproducen se solapan. Si miramos el componente "Sound" en el Diseñador, veremos una propiedad llamada "MinimumInterval". Esta propiedad determina cómo de juntos podrán reproducirse varios sonidos sucesivos. Actualmente está fijado a un valor un poco más bajo de medio segundo (400 milisegundos), lo que es menos que la duración de un solo maullido. Ajustando el intervalo mínimo, podemos cambiar cuánto se solapan los diferentes maullidos.

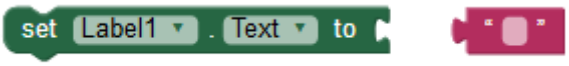
- Si ejecutamos la aplicación empaquetada y caminamos con el móvil en nuestro bolsillo, el dispositivo maullará cada vez que nos movamos bruscamente, lo que resultará bastante embarazoso. Las aplicaciones de Android están diseñadas para seguir ejecutándose incluso cuando no las estamos mirando; por lo tanto, nuestra app seguirá comunicándose con el acelerómetro y los maullidos seguirán sonando. Para parar la app del todo, debemos acceder a la aplicación "HelloPurr" y presionar el botón de menú del dispositivo. Este menú te ofrecerá la opción de detener la aplicación.

2.7. EJERCICIOS DEL CAPÍTULO 2.

Ejercicio 2.1. Hola mundo con botón y etiqueta.
Programa una aplicación llamada "HelloWorld3" basada en las apps que desarrollamos en el capítulo 1. En este caso, cuando el usuario pulsa un botón, una etiqueta muestra por pantalla el texto "Hello World!". Hasta que el usuario no presione el botón, la etiqueta no mostrará texto alguno.



PISTA: Para cambiar la propiedad "Text" de una etiqueta desde el Editor de Bloques, podemos usar el bloque "set (Label).Text to". El texto que debe mostrar la etiqueta puede especificarse mediante un bloque de texto (bandeja "Text").



Ejercicio 2.2. Látigo.
Crea una aplicación que reproduzca el sonido de un látigo cada vez que agites tu teléfono móvil. Usa el archivo *whip.mp3*.

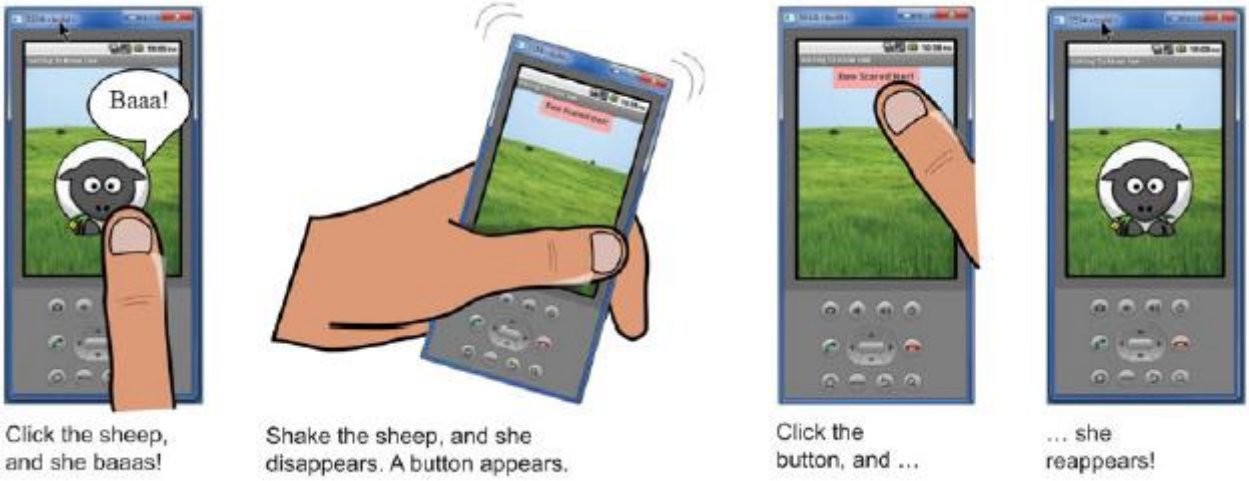
Ejercicio 2.3. Instrumentos musicales.
Programa una aplicación Android en la que la interfaz de usuario muestre 3 botones de instrumentos musicales (digamos, un tambor, un piano, y una trompeta), y una sola etiqueta bajo los tres botones. Inicialmente, el texto de la etiqueta está en blanco. Cuando el usuario pulsa cualquiera de estos tres botones, la etiqueta muestra un texto que indica sobre qué instrumento se ha pulsado, y además, suena un audio con un sonido del instrumento pulsado. (NOTA: Para este ejercicio necesitarás los archivos de sonido

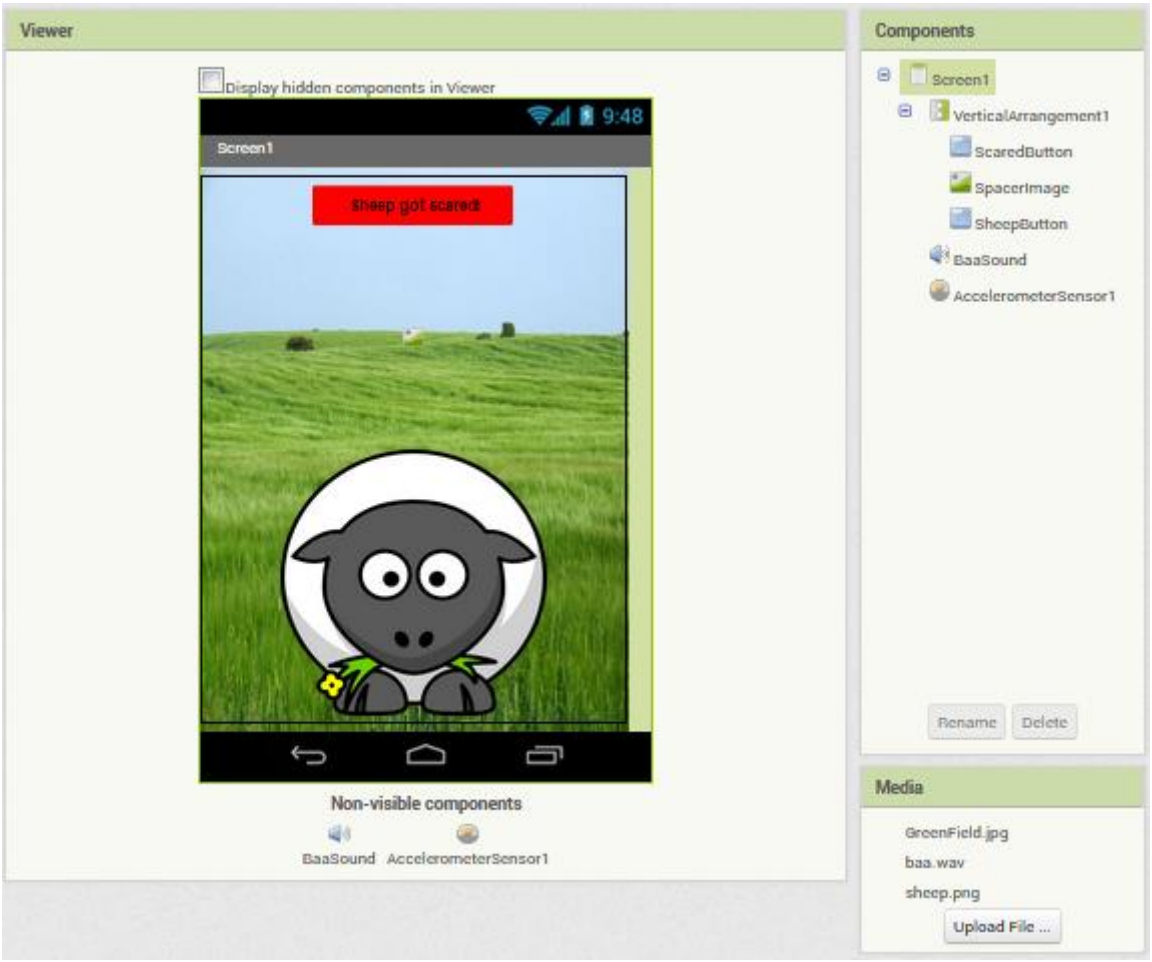
drum.mp3, *piano.mp3*, y *trumpet.mp3*. Busca en Internet las imágenes que necesites para los botones. Para buscar imágenes PNG gratuítas, recomendamos la web http://www.clker.com/).

Ejercicio 2.4. Oveja.

Crea una app en la que aparezca una oveja sobre un campo verde. La app debe funcionar así:

- Al pulsar sobre la oveja, la oveja bala y el móvil vibra durante 1 segundo.
- Al agitar el móvil, la oveja se asusta y desaparece. Al mismo tiempo, aparece un botón con el texto "Sheep got scared!".
- Al pulsar sobre ese botón, la oveja vuelve a aparecer y el botón desaparece.





PISTAS:

1) Carga a tu proyecto los archivos *baa.wav*, *GreenField.jpg*, y *sheep.png*, ver figura.
2) Fija el fondo de la pantalla al archivo *GreenField.jpg*, ver figura.

3) Utiliza un componente "VerticalArrangement" para distribuir adecuadamente sobre la pantalla a la oveja y al botón que la hace reaparecer. Fija su propiedad "BackgroundColor" adecuadamente para hacerlo transparente. Ubica el botón en la parte superior del "VerticalArrangement" y a la oveja en la parte inferior. Inserta entre ambos un componente "Image" vacío que actúe como separador. Fija adecuadamente la propiedad "Height" de "Image" para que el botón quede en la parte alta de la pantalla, y la oveja en la parte baja (sobre la hierba), ver figura.
4) Nada más empezar la app, la oveja debería estar visible y el botón debería ser invisible. Para ello:
   - El manejador de evento que nos permite detectar que la app acaba de empezar es "when (Screen1).Initialize". (Ver capítulo 1)
   - Para establecer la visibilidad de un componente, debemos fijar su propiedad "Visible" a "true" o "false", mediante el bloque "set (Component).Visible to" y los bloques "true" o "false" de la bandeja "Logic".
5) Todo lo demás deberías ser capaz de hacerlo tú mismo.

3. ORGANIZADORES Y LIENZOS.

Hacer que los componentes aparezcan en el lugar adecuado en la pantalla puede llegar a ser muy complicado, sobre todo porque los dispositivos Android están fabricados en formas y tamaños muy diversos. Además, nunca sabremos si el usuario va a inclinar el dispositivo, haciendo que la pantalla cambie de vertical a apaisada, y viceversa. En el capítulo previo ya usamos un componente "VerticalArrangement" para conseguir que los componentes apareciesen en la posición deseada en la pantalla, y usamos un separador invisible (componente "Image" vacío) para insertar un espacio entre una etiqueta y un botón. En la primera parte de este capítulo ampliaremos estas ideas utilizando más propiedades de la pantalla (componente "Screen1") y recurriendo a otros componentes organizadores de la bandeja "Layout".

En la segunda parte del capítulo presentaremos el componente "Canvas" (lienzo), el cual constituye una poderosa herramienta para agregrar elementos gráficos a la pantalla. En este capítulo veremos que los lienzos nos permiten interactuar con la pantalla de formas muy interesantes. En concreto, construiremos una app que nos permitirá pintar sobre la pantalla arrastrando nuestro dedo sobre ella.

3.1. ORGANIZADORES.

En esta sección vamos a aprender a organizar los componentes en la pantalla para que se ajusten a una cuadrícula, y después aplicaremos estos conocimientos a la construcción de una app que reproduce sonidos fantasmagóricos al presionar ciertos botones. Esta app parece sencilla, pero además de reproducir los sonidos también debe ajustarse a los siguientes requisitos:

- La app debe responder adecuadamente al giro o inclinación del dispositivo por parte del usuario. Si la pantalla también rota, algunos de los botones de sonido podrían quedar inaccesibles.
- Los botones deben aparecer centrados en una cuadrícula también centrada, y con el tamaño adecuado.

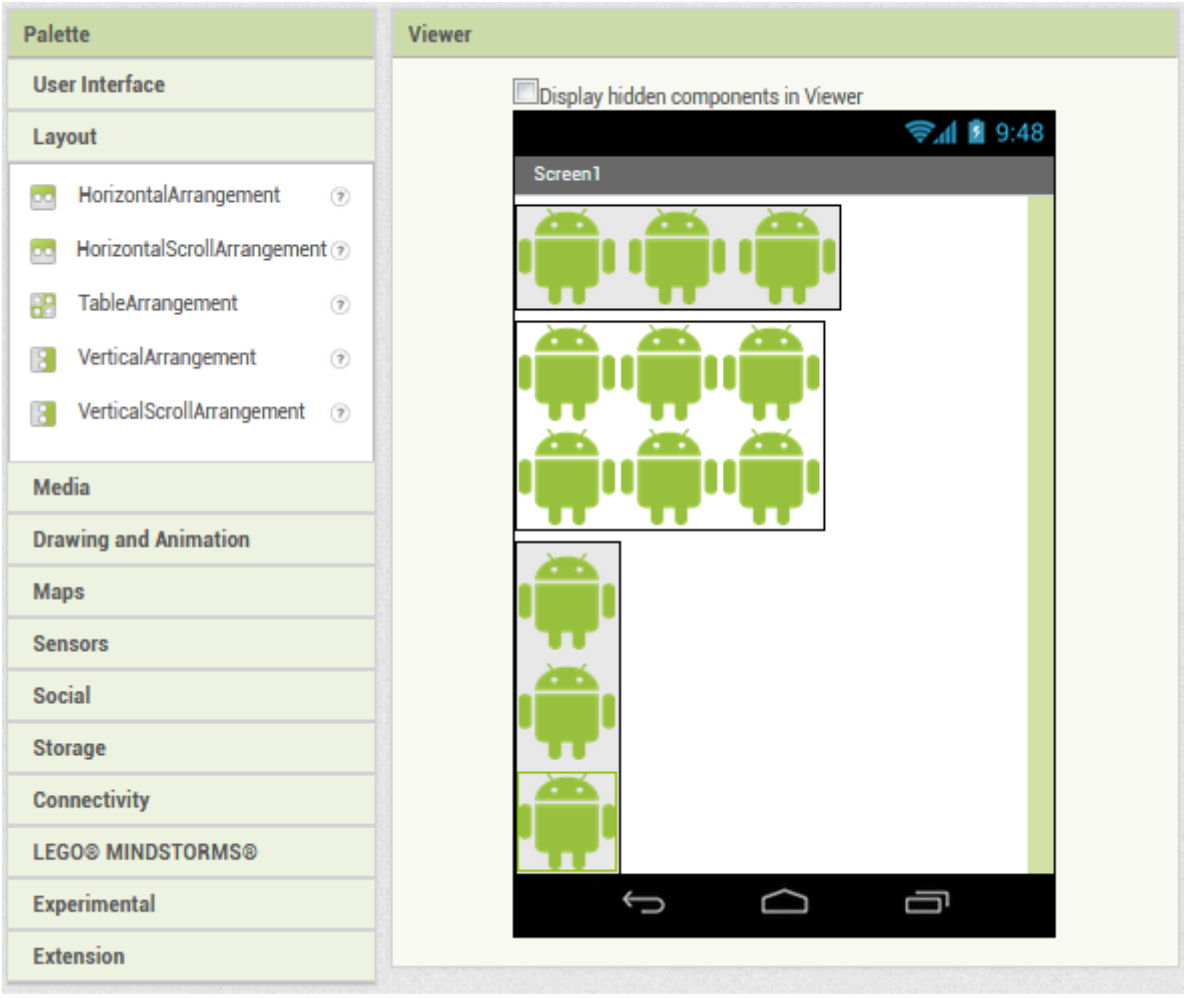
Gestionar la rotación del teléfono es fácil: Hay una propiedad de la pantalla que nos permite activar o desactivar la rotación automática. En el caso de esta app, vamos a impedir que la pantalla rote.

Organizar los elementos en una cuadricula es algo un poco más complejo. A menos que le especifiquemos otra forma de hacerlo, App Inventor apila por defecto todos los componentes agregados a la pantalla uno encima de otro. Podríamos cambiar el orden en el que están apilados, pero si queremos otra disposición distinta debemos usar los componentes **organizadores** ("Arrangements"). En la bandeja "Layout" podemos elegir entre tres organizadores distintos:

- El componente "HorizontalArrangement" se usa cuando queremos ubicar los componentes horizontalmente a lo ancho de la pantalla.
- El componente "TableArrangement" nos permite distribuir los componentes en una cuadrícula o tabla. La cuadrícula estará dividida en celdas, y en cada celda solo podemos ubicar un componente. Si añadimos dos componentes en una única celda, solo se mostrará el último componente añadido.
- El componente "VerticalArrangement" nos permite ubicar los componentes apilados verticalmente uno sobre otro.

La figura muestra los tres organizadores, y cómo actúan sobre varios componentes "Image" contenidos dentro de ellos. En general, usaremos los componentes organizadores para crear diseños verticales, horizontales, o tabulares. También podemos crear diseños más complejos insertando y anidando organizadores dentro de otros organizadores.

Es importante puntualizar que cuando creamos un organizador "TableArrangement", debemos especificar el número de columnas y de filas en sus propiedades "Columns" y "Rows", respectivamente. Por ejemplo, para el organizador "TableArrangement" de la figura (con 6 iconos Android), hemos fijado 2 filas y 3 columnas.



3.2. COMENZAR CON LA APP "SOUNDFX".

Vamos a crear una app llamada "SoundFX" que nos permita reproducir hasta ocho sonidos fantasmagóricos diferentes al presionar el botón correspondiente. La figura muestra el aspecto final de la interfaz de usuario de la app. Los ocho sonidos y los iconos para los ocho botones están disponibles en los archivos de los alumnos.

Vamos a comenzar a crear la app. En el navegador, nos conectamos a la web de App Inventor (http://ai2.appinventor.mit.edu). Comenzamos un nuevo proyecto y lo llamamos "SoundFX". En el menú superior de App Inventor pinchamos en "Connect" y configuramos nuestro dispositivo (o el emulador) para hacer pruebas en vivo. (Si tenemos problemas para establecer este proceso de pruebas en vivo, acudimos a http://appinventor.mit.edu/explore/ai2/setup).

A continuación, a la derecha del Diseñador, vamos al panel de Propiedades. Dentro de las propiedades de la pantalla (componente "Screen1") cambiamos el título de la pantalla (propiedad "Title") a "Sound FX - Spooky". Este cambio se reflejará en nuestro dispositivo, en la barra de título de la pantalla.

3.3. DISEÑAR LOS COMPONENTES DE "SOUNDFX".

AÑADIR LOS COMPONENTES Y ARCHIVOS MULTIMEDIA.

La tabla a continuación lista todos los componentes necesarios para la app, así como los ajustes de sus propiedades:

Spooky Sound FX app			
Screen1 properties	`AlignHorizontal`: Center `ScreenOrientation`: Portrait `Scrollable`: Yes (selected) `Title`: Sound FX: Spooky `Icon`: ghost_button.png (downloaded from our website) `BackgroundColor`: Black (A black background makes gridlines hard to see, so only set this after you've arranged all the other components onscreen.)		
Components	**What do I rename it?**	**What does it do?**	**What properties do I set?**
`TableArrangement` Palette group: Layout	`TableArrangement1`	Creates a 2-column x 4-row grid that organizes your buttons on the screen.	`Columns`: 2 `Rows`: 4 `Width`: Automatic `Height`: Fill Parent

8 `Buttons`
Palette group:
User Interface
	`GhostButton`
	`BatButton`
	`OwlButton`
	`FootstepButton`
	`CatButton`
	`WolfButton`
	`CackleButton`
	`ThunderButton`
		These are the buttons the user clicks to hear a sound.	`Image`: Upload the matching media file for each button: ghost_button.png for `GhostButton`, bat_button.png for `BatButton`, and so on.
			`Text`: Nothing—delete the text.
			`Width`: 90 pixels
			`Height`: 90 pixels

8 `Sound` objects
Palette group: Media
	`GhostSound`
	`BatSound`
	`OwlSound`
	`FootstepSound`
	`CatSound`
	`WolfSound`
	`CackleSound`
	`ThunderSound`
		Creates eight sound objects you can play using the Blocks Editor, just like you played the "baa" sound in Getting to Know Ewe.	`Source`: Upload the matching media file for each sound: ghost.wav for `GhostSound`, bats.wav for `BatSound`, and so on.

La siguiente tabla muestra los archivos multimedia (imágenes y sonidos) que necesitaremos en nuestro proyecto. Como ya indicamos, todos estos recursos están disponibles en la carpeta de archivos de los alumnos. Para agregar estos archivos al proyecto, debemos cargarlos uno a uno en la sección "Media" del Diseñador de Componentes.

Media files	
8 sound files downloaded from our website: ghost.wav bats.wav owl.wav cackling.wav thunderclap.wav cat.wav wolf.wav footstep.wav	Each sound object's `Source` property will be set to one of these matching files so that when you play the sound, you hear the correct file.
8 image files downloaded from our website: bat_button.png lightning_button.png cat_button.png owl_button.png footsteps_button.png witch_button.png ghost_button.png wolf_button.png	Each button's `Image` property will be set to display one of these images. 

AJUSTAR LAS PROPIEDADES DE LA PANTALLA.

En ocasiones querremos que la app cambie la orientación de la pantalla cuando el usuario gire el teléfono, pero en esta vez necesitamos dejar la orientación de la pantalla fija en vertical. También queremos que el usuario pueda desplazarse por la pantalla para ver los botones ocultos en el caso de que la pantalla física de su terminal sea muy pequeña. Finalmente, queremos que la tabla y los botones estén centrados. Todos estos aspectos pueden controlarse mediante las propiedades del componente "Screen1" que indicamos a continuación:

- La propiedad "AlignHorizontal" nos permite fijar un alineamiento horizontal común a todos los componentes ubicados en la pantalla.
- Si ajustamos la propiedad "ScreenOrientation" a "Portrait" fijamos la orientación de la pantalla a vertical, incluso aunque el usuario gire su teléfono.
- Al activar la propiedad "Scrollable" permitimos que el usuario pueda desplazarse por la pantalla para visualizar aquellos elementos que han quedado fuera de la zona visible.

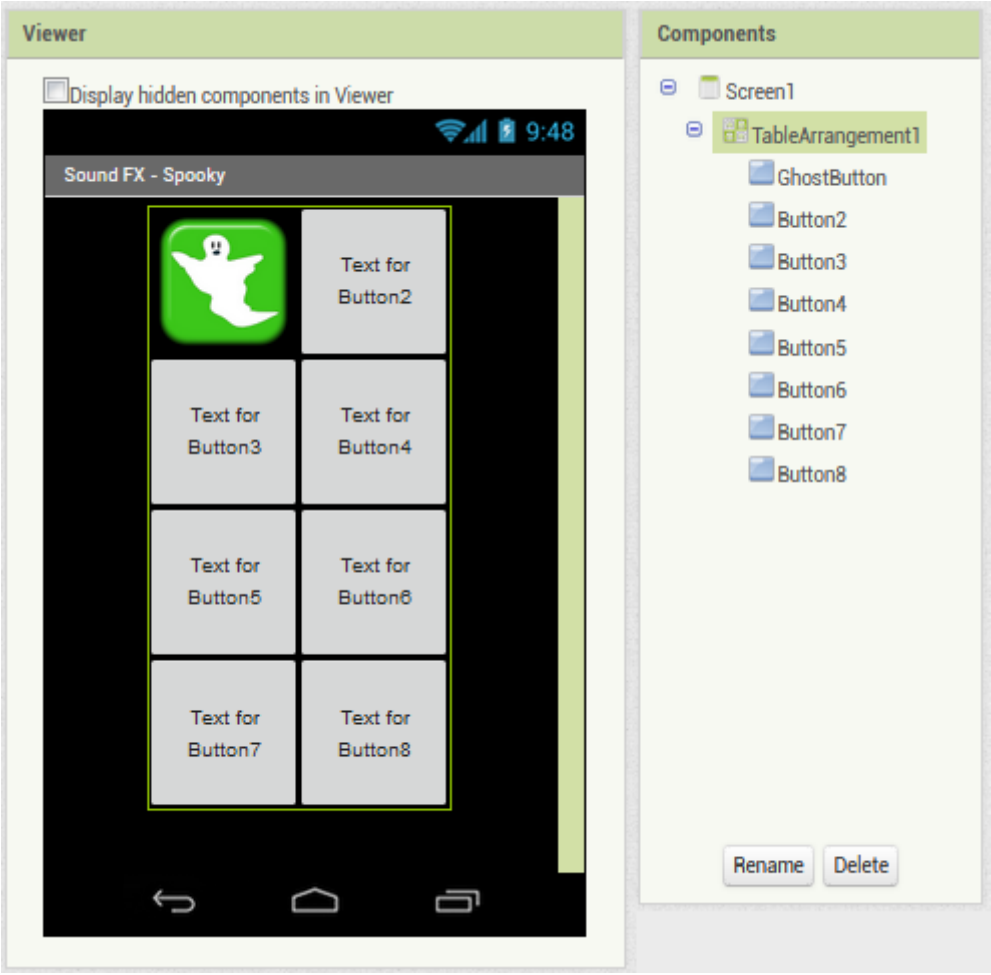
Basándonos en la tabla previa, fijamos los valores adecuados para las propiedades recién comentadas. Además, debemos fijar la propiedad "BackgroundColor" (el color de fondo de la pantalla) a negro.

ORGANIZAR LOS BOTONES.

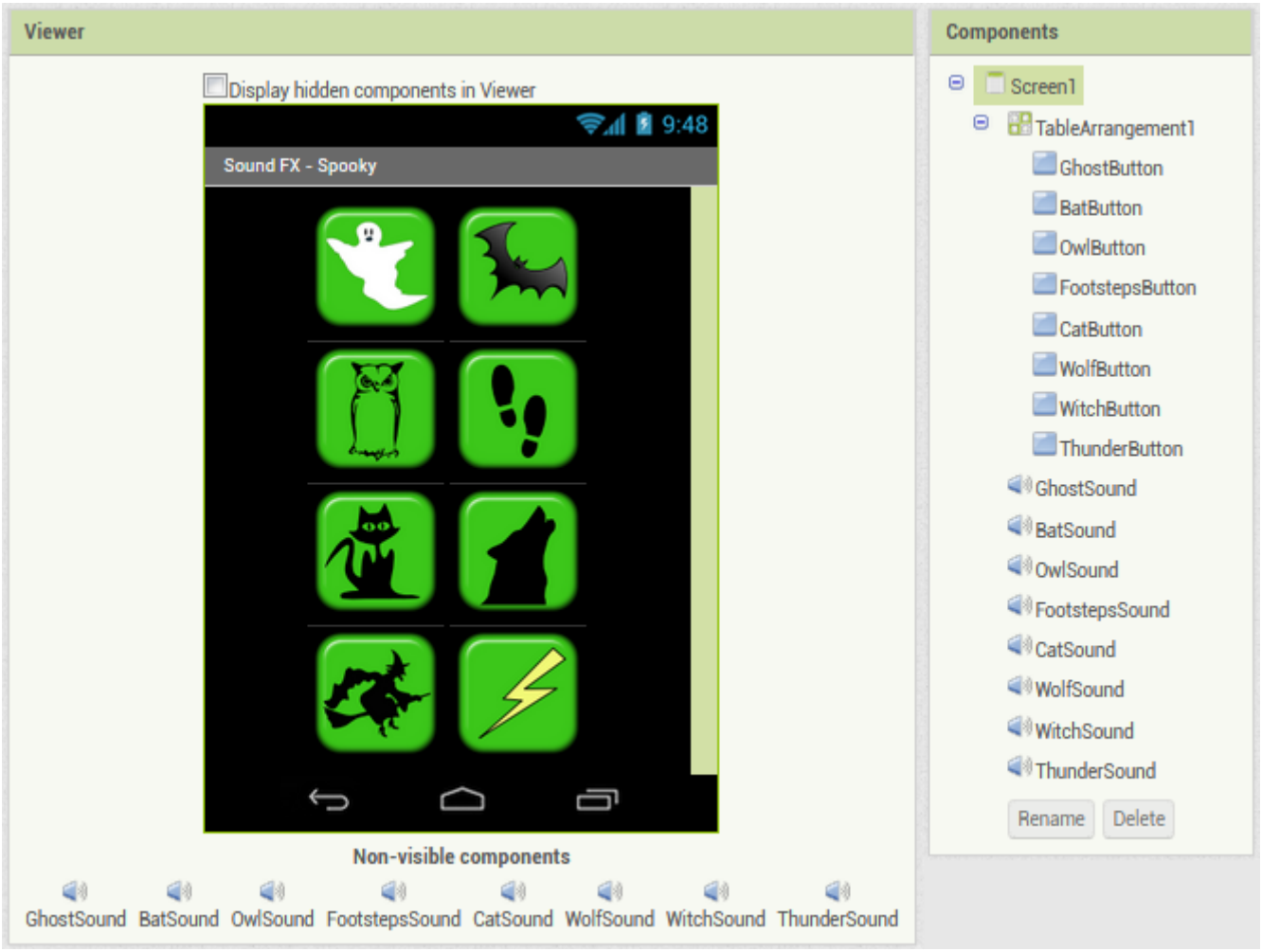
Una vez añadido el componente "TableArrangement", nos aseguramos de fijar sus propiedades tal y como indica la tabla previa. Deberíamos ver una caja vacía con un borde verde, que representa al componente organizador. Ahora añadimos los ocho botones, ubicando cada uno de ellos en la celda adecuada de la tabla (ver figura). Ahora renombramos estos ocho botones para poder identificarlos fácilmente, cambiamos su imagen asociada, borramos su texto por defecto, y ajustamos su anchura y altura a 90 píxeles.

AÑADIR LOS OCHO COMPONENTES DE SONIDO.

Para terminar, añadimos los ocho componentes "Sound" (todos ellos aparecerán en la zona no visible del Diseñador). Los renombramos como indica la tabla, y fijamos sus propiedades "Source" a los archivos de audio adecuados. Por ejemplo, fijamos la propiedad "Source" del componente "WolfSound" al archivo llamado *wolf.wav*.

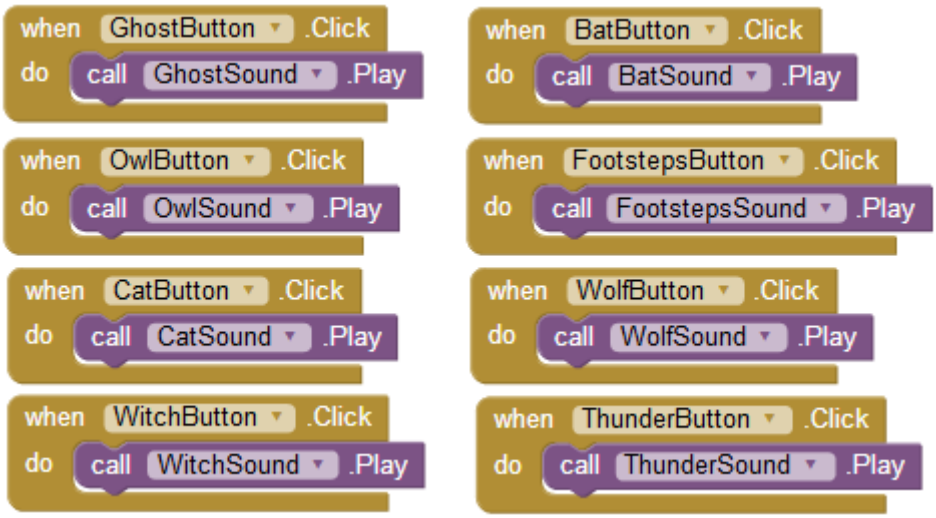


Con todo esto, la pantalla de nuestra app ya está completa, y debería parecerse a la mostrada en la figura:

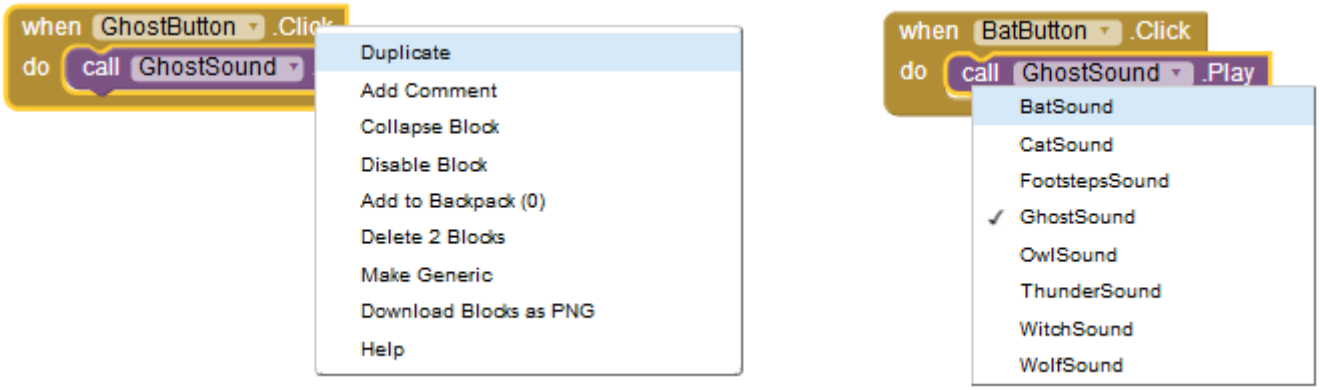


3.4. AÑADIR COMPORTAMIENTOS A LOS COMPONENTES DE "SOUNDFX".

El objetivo principal de esta app es mostrar cómo organizar los componentes en pantalla. En consecuencia, el código que gobierna su funcionamiento es muy sencillo: Es idéntico al código del botón del gato de la app "HelloPurr". La única diferencia es que tendremos que replicar el mismo código para los ocho botones de los sonidos fantasmagóricos. La figura muestra todos los programas necesarios:



Un consejo: Para copiar rápidamente bloques de código que son muy similares, podemos pinchar sobre el bloque de código a copiar, clicar el botón derecho del ratón, y seleccionar la opción "Duplicate" (ver figura). De esta forma, obtendremos una copia exacta del bloque duplicado. A continuación, usamos los menús desplegables de los bloques para seleccionar el componente y el archivo de audio adecuados (ver figura).



Y con esto hemos terminado la aplicación. Vamos a probar cómo funciona en nuestro dispositivo. La apariencia de la app debería ser idéntica a la mostrada en la figura al principio de la sección 3.2. Debemos comprobar que al girar el dispositivo la pantalla de la app no gira automáticamente, sino que sigue apareciendo en vertical. Además, al tocar en cualquiera de los ocho botones de la interfaz de usuario, debería reproducirse el sonido correspondiente.

3.5. MODIFICACIONES A "SOUNDFX".

A continuación proponemos algunos cambios con los que podríamos mejorar esta app:

- Podemos añadir más filas y columnas de botones a la app. Para ello, deberemos buscar algunas imágenes de botones y sus sonidos correspondientes.
- Podemos crear una app alternativa con botones que reproduzcan sonidos de otra temática, como por ejemplo, sonidos de ciencia ficción, de acción, etc.

- Otra app de botones de sonidos podría reproducir sonidos y diálogos de nuestra película favorita.
- También podríamos hacer una app que reproduzca sonidos con mensajes grabados de nuestros familiares o amigos, y donde los botones sean fotografías suyas.
- Una última idea sería una app con nuestros chistes favoritos. También deberíamos añadir un botón de aplauso y un botón de risas enlatadas, para reproducirlos en el caso de que no se ría nadie.

Hay muchos sitios de internet desde donde podemos descargar efectos de sonido e imágenes prediseñadas (clip arts) para nuestros botones. Una web muy recomendable para las imágenes es http://www.clker.com/. Para los archivos de sonido podemos recurrir a https://freesound.org/, https://www.freesfx.co.uk/, o a http://soundbible.com/, Y si lo necesitamos, podemos editar los archivos de sonido mediante un programa edición de audio gratuito, como Audacity (https://sourceforge.net/projects/audacity/).

NOTA: Formato de los archivos de imagen y de audio.
El componente "Sound" de App Inventor suele funcionar bien con archivos de audio en formato WAV o MP3, pero en ocasiones da problemas con los archivos de audio que son muy largos, o que incluyen etiquetas. En tales casos, podemos hacer lo siguiente:

- Comprobamos que el archivo de audio es corto (menos de 30 segundos).
- Lo convertimos a un formato diferente. Por ejemplo, si es un archivo MP3, lo convertimos a WAV. Para ello, podemos usar Audacity o un software online como https://www.media.io/.
- Usamos Audacity para hacer el archivo más corto reduciendo su calidad (bit rate). Un valor de $11{,}025\ Hz$ o inferior es suficiente.
- Cerramos la ventana de App Inventor, desconectamos el teléfono, y lo reiniciamos todo. Es sorprendente comprobar que en ocasiones esto es todo lo que debemos hacer para que las cosas funcionen.

En lo que respecta a los archivos de imagen, es preferible limitarse a archivos de tipo JPEG (imágenes JPG) o PNG (imágenes con fondo transparente). Además es preferible redimensionar el archivo en un editor gráfico antes de cargarlo en App Inventor, que cargar una imagen grande para después reducir su tamaño en App Inventor.

3.6. INTRODUCCIÓN AL COMPONENTE CANVAS.

"Canvas" (**lienzo** o **panel**) es un componente que proporciona acceso a otros componentes gráficos avanzados, permite animaciones, y posibilita ciertas interacciones de usuario. Entonces, ¿en qué se diferencia el componente "Canvas" del componente "Screen"?

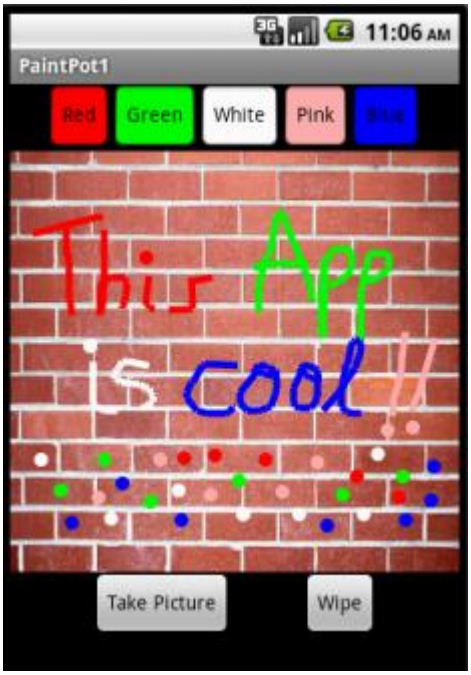
Toda app tiene una pantalla (componente "Screen1") que se genera nada más crear el proyecto. No está permitido cambiar el nombre del componente "Screen1". Esta pantalla es esencialmente el contenedor del resto de componentes con los que interacciona el usuario, incluidos los componentes "Canvas" que pueda contener el proyecto en cuestión. Por su parte, el componente "Canvas" es tanto un contenedor como un panel rectangular con sensor táctil. Es capaz de detectar eventos como tocarlo con el dedo (evento "Touched"), arrastrar el dedo sobre él (evento "Dragged"), o deslizar rápidamente el dedo sobre él (evento "Flung").

El usuario puede dibujar sobre un lienzo, y los **sprites** (las figuras animadas) pueden moverse sobre él. De hecho, si queremos añadir cualquier tipo de animación a nuestra app, debemos incluir un componente "Canvas" que la soporte. Además de incluir las ya habituales propiedades de color, anchura, altura, imagen de fondo, etc., el componente "Canvas" también nos permite fijar el color por defecto con el que el usuario pintará sobre el lienzo, y el grosor de las líneas que trazará al arrastrar el dedo sobre él.

Para ilustrar uno de los usos del componente "Canvas", construiremos una app a la que llamaremos "PaintPot1" (bote de pintura), con la que el usuario podrá dibujar sobre la pantalla del teléfono con

diferentes colores. La app también le permitirá tomar una fotografía con la cámara del dispositivo y dibujar sobre ella.

3.7. COMENZAR CON LA APP "PAINTPOT1".



En el navegador, nos conectamos a la web de App Inventor (http://ai2.appinventor.mit.edu). Comenzamos un nuevo proyecto y lo llamamos "PaintPot1".[6] En el menú superior de App Inventor, pinchamos en "Connect" y configuramos nuestro dispositivo o el emulador para hacer pruebas en vivo.

A continuación, a la derecha del Diseñador, vamos al panel de Propiedades. Dentro de las propiedades del componente "Screen1", cambiamos el título de la pantalla (propiedad "Title") a "PaintPot1". Además, fijamos su color de fondo (propiedad "BackgroundColor") a negro.

Si nos preocupa confundir el nombre del proyecto (PaintPot1) con el nombre de la pantalla (también PaintPot1), no hay problema. En App Inventor hay tres nombres clave:

- El nombre del proyecto en el que estamos trabajando. Éste también será el nombre de la aplicación cuando la empaquetemos para instalarla en un dispositivo. Notar que podemos pinchar en "Projects" y seleccionar "Save project as …" para comenzar con una nueva versión o para renombrar el proyecto.

- El nombre con el que se identifica a un componente en el panel de Componentes (en nuestro caso, la pantalla inicial se denomina "Screen1"). Como ya hemos indicado antes, en esta versión de App Inventor no está permitido cambiar el nombre de la pantalla inicial.

- El título de la pantalla, que es lo que vemos en la barra de título de la app. Este título comienza siendo igual que el nombre del componente, a saber, "Screen1". Pero el título de la pantalla puede cambiarse, como acabamos de hacer para la app "PaintPot1".

3.8. DISEÑAR LOS COMPONENTES DE "PAINTPOT1".

Para esta app usaremos los siguientes componentes:

- Cinco componentes "Button" para elegir la pintura roja, verde, blanca, rosa, o azul, y un componente "HorizontalArrangement" para organizarlos.

- Otro componente "Button" para limpiar la pantalla, y un componente "Button" más para abrir la cámara y hacer una fotografía.

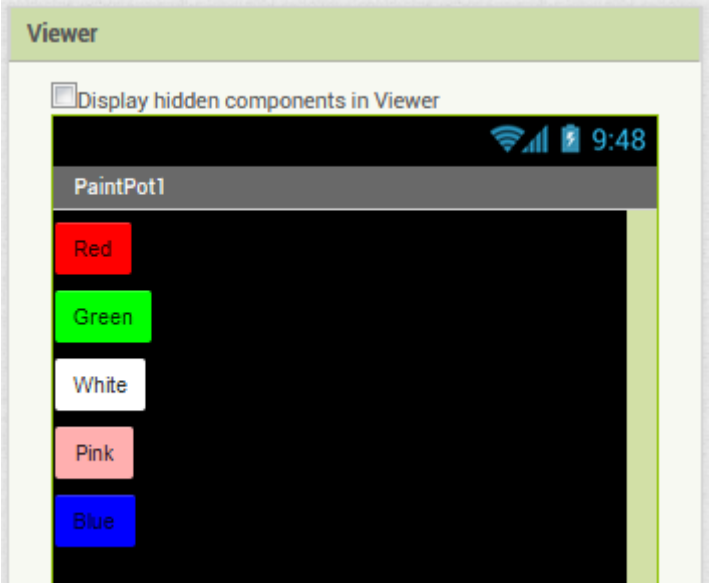
- Un componente "Canvas", que será la superficie sobre la que dibujaremos. "Canvas" tiene una propiedad "BackgroundImage" (imagen de fondo) para la que estableceremos el archivo *Wall.png* (el muro sobre el que pintaremos graffitis). Más adelante, modificaremos la app para permitir que la imagen de fondo sea una fotografía realizada por el usuario con su dispositivo.

[6] En capítulos posteriores iremos creando versiones mejoradas de este proyecto, a las que llamaremos "PaintPot2" y "PaintPot3".

CREAR LOS BOTONES PARA LOS COLORES.

En primer lugar creamos los cinco botones para los colores siguiendo estas instrucciones:

1) Arrastramos un componente "Button" al Visor del Diseñador, cambiamos su propiedad "Text" a "Red" (rojo), y fijamos su propiedad "BackgroundColor" a rojo.
2) En la lista de componentes del Visor, pinchamos en "Button1" para seleccionarlo (puede que ya esté seleccionado), y pinchamos en el botón "Rename" para cambiar su nombre de "Button1" a "RedButton". Notar que no se permiten espacios en los nombres de los componentes, por lo que es muy habitual poner en mayúsculas la primera letra de cada palabra en un nombre.
3) De forma similar, creamos cuatro botones más para el verde, blanco, rosa y azul (a los que llamaremos "GreenButton", "WhiteButton", "PinkButton", y "BlueButton") y los ubicamos en el Visor, donde aparecerán apilados en vertical. Nuestra aplicación debería parecerse a la de la figura:



Observar que en este proyecto estamos cambiando los nombres de los componentes, en lugar de dejarlos con sus nombres por defecto, como hicimos en la app "HelloPurr". Utilizar nombres con significado hace que nuestros proyectos sean mucho más legibles, y es muy útil cuando pasamos al Editor de Bloques y nos referimos a los componentes por su nombre.

USAR UN ORGANIZADOR PARA MEJORAR EL DISEÑO.

Llegados a este punto deberíamos tener cinco botones apilados en vertical. Pero en esta app queremos disponerlos en horizontal a lo largo de la parte superior de la pantalla. Esto podemos conseguirlo con el componente "HorizontalArrangement":

1) De la bandeja "Layout", arrastramos un componente "HorizontalArrangement" y lo colocamos bajo los botones.
2) En el panel de Propiedades cambiamos su propiedad "Width" a "Fill Parent", para que ocupe toda la anchura de la pantalla. Además, fijamos su propiedad "BackgroundColor" a negro, igual que la pantalla.
3) Movemos los cinco botones uno a uno dentro del componente "HorizontalArrangement1". Veremos una línea vertical azul que muestra dónde irá la pieza que estamos arrastrando.

Si ahora miramos la lista de componentes, veremos que los cinco botones cuelgan bajo el componente "HorizontalArrangement1". Esto indica que los botones son ahora subcomponentes de "HorizontalArrangement1". Notar que a su vez todos estos componentes cuelgan bajo "Screen1".

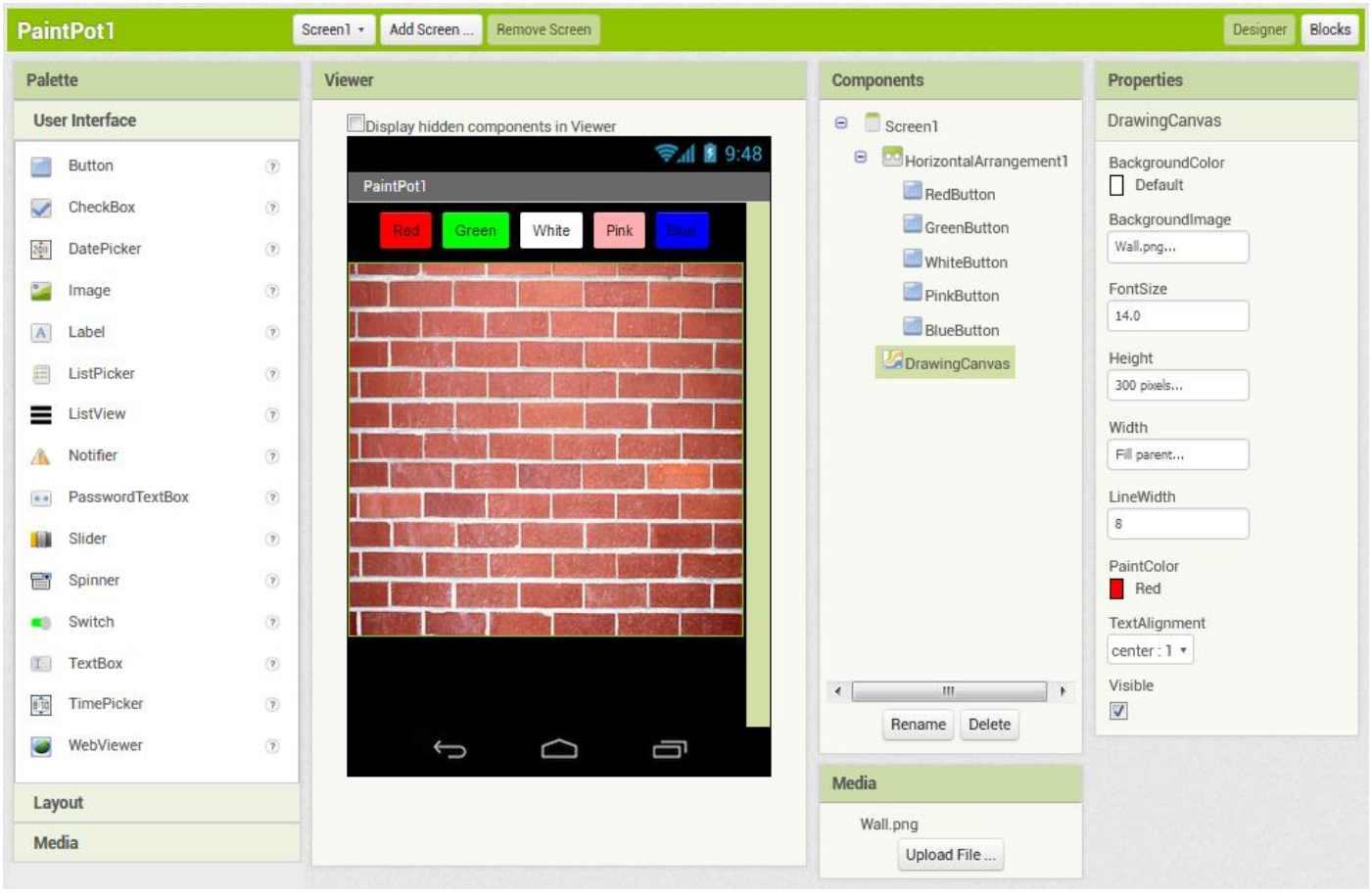
Ahora podemos centrar la fila de botones en pantalla cambiando la propiedad "AlignHorizontal" de "HorizontalArrangement1" a "Center".

AÑADIR EL LIENZO.

El siguiente paso es configurar el lienzo donde realizaremos los dibujos:

1) Acudimos a la bandeja "Drawing and Animation", y arrastramos el componente "Canvas" al Visor. Cambiamos su nombre a "DrawingCanvas". Fijamos su propiedad "Width" a "Fill parent" para que se extienda a lo largo de toda la anchura de la pantalla. Fijamos su propiedad "Height" a 300 pixeles, de forma que nos quede espacio en vertical para las dos filas de botones que tendremos en la pantalla.
2) Fijamos la propiedad "LineWidth" de "DrawingCanvas" a 5, lo cual fijará el grosor por defecto de las líneas que tracemos sobre el lienzo a 5 píxeles.
3) Vamos a la propiedad "BackgroundImage" de "DrawingCanvas", y establecemos el archivo *Wall.png* como imagen de fondo del lienzo.
4) Fijamos la propiedad "PaintColor" de "DrawingCanvas" a rojo. De esta forma, cuando el usuario arranca la aplicación y todavía no ha pinchado en ningún botón de color, la app comienza pintando con un color por defecto de rojo.

Llegados a este punto, nuestra aplicación debería parecerse a la mostrada en la figura:



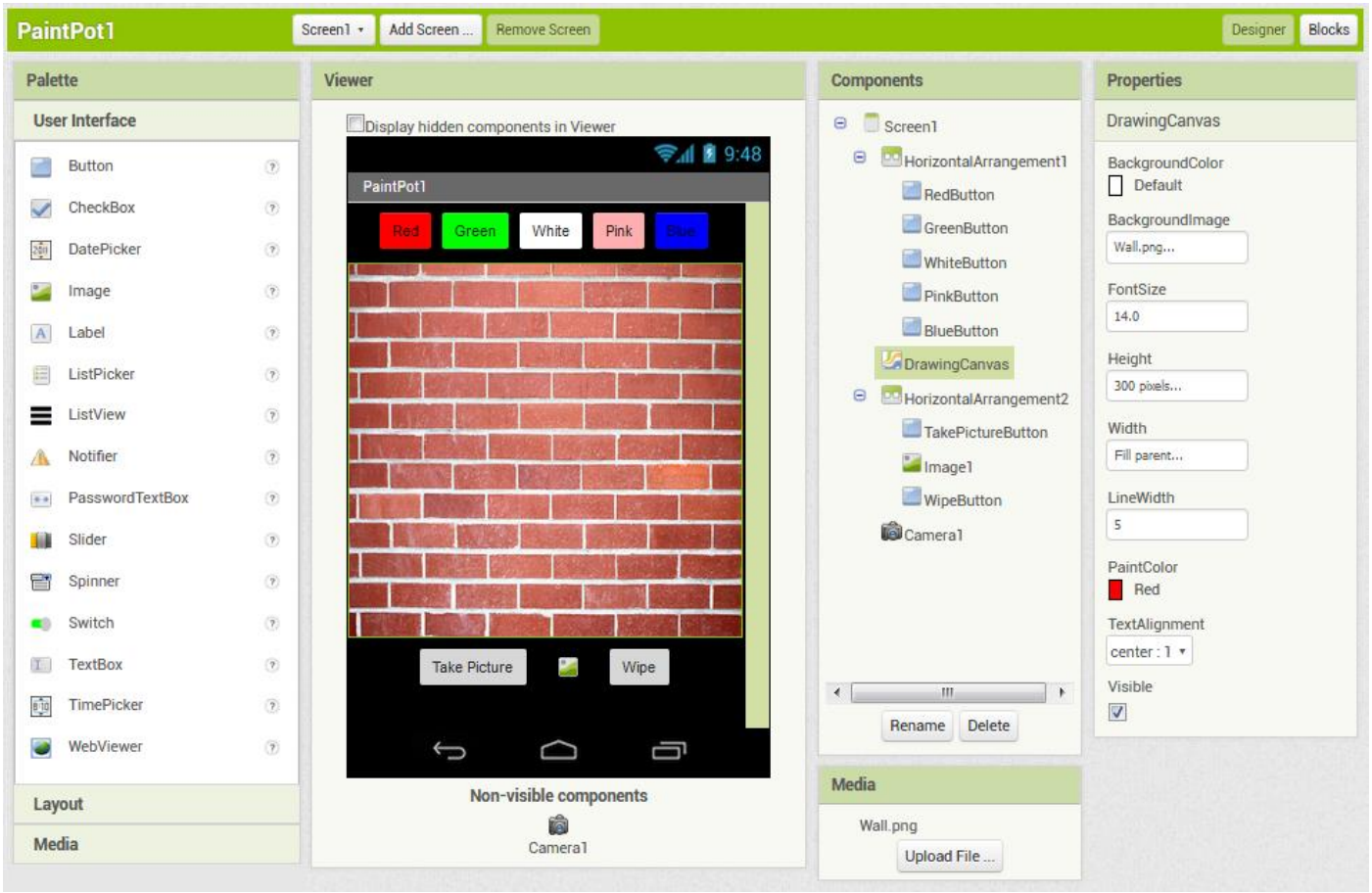
ORGANIZAR LOS BOTONES INFERIORES Y AÑADIR EL COMPONENTE CÁMARA.

Vamos a añadir los botones para tomar una fotografía y limpiar la pantalla, y el componente "Camera" para poder acceder a la cámara del dispositivo:

1) Arrastramos un segundo "HorizontalArrangement" al Visor, y lo ubicamos bajo el lienzo. Fijamos su propiedad "BackgroundColor" a negro, y su propiedad "Width" a "Fill parent".

2) A continuación arrastramos dos componentes "Button" más al Visor, y los colocamos dentro de este nuevo "HorizontalArrangement" en la parte inferior de la pantalla. Cambiamos el nombre del primer botón a "TakePictureButton" y su propiedad "Text" a "Take Picture". Cambiamos el nombre del segundo botón a "WipeButton" (botón de limpieza) y su propiedad "Text" a "Wipe" (limpiar).
3) Añadimos entre los dos botones un componente "Image" vacío que haga de separador. Fijamos su propiedad "Width" a 50 píxeles.
4) Desde la bandeja "Media" de la Paleta, arrastramos un componente "Camera" al Visor. Aparecerá en la zona de componentes no visibles.

La apariencia de la app debería parecerse a la de la figura:



3.9. AÑADIR COMPORTAMIENTOS A LOS COMPONENTES DE "PAINTPOT1".

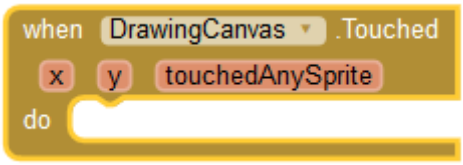
El siguiente paso es definir la forma en la que se comportan los componentes de nuestra app. Crear un programa para pintar sobre la pantalla de un teléfono puede parecer ciertamente complicado, pero App Inventor incluye bloques para gestionar los toques y deslizamientos de los dedos del usuario sobre la pantalla, para dibujar, y para hacer fotografías.

En el Diseñador hemos añadido un componente "Canvas" llamado "DrawingCanvas". Como todos los "Canvas", "DrawingCanvas" tiene un evento "Touched" (tocado) y un evento "Dragged" (arrastrado). El evento "Touched" nos permitirá dibujar un círculo cuando el usuario toque con su dedo sobre la pantalla. El evento "Dragged" nos permitirá trazar una línea cuando el usuario arrastre su dedo sobre la pantalla. Después programaremos los botones para cambiar el color del dibujo, para limpiar el lienzo, y para cambiar el fondo del lienzo a una fotografía que haya tomado el usuario con la cámara de su dispositivo móvil.

AÑADIR UN EVENTO TOUCHED PARA DIBUJAR UN PUNTO.

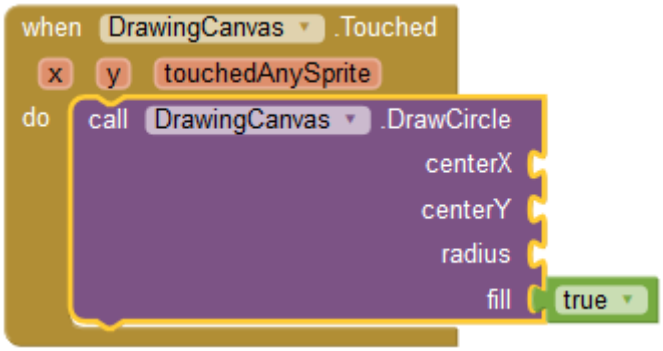
En primer lugar vamos a programar la app para que, cuando el usuario toque el componente "DrawingCanvas", se pinte una marca en el punto de contacto. Para ello hacemos lo siguiente:

1) En el Editor de Bloques seleccionamos la bandeja del componente "DrawingCanvas", y arrastramos el manejador de evento "when (DrawingCanvas).Touched". Veremos que el bloque incluye unos rectángulos naranjas adicionales llamados "x", "y", y "touchedAnySprite". Estos elementos se llaman *argumentos*, y su misión es almacenar información sobre la localización del toque.



Los manejadores de eventos que utilizamos en el capítulo previo, como "when (Button1).Click", eran muy simples porque no incorporaban información relacionada con ese evento. Pero hay otros manejadores de eventos que nos proporcionan ciertas informaciones acerca de ese evento, llamadas **argumentos**. Así, el evento "when (DrawingCanvas).Touched" proporciona los argumentos "x" e "y", los cuales nos indican las coordenadas x e y del punto donde hemos tocado en pantalla. También incluye el argumento "touchedAnySprite", que nos informa sobre qué figura animada (sprite) del "Canvas" hemos tocado (aunque esta información no la necesitaremos en este capítulo). Nosotros usaremos los arumentos "x" e "y" para conocer la posición en la que el usuario ha tocado en pantalla. Usaremos esta información para dibujar un punto en esa posición de la pantalla.

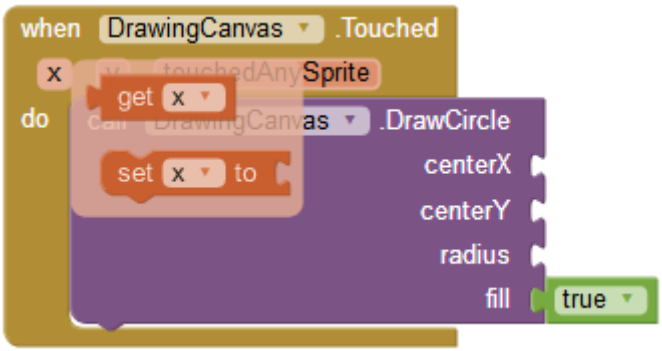
2) De la bandeja de "DrawingCanvas", arrastramos el bloque de función "call (DrawingCanvas).DrawCircle" y lo ubicamos dentro de la sección "do " del bloque "when (DrawingCanvas).Touched", como muestra la figura:



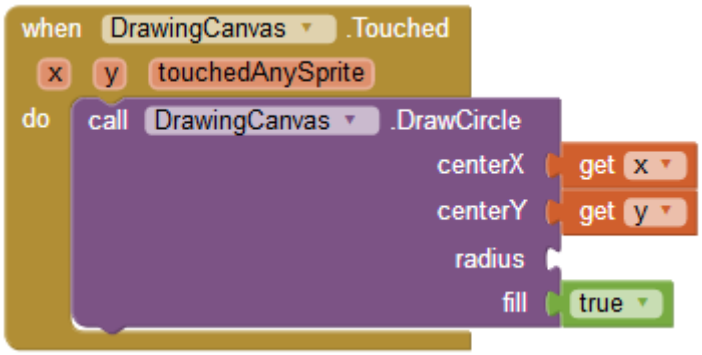
A la derecha de "call (DrawingCanvas).DrawCircle" veremos cuatro ranuras abiertas donde conectar los valores de los **parámetros** que debemos proporcionar para que el bloque funcione correctamente, a saber, "centerX", "centerY", "radius", y "fill". Los parámetros "centerX" y "centerY" especifican la localización donde se dibujará el centro del círculo, y "radius" determina el radio con el que se dibujará el círculo. (El parámetro "fill" está automáticamente conectado a un bloque lógico con el valor "true" asignado, y sirve para indicar si el círculo se pintará relleno ("true") o vacío ("false")).

El bloque "call (DrawingCanvas).DrawCircle" puede parecernos un poco confuso, porque dispone de unos parámetros "centerX" y "centerY" al estilo de los argumentos "x" e "y" del manejador de evento "when (DrawingCanvas).Touched". Tan solo debemos recordar que los valores de los *argumentos* "x" e "y" del bloque "when (DrawingCanvas).Touched" indican dónde ha tocado el usuario, mientras que los valores de los *parámetros* "centerX" y "centerY" del bloque "call (DrawingCanvas).DrawCircle" son conectores abiertos para especificar dónde se debe dibujar el círculo. Como queremos dibujar el círculo allá donde haya tocado el usuario, usaremos los valores de los argumentos "x" e "y" de "when

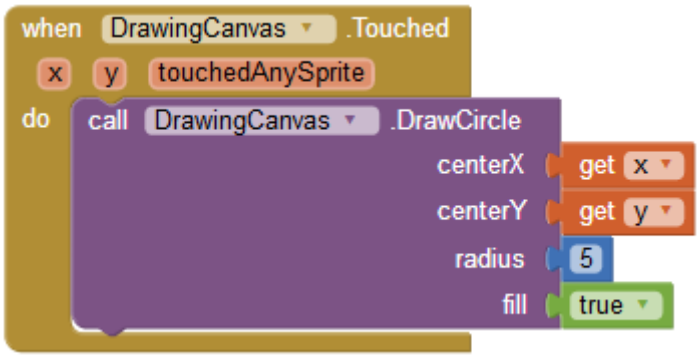
(DrawingCanvas).Touched" como valores para los parámetros "centerX" y "centerY" del bloque "call (DrawingCanvas).DrawCircle".



3) Podemos acceder a los valores de los argumentos de los manejadores de evento pasando el ratón sobre los argumentos del bloque en cuestión, como muestra la figura previa. Así pues, arrastramos sendos bloques "get ()" para los valores de los argumentos "x" e "y", y los pegamos a los conectores de los parámetros "centerX" y "centerY" del bloque "call (DrawingCanvas).DrawCircle", como muestra la siguiente figura.



4) Ahora debemos especificar el valor del radio para dibujar el círculo en el parámetro "radius". El radio se mide en píxeles, que es el punto más pequeño que se puede dibujar en la pantalla del dispositivo. Por ahora, vamos a *fijarlo* a 5 píxeles. Para ello, pinchamos en una zona en blanco del área de trabajo, escribimos un 5, y pulsamos la tecla RETURN. (Con esto creamos un bloque numérico automáticamente). Ahora, conectamos este bloque al conector "radius".



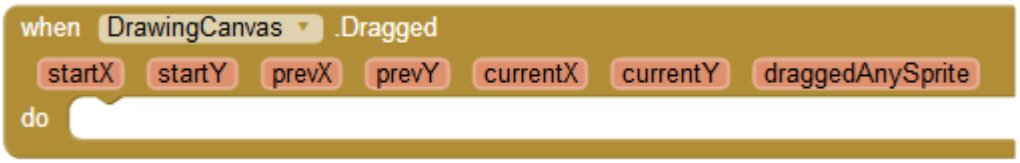
Vamos a probar cómo funciona nuestra app en el dispositivo. Cuando toquemos el "DrawingCanvas", nuestro dedo debería dejar un punto allá donde hayamos tocado. Los puntos serán rojos, porque previamente fijamos la propiedad "PaintColor" de "DrawingCanvas" a rojo (en caso contrario, el color por defecto es el negro).

AÑADIR UN EVENTO DRAGGED PARA DIBUJAR UNA LÍNEA.

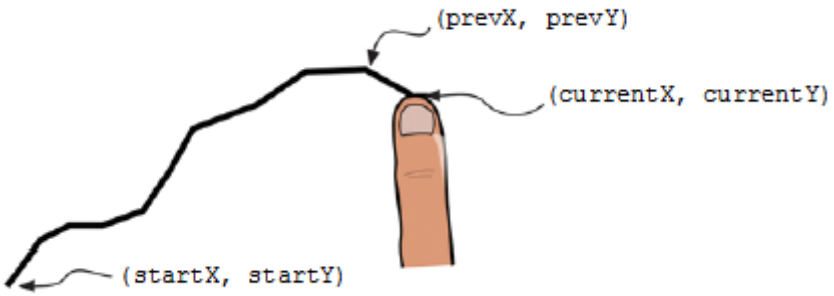
A continuación vamos a añadir el manejador del evento "Dragged". Este evento se diferencia del evento "Touched" en lo siguiente:

- El evento "Touched" se lanza cuando ponemos nuestro dedo sobre el "DrawingCanvas" y lo levantamos sin moverlo.
- El evento "Dragged" se lanza cuando ponemos nuestro dedo sobre el "DrawingCanvas" y lo movemos *suavemente* mientras lo mantenemos en contacto con la pantalla.

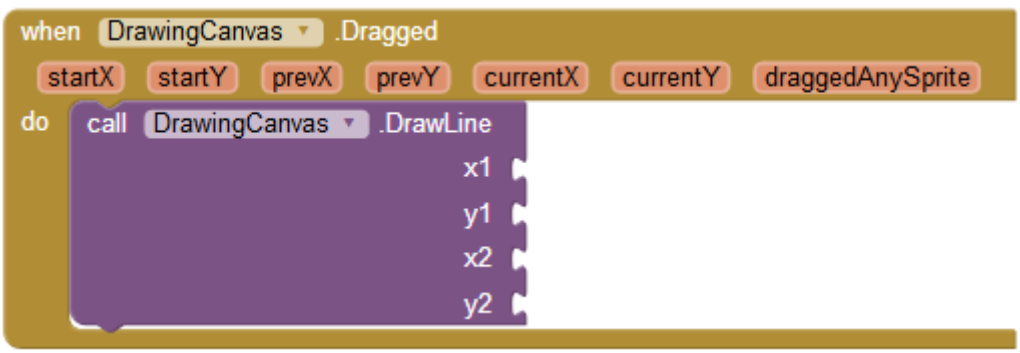
1) De la bandeja de "DrawingCanvas" arrastramos el bloque "when (DrawingCanvas).Dragged" al área de trabajo. Este manejador se muestra en la figura, donde vemos que viene con los siguientes argumentos:
   - "startX", "startY": identifican la posición de nuestro dedo en el punto donde empezó a deslizarse sobre la pantalla. Estos argumentos solo cambian cuando el usuario empieza un nuevo arrastre de su dedo sobre la pantalla.
   - "currentX", "currentY": identifican la posición actual de nuestro dedo en la pantalla.
   - "prevX", "prevY": identifican la posición inmediatamente anterior del dedo en la pantalla. Cada vez que nuestro dedo para o cambia de dirección, su posición se almacena en estos dos argumentos.
   - "draggedAnySprite": es un valor Booleano que estará a "true" si el usuario desliza su dedo sobre un sprite situado en el lienzo. Aquí no usaremos este argumento.



¿Para qué sirven los argumentos "currentX", "currentY", y "prevX", "prevY"? Cuando arrastramos nuestro dedo trazando un arco sobre la pantalla, dibujamos una línea curva que sigue el camino recorrido por el dedo. Pero lo que realmente ocurre es que se dibujan muchas líneas rectas diminutas, por lo que los valores de (prevX, prevY) y (currentX, currentY) están cambiando constantemente (ver figura). Esos valores están permanentemente accesibles a través de dichos argumentos.

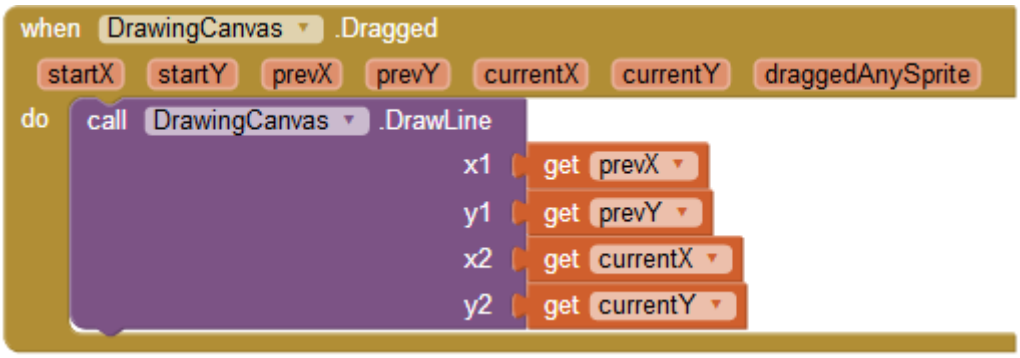


2) De la bandeja de "DrawingCanvas", cogemos el bloque "call (DrawingCanvas).DrawLine" y lo colocamos dentro de la sección "do" del manejador de evento "when (DrawingCanvas).Dragged".



Este bloque tiene cuatro parámetros, dos para cada punto que determina la línea a dibujar. Así, $(x1, y1)$ son las coordenadas de uno de los puntos, y $(x2, y2)$ las coordenadas del otro punto. ¿Qué valores crees que necesitan estos parámetros? Recordemos que el evento "Dragged" puede llamarse muchas veces mientras deslizamos nuestro dedo a lo largo de "DrawingCanvas". La aplicación dibujará una línea diminuta cada vez que nuestro dedo se mueva, desde (prevX, prevY) hasta (currentX, currentY).

3) Tomamos los bloques "get ()" que necesitemos para obtener los valores de los argumentos "x1", "y1", "x2", e "y2". Conectamos un bloque "get (prevX)" al conector del parámetro "x1" y un bloque "get (prevY)" al conector del parámetro "y1". Análogamente, conectamos sendos bloques "get (currentX)" y "get (currentY)" a los conectores "x2" e "y2", respectivamente. El programa debe quedar así:



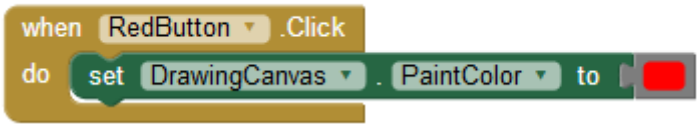
Ahora probamos cómo funciona la app en nuestro dispositivo. Cada vez que deslicemos el dedo sobre la pantalla, deberían dibujarse líneas y curvas. También podemos tocar la pantalla para dibujar puntos.

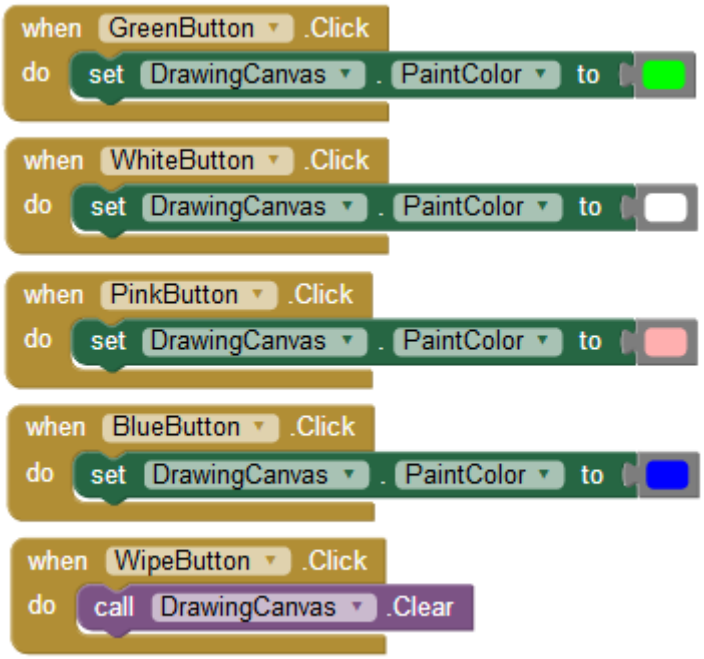
CAMBIAR EL COLOR Y BORRAR LA PANTALLA.

Nuestra app permite que el usuario dibuje, pero de momento solo en rojo. Ahora vamos a añadir los manejadores de evento para los botones que le permiten al usuario cambiar el color con el que pinta. También añadiremos un manejador de evento al botón "WipeButton" para que el usuario pueda borrar la pantalla. Para ello:

1) En el Editor de Bloques abrimos la bandeja de "RedButton" y arrastramos el manejador de evento "when (RedButton).Click" al área de trabajo.
2) Abrimos la bandeja de "DrawingCanvas", seleccionamos el bloque "set (DrawingCanvas).(PaintColor) to". (Este bloque está casi al final de los bloques disponibles en la bandeja), y lo ubicamos dentro de la sección "do" del bloque "when (RedButton).Click".
3) Abrimos la bandeja "Colors", seleccionamos el bloque para el color rojo, y lo conectamos al bloque "set (DrawingCanvas).(PaintColor) to".
4) Repetimos los pasos $2 - 4$ para los botones de los colores verde, blanco, rosa, y azul.
5) El último botón a configurar es "WipeButton". Sacamos el bloque "when (WipeButton).Click" de la bandeja de bloques de "WipeButton". A continuación vamos a la bandeja de "DrawingCanvas", seleccionamos el bloque "call (DrawingCanvas).Clear" y lo ponemos dentro del manejador de evento "when (WipeButton).Click".

Comprobamos que nuestro código queda como en la figura:





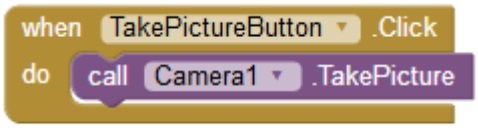
Como de costumbre, probamos en nuestro dispositivo el funcionamiento de la aplicación. Tocamos en cada uno de los botones de color para comprobar que los puntos y líneas que pintamos por pantalla se dibujan con el color seleccionado. Tocamos el botón "WipeButton" para comprobar que la app borra la pantalla de puntos y trazos.

PERMITIR AL USUARIO HACER UNA FOTOGRAFÍA.

Las apps de App Inventor pueden interactuar con potentes características internas de los dispositivos Android, incluyendo la cámara. A continuación vamos a permitir que el usuario cambie la imagen de fondo del dibujo sacando una fotografía con la cámara del dispositivo.

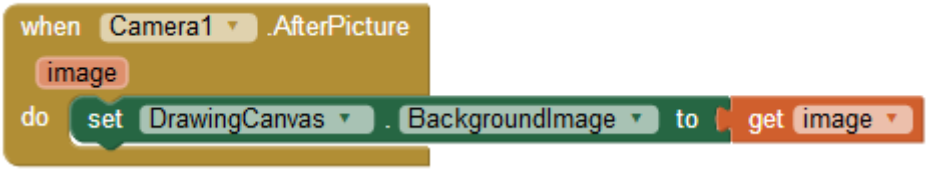
El componente "Camera" dispone de dos bloques básicos: El bloque de función "call (Camera1).TakePicture" lanza la aplicación de la cámara en el dispositivo. El manejador de evento "when (Camera1).AfterPicture" se activa cuando el usuario ha terminado de sacar la fotografía. En esta aplicación vamos a añadir bloques dentro del manejador "when (Camera1).AfterPicture" para establecer la propiedad "BackgroundImage" del componente "DrawingCanvas" a la fotografía que el usuario acaba de sacar. Para ello:

1) Abrimos la bandeja de bloques de "TakePictureButton" y arrastramos el bloque "when (TakePictureButton).Click" al área de trabajo.
2) De la bandeja de "Camera1", seleccionamos el bloque "call (Camera1).TakePicture" y lo ponemos dentro del bloque manejador de evento "when (TakePictureButton).Click". El programa debe quedar como muestra la figura:



3) De la bandeja de "Camera1", arrastramos el manejador de evento "when (Camera1).AfterPicture" al área de trabajo.
4) De la bandeja de "DrawingCanvas" llevamos el bloque "set (DrawingCanvas).BackgroundImage to" dentro del bloque "when (Camera1).AfterPicture".
5) El manejador de evento "when (Camera1).AfterPicture" incluye un argumento llamado "image", que representa la fotografía que se acaba de tomar. Podemos hacer una referencia a esta imagen sacando un bloque "get (image)" del manejador "when (Camera1).AfterPicture". A continuación, conectamos el

bloque "get (image)" al conector abierto del bloque "set (DrawingCanvas).BackgroundImage". El programa debería quedar como muestra la figura:



Comprobamos el funcionamiento de la app en nuestro dispositivo Android. Tocamos el botón "TakePictureButton" y hacemos una fotografía. La imagen de fondo del muro debería cambiar a la fotografía que acabamos de hacer, y a partir de ahora, podemos dibujar sobre esta fotografía.

3.10. INTERACCIÓN DE USUARIO CON LA PANTALLA TÁCTIL.

Una de las características distintivas de las apps de los teléfonos móviles frente a los programas de PC es el uso de la pantalla táctil, lo que le permite al usuario interactuar de forma directa con esas apps. Conforme adquiramos más experiencia con App Inventor, veremos que el usuario puede interactuar de distintas maneras con los diversos componentes disponibles (etiquetas, botones, imágenes, notificadores, lienzos, etc.).

Por ejemplo, las etiquetas son simplemente eso, etiquetas: Una etiqueta no puede reproducir sonidos ni moverse, no podemos arrastrarlas ni pinchar sobre ellas; las etiquetas simplemente pueden mostrar un texto para informar al usuario acerca de otro elemento o componente de la app. En definitiva, el usuario no puede interactuar físicamente con las etiquetas. Y lo mismo pasa con los componentes "Image".

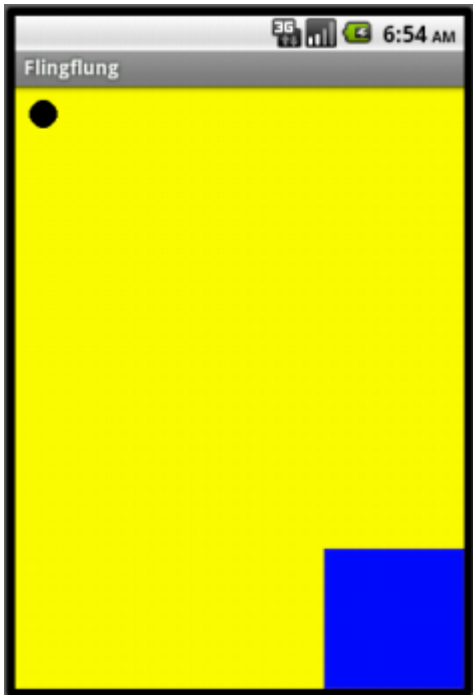
Pero hay otros componentes con los que el usuario sí que puede interactuar, como los botones, los lienzos, las pelotas (componente "Ball"), y las figuras animadas (componente "ImageSprite"). Hasta ahora solo hemos usado botones y lienzos, pero no las pelotas ni las figuras animadas (en general, **sprites**). El término *sprite* fue acuñado por los primeros programadores de videojuegos en la década de 1970 para referirse a las imágenes que son capaces de moverse por la pantalla. Los sprites suelen usarse como personajes animados en los videojuegos, como Super Mario, Pac - Man, etc. Si estamos creando una app que use una pelota o una figura animada, debemos utilizar un lienzo que las albergue y sobre el que se puedan mover.

Tanto los lienzos como los sprites responden ante los eventos "Touched", "Dragged", y "Flung". En la app "PaintPot1" ya usamos los eventos "Touched" y "Dragged" para dibujar puntos y líneas sobre un lienzo, y en esta sección vamos a presentar el evento "Flung". El evento "Flung" se activa cada vez que el usuario desliza *velozmente* su dedo dobre un lienzo. No debemos confundir el evento "Flung" con el evento "Dragged", que se activa cuando el usuario arrastra su dedo *suavemente* sobre un lienzo.

Event/Component	Canvas	Ball	Sprite
`Dragged`—Collects coordinates related to the position of the user's finger when they have dragged it across the screen	✓	✓	✓
`Flung`—Provides arguments such as the initial position and velocity of the swipe when a fling gesture (or a quick swipe) is made on the screen	✓	✓	✓
`Touched`—Provides position coordinates when touch-down and touch-up have occurred	✓	✓	✓

Como en el caso de los lienzos, los sprites también pueden ser tocados ("Touched"), arrastrados ("Dragged"), o movidos velozmente ("Flung") por el usuario. Si en la app "PaintPot1" usamos los eventos "Touched" y "Dragged" para pintar sobre un lienzo, en este caso usaremos los eventos "Dragged" y "Flung" para permitirle al usuario controlar el movimiento de los sprites a través de la pantalla táctil del terminal móvil.

La tabla previa lista estos tres eventos, junto con los componentes con los que pueden usarse. Siempre es bueno saber de antemano qué componentes pueden arrastrarse o deslizarse, y cuáles no. En todo caso, el equipo del Instituto Tecnológico de Masachusets (MIT) encargado del mantenimiento de App Inventor está constantemente actualizando y mejorando los componentes disponibles en esta herramienta. Para comprobar lo que un componente puede y no puede hacer, es muy recomendable visitar la web http://ai2.appinventor.mit.edu/reference/components/.



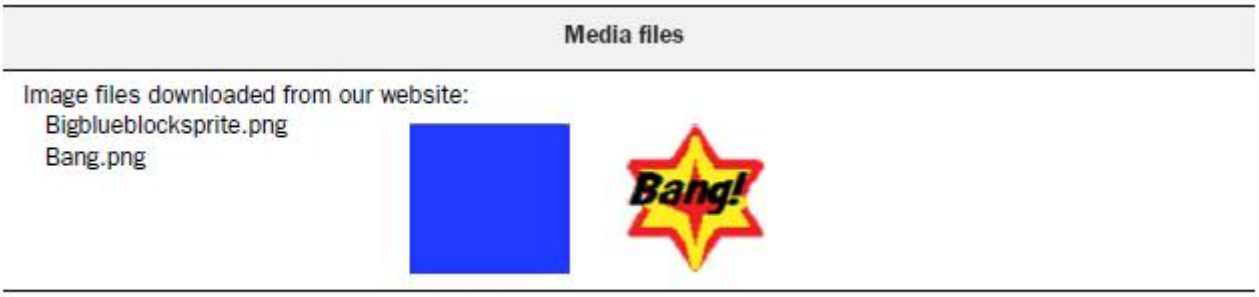
3.11. COMENZAR CON LA APP "FLINGFLUNG".

En esta sección vamos a construir una app llamada "FlingFlung" que muestre el uso de los eventos "Flung" y "Dragged" aplicados a pelotas y a figuras animadas. El usuario desplaza un bloque por la pantalla arrastrándolo con su dedo. Cuando el bloque está en posición, el usuario lanza una pelota contra el bloque. Cuando la pelota impacta contra el bloque, éste cambia su imagen habitual por la de una explosión.

3.12. DISEÑAR LOS COMPONENTES DE "FLINGFLUNG".

La tabla a continuación lista los componentes necesarios para esta app y los ajustes de sus propiedades. Añádelos al Diseñador de App Inventor para conseguir una interfaz de usuario como la mostrada en la figura.

Flingflung			
Screen1 properties	`AlignHorizontal`: Center `Scrollable`: No `Icon`: Bang.png (downloaded from our website) `AlignVertical`: Center `Title`: Flingflung `ScreenOrientation`: Portrait		
Components	**What do I rename it?**	**What does it do?**	**What properties do I set?**
`Canvas`	`Canvas`	Allows you to position sprites	`BackgroundColor`: Yellow
`Ball` Palette group: Animation	`Ball1`	Can be flung toward the block target	`Radius`: 10 `Interval`: 10
`Imagesprite` Palette group: Animation	`Blocksprite`	Stays still unless repositioned by the user, who can drag it anywhere on the screen to provide a target toward which to fling the ball	`Picture`: Upload the matching image file for the `imagesprite`, Bigblueblocksprite.png. This `imagesprite` has another image called Bang.png that you need to upload to Media. You access it from the Blocks palette.



Para el sprite del bloque necesitaremos dos imágenes: los archivos *Bigblueblocksprite.png* y *Bang.png*. (Para la pelota no necesitaremos ninguna imagen, porque el componente "Ball" siempre adquiere la forma de una pelota de color y radio configurables).

3.13. AÑADIR COMPORTAMIENTOS A LOS COMPONENTES DE "FLINGFLUNG".

PROPIEDADES "HEADING", "SPEED", E "INTERVAL" DE LOS SPRITES.

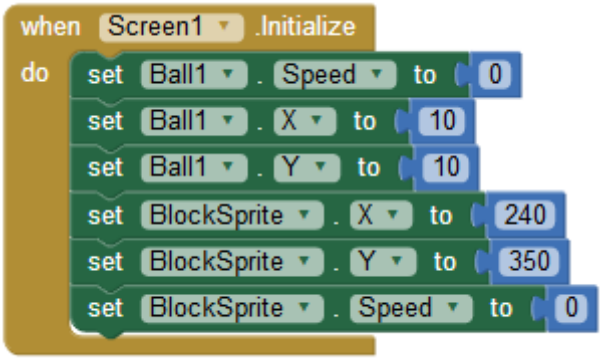
Los componentes "Ball" e "ImageSprite" incluyen unas propiedades que nos permiten controlar directamente su movimiento: se trata de las propiedades "Heading", "Speed", e "Interval".

La propiedad "Heading" nos permite especificar la dirección del movimiento del sprite, y admite valores entre 0 y 360 grados, donde 0 es hacia la derecha, 90 hacia arriba, 180 hacia la izquierda, y 270 hacia abajo.

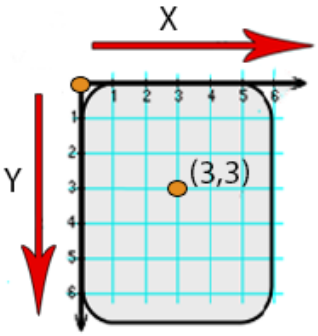
Por su parte, las propiedades "Speed" e "Interval" permiten especificar de forma conjunta la rapidez del movimiento: "Speed" indica cuantos píxeles avanza el sprite cada vez que transcurren el número de milisegundos especificados en "Interval". (Recordar que mil milisegundos son un segundo). Por ejemplo, si el valor de "Interval" fuese 50 y el valor de "Speed" fuese 10, el sprite se movería 10 píxeles cada 50 milisegundos. Para conseguir que la pelota se deslice suavemente, en el Diseñador hemos fijado su intervalo a un valor de 10. El valor de su propiedad "Speed" lo fijaremos mediante código en el Editor de Bloques.

INICIALIZACIÓN DEL PROGRAMA.

Para empezar, y nada más arrancar la app, hacemos que la pelota comience inmóvil en la parte superior izquierda de la pantalla, y que el bloque aparezca en la parte inferior derecha. Como ya sabemos, para realizar alguna acción justo al comezar la app, debemos usar el manejador de evento "when (Screen1).Initialize". Para ubicar a la pelota y al bloque en sus posiciones iniciales, dentro del manejador configuramos las propiedades "X" e "Y" de la pelota y del bloque a los valores adecuados, y sus propiedades "Speed" a cero.



NOTA: Los lienzos en los que se mueven los sprites son tablas de píxeles, donde un píxel es el punto más pequeño que puede dibujarse en la pantalla de un teléfono móvil (o de un ordenador). Cada uno de estos píxeles tiene una localización sobre el lienzo, que se especifica usando un sistema coordenado Cartesiano con el origen en la esquina superior izquierda del lienzo, con la coordenada x incrementándose hacia la derecha, y con la coordenada y incrementándose hacia abajo. Con este sistema de coordenadas, podemos indicar la posición de un sprite sobre el lienzo en el que se mueve, como hemos hecho en el bloque de código previo. Hablaremos más sobre todo esto en capítulos posteriores.

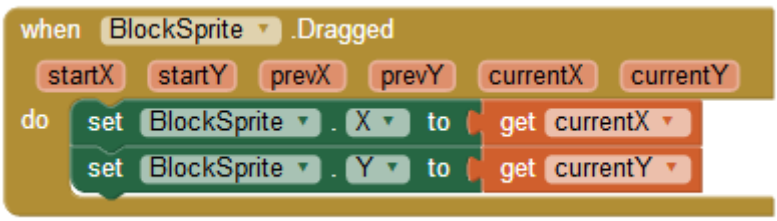


PROGRAMAR EL ARRASTRE DEL BLOQUE.

En esta app queremos que el usuario pueda arrastrar el bloque a una nueva ubicación desde su posición inicial, para lo cual usaremos el manejador de eventos "when (BlockSprite).Dragged". Como ya sabemos de la app "PaintPot1" este manejador reporta numerosos argumentos, pero en esta ocasión únicamente necesitaremos los argumentos "currentX" y "currentY" que indican el lugar en el que el dedo del usuario deja de tocar la pantalla después de haberlo arrastrado.

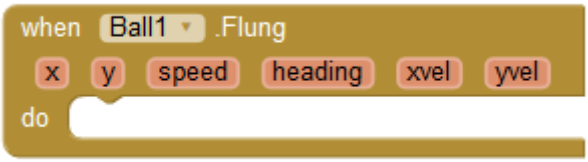
Para programar el comportamiento del bloque, hacemos lo siguiente:

1) De la bandeja del componente "BlockSprite", sacamos el manejador "when (BlockSprite).Dragged".
2) De la bandeja de "BlockSprite", sacamos los bloques "set (BlockSprite).(X) to" y "set (BlockSprite).(Y) to" y los conectamos dentro de la sección "do" del manejador.
3) Como queremos que el manejador fije las coordenadas x e y de la nueva posición del bloque al valor actual de los argumentos "currentX" y "currentY", pasamos el ratón sobre estos argumentos para obtener los bloques "get (currentX)" y "get (currentY)". Estos bloques almacenan las coordenadas x e y del punto donde el dedo del usuario deja de tocar la pantalla tras haberlo arrastrado.
4) Finalmente, conectamos los bloque "get (currentX)" y "get (currentY)" a los bloques "set (BlockSprite).(X)" y "set (BlockSprite).(Y)", respectivamente. Nuestro código debería quedar como muestra la figura:



PROGRAMAR EL LANZAMIENTO DE LA PELOTA.

Tras haber reubicado el bloque azul, el usuario desliza velozmente su dedo sobre la pelota para lanzarla contra el bloque. Para implementar este comportamiento, necesitaremos usar el evento "Flung".



Como vemos en la figura, el manejador del evento "Flung" reporta hasta seis argumentos:

- "x": Es la coordenada x del punto donde el usuario inicia el deslizamiento de su dedo sobre la pantalla.
- "y": Es la coordenada y del punto donde el usuario comienza el deslizamiento de su dedo sobre la pantalla.
- "speed": Indica la rapidez del movimiento de deslizamiento realizado por el usuario, en píxeles por segundo. No confundir con la propiedad "Speed" de los sprites.

- "heading": Especifica la dirección del movimiento de deslizamiento, en grados. 0 grados es hacia la derecha. No confundir con la propiedad "Heading" de los sprites.
- "xvel": Es la rapidez del movimiento de deslizamiento en la dirección x.
- "yvel": Es la rapidez del movimiento de deslizamiento en la dirección y.

Como ya hemos visto en varias ocasiones, no es obligatorio que nuestros programas usen todos los argumentos de los eventos "Touched", "Dragged", y "Flung". Por ejemplo, en esta app gestionaremos el lanzamiento de la pelota usando únicamente la rapidez (argumento "speed") y la dirección (argumento "heading") del evento "Flung".

Vamos a programar el comportamiento de la pelota. Como sabremos de ciencias, para especificar la velocidad de un objeto hemos de indicar tanto la rapidez como la dirección de su movimiento. Así pues, para darle a la pelota la velocidad del movimiento de deslizamiento realizado por el usuario sobre ella, debemos darle al componente "Ball1" unos nuevos valores para sus propiedades "Speed" y "Heading". Para ello, y dentro del manejador "when (Ball1).Flung", le damos a las propiedades "Speed" y "Heading" de "Ball1" unos valores proporcionales a los valores de los argumentos "speed" y "heading" reportados por el manejador. Para ello, hacemos lo siguiente:

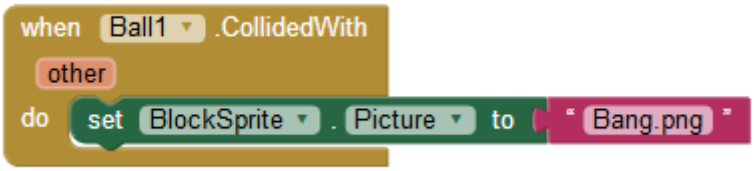
1) Extraemos un bloque manejador de evento "when (Ball1).Flung" de la bandeja "Ball1".
2) Extraemos los bloques "set (Ball1).(Heading)" y "set (Ball1).(Speed)" de la bandeja "Ball1".
3) Pasamos el ratón sobre los argumentos "speed" y "heading" del manejador para obtener sendos bloques "get".
4) Conectamos el bloque "get (heading)" al bloque "set (Ball1).(Heading)".
5) Obtenemos un bloque de producto de la bandeja "Math", y multiplicamos el bloque "get (speed)" por 5. (Multiplicamos el valor de la rapidez del movimiento de deslizamiento por 5 para hacer que la pelota se mueva más rápido).
6) Conectamos este bloque de producto al bloque "set (Ball1).(Speed)".

PROGRAMAR LA COLISIÓN ENTRE LA PELOTA Y EL BLOQUE.

Ahora queremos que cuando la pelota colisione contra el bloque, el bloque cambie su imagen normal (*Bigblueblocksprite.png*) por la imagen de una explosión (*Bang.png*). Para programar este comportamiento, hacemos lo siguiente:

1) El evento que le permite a un sprite detectar si ha colisionado con otro sprite es "CollidedWith". Por consiguiente, de la bandeja del componente "Ball1" sacamos un bloque manejador "when (Ball1).CollidedWith".
2) Para cambiar la imagen del bloque debemos modificar su propiedad "Picture". Para ello, acudimos a la bandeja de "BlockSprite" y extraemos un bloque "set (BlockSprite).(Picture) to", que conectamos dentro de la sección "do" del manejador.
3) Para especificar el nombre del archivo de imagen al que queremos fijar la propiedad "Picture" del bloque, vamos a la bandeja "Text" y sacamos un bloque de texto, que rellenaremos con el nombre del archivo de imagen de la exposión, *Bang.png*.
4) Finalmente, conectamos el bloque de texto al bloque "set (BlockSprite).(Picture) to".

Nuestro programa debería quedar como el de la figura:



Y con esto ya hemos terminado. El código ha sido bastante sencillo, ahora vamos a probarlo. Al arrancar la app la pelota debería aparecer inmóvil en la parte superior izquierda de la pantalla, coordenadas $(x, y) = (10, 10)$, y el bloque en la parte inferior derecha, coordenadas $(x, y) = (240, 350)$. A continuación, podemos actuar sobre el bloque (arrastrándolo a una nueva posición), o sobre la pelota (deslizándola velozmente para lanzarla en una cierta dirección y a una cierta velocidad). Cuando deslizamos la pelota, el manejador de evento "when (Ball1).Flung" registra la rapidez y la dirección de este movimiento de deslizamiento para ajustar la rapidez y la dirección del movimiento de la pelota. Si la pelota impacta sobre el bloque, el programa cambia la imagen del bloque azul por la imagen de una explosión.

3.14. MÁS COMPONENTES PARA LA INTERACCIÓN DEL USUARIO.

En las secciones previas hemos creado la app "FlingIt" usando tres eventos de interacción del usuario. Pero hay muchos más eventos y componentes que nos podrían ser útiles para desarrollar nuestras propias aplicaciones. Si echamos un vistazo a la tabla a continuación tal vez identifiquemos algunos de los más usados en las apps que tenemos instaladas en nuestros teléfonos móviles. Como podemos ver, algunos componentes son capaces de registrar más eventos que otros.

Event/Component	Button	Canvas	Ball	Sprite	Accelerometer
`Click`—Indicates that a user has clicked a button	✓				
`GotFocus`—Indicates that the cursor has moved over the button so the user can now click it	✓				
`LostFocus`—Indicates that the cursor has moved away from the button so it's no longer possible for the user to click it	✓				
`Dragged`—Collects coordinates related to the position of the user's finger when they have dragged it across the screen		✓	✓	✓	
`Flung`—Provides arguments such as the initial position and velocity of the swipe when a fling gesture (or a quick swipe) is made on the screen		✓	✓	✓	
`TouchDown`—Provides coordinates of the touch when the user begins touching the sprite		✓	✓	✓	
`TouchUp`—Provides coordinates of the position when the user stops touching the sprite		✓	✓	✓	
`Touched`—Provides position coordinates when touch-down and touch-up have occurred		✓	✓	✓	
`Shaking`—Called repeatedly when the Android device is physically shaken					✓

Como de costumbre, podemos intetar mejorar las apps que hemos construido en este capítulo de múltiples formas creativas e imaginativas. Para ello, podemos probar a usar los componentes de interacción de usuario listados en la siguiente tabla. Pero si no se nos ocurre ninguna idea, he aquí unas cuantas sugerencias:

1) En la aplicación "FlingIt" podemos incrementar la dificultad del juego grabando dos bandas sonoras distintas, una rápida y otra lenta. Para ello, deberíamos incorporar dos botones de selección de nivel para que el usuario pueda elegir la dificultad deljuego.

2) De nuevo para la aplicación "FlingIt", podemos cambiar los sprites por imágenes de instrumentos de percusión que sugieran los efectos de sonido que reproducirán. Por ejemplo, podemos usar la imagen de un tambor para el sprite "TouchitSprite", un silbato para el sprite "FlingitSprite", etc.
3) En la app "FlingFlung", cuando la pelota impacte contra el bloque, podemos cambiar la imagen del bloque para que sea la de un bloque agrietado o partido.

Las apps que hemos desarrollado hasta ahora parecen videojuegos, y de momento funcionan bien, pero les faltan unas cuantas cosas: puntuaciones, niveles, animaciones, etc. En próximos capítulos aprenderemos a implementar todas estas mejoras. Cuando dominemos esas técnicas, tal vez queramos volver a estas apps y probar a hacer lo siguente:

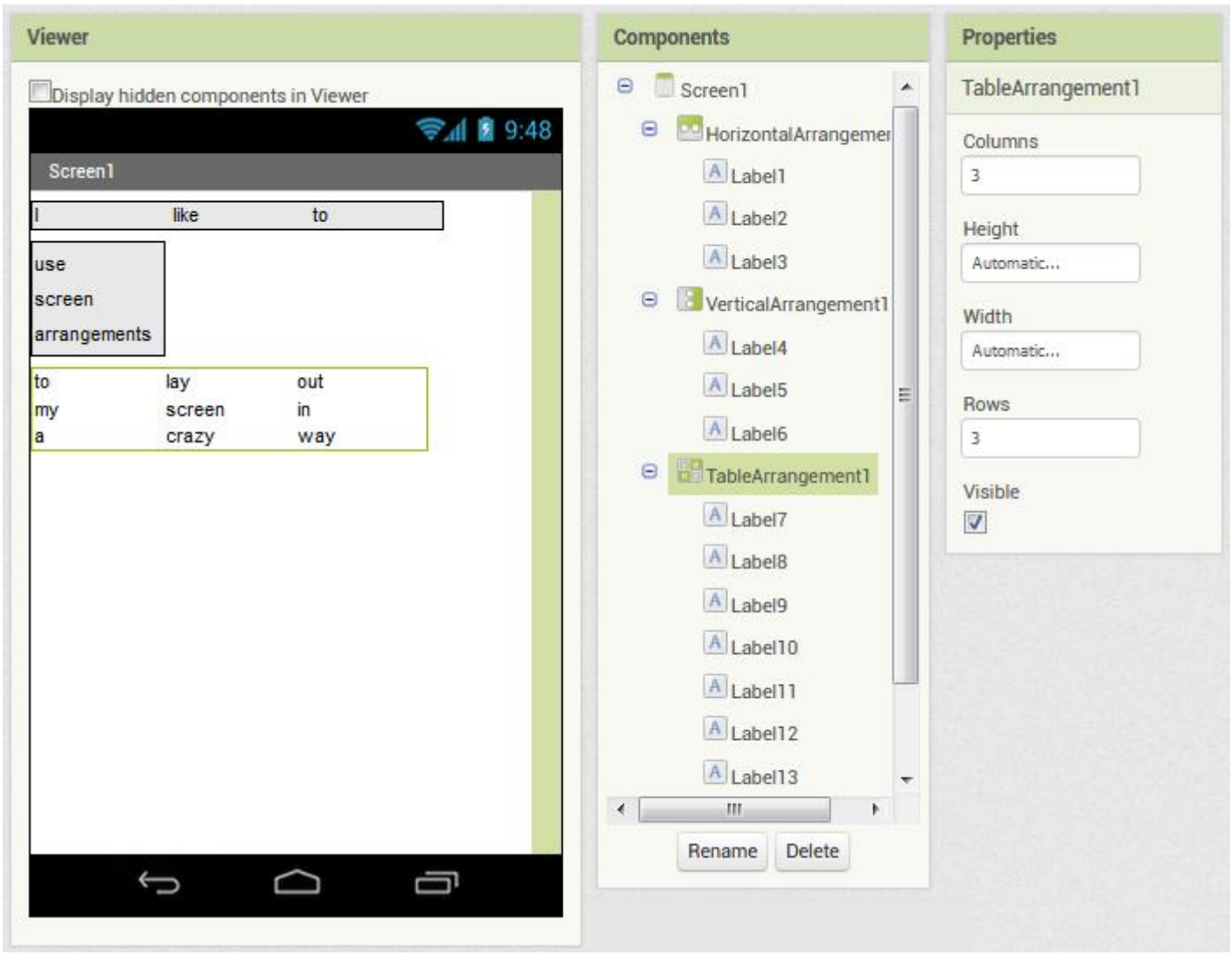
1) Incrementar el nivel de dificultad de la app "FlingIt" en función de cómo juegue el usuario, y no eligiéndolo desde el principio.
2) En lugar de usar el archivo *Flingitmachine.png* como fondo para el lienzo, podemos crear nuestro propio fondo. Para ello podemos optar por dibujarlo a mano y escanearlo, o usar un software de dibujo. Las tres áreas del fondo deberían estar lo suficientemente lejos como para que el deslizamiento del dedo del usuario no invada la zona destinada al toque.
3) En próximos capítulos parenderemos a llevar la puntuación de un videojuego. Cuando sepamos cómo hacerlo, podemos modificar la aplicación "FlingIt" para contabilizar la puntuación del jugador, incluyendo bandas sonoras con un número específico y conocido de menciones a las acciones de deslizar, arrastrar, y tocar.
4) En la aplicación "FlingIt", en vez de tener formas estáticas que respndan a los toques, arrastres, o deslizamientos de los dedos del usuario, podemos usar sprites que realicen una acción (además de reproducir un sonido) en respuesta al evento. Por ejemplo, en respuesta a un toque, el tambor que representa al sprite "TouchitSprite" podría vibrar.

3.15. EJERCICIOS DEL CAPÍTULO 3.

Ejercicio 3.1. Organizadores.
A menos que le indiquemos lo contrario, App Inventor suele apilar los componentes que añadimos a la pantalla de nuestra app. Por ejemplo, si agregamos tres componentes al Visor del Diseñador, App Inventor los ubica uno encima de otro. Sin embargo, y como vimos en la sección 3.2, los organizadores nos permiten distribuir estos componentes verticalmente ("VerticalArrangement"), horizontalmente ("HorizontalArrangement"), o en forma de tabla ("TableArrangement"). Para practicar el uso de organizadores, crea la interfaz de usuario mostrada en la figura en base a etiquetas.

Ejercicio 3.2. Arrastrar la pelota dentro de la caja.
En este ejercicio vamos a desarrollar una aplicación en la que el usuario debe *arrastrar* una pelota dentro de una caja que permanece estática. Para ello usaremos dos sprites: un componente "ImageSprite" para la caja (cuya imagen buscaremos en internet), y un componente "Ball" para la pelota. Ambos sprites existirán en un componente "Canvas" de anchura y altura igual a 300 píxeles. Cuando el usuario arrastre la pelota dentro de la caja (evento que detecteremos cuando ambos sprites colisionen), la app mostrará por pantalla una etiqueta que nos informe de que la pelota se ha guardado en la caja, y la pelota se queda inmóvil dentro

de la caja. Añade un botón que te permita quitar la etiqueta y volver a sacar la pelota de la caja, colocándola en pantalla en una posición alejada de la caja.



4. VARIABLES, TOMA DE DECISIONES, Y PROCEDIMIENTOS.

En este capítulo vamos a experimentar con tres poderosas herramientas de programación: En primer lugar entenderemos la forma en la que los programas pueden recordar y reutilizar información usando variables. Después estudiaremos la manera en la que los programas pueden formular preguntas y tomar decisiones mediante los operadores de comparación y los bloques de control. Finalmente veremos cómo podemos dividir un programa en trozos de código reutilizables llamados procedimientos.

4.1. APPS QUE RECUERDAN DATOS

Al igual que las personas, las apps necesitan recordar ciertos datos para poder recuperarlos más adelante. Es por ello que, al igual que las personas, las apps también precisan de una memoria. Pero su funcionamiento es mucho menos complejo y misterioso que el de nuestro cerebro. En esta sección aprenderemos a configurar la memoria de una app, a almacenar información en ella, y a recuperar la información guardada para usarla posteriormente.

La memoria de una app no es más que un conjunto de posiciones de memoria con un nombre identificativo. Algunas de estas posiciones de memoria se crean cuando en el Diseñador añadimos un componente a nuestra app: estas posiciones de memoria son las **propiedades**. Pero también podemos definir posiciones de memoria que no estén asociadas a ningún componente en particular: tales posiciones de memoria se denominan **variables**.

4.2. PROPIEDADES.

Los usuarios de las apps pueden visualizar en la pantalla de su dispositivo componentes como botones, etiquetas, cajas de texto, etc. Sin embargo, internamente los distintos componentes (visibles y no visibles) de una app están completamente definidos por el conjunto de sus propiedades. Los valores almacenados en las posiciones de memoria asociadas a estas propiedades determinan la forma en la que se muestra y funciona cada componente.

Podemos fijar los valores de las propiedades de un componente directamente desde el Diseñador. Por ejemplo, la figura muestra el panel para establecer las propiedades de un componente "Canvas".

Como vemos, el componente "Canvas" tiene propiedades de varios tipos. Por ejemplo, las propiedades "BackgroundColor" y "PaintColor" son posiciones de memoria que almacenan un color. La propiedad "BackgroundImage" almacena un nombre de archivo (*Wall.png*). La propiedad "Visible" almacena un valor Booleano ("true" o "false"), dependiendo de si la casilla asociada está activada o no. Y las propiedades "Width" y "Height" almacenan un número o una designación especial (por ejemplo, "Fill parent").

Cuando fijamos el valor de una propiedad en el Diseñador, lo que estamos haciendo es especificar su *valor inicial*, esto es, el valor que toma cuando arranca la app. Los valores de las propiedades también pueden cambiarse durante la ejecución de la app mediante bloques.

Y sin embargo, los valores mostrados en el Diseñador no cambian. El Diseñador siempre muestra los valores iniciales de las propiedades. Esto puede resultar desconcertante cuando probamos la app, porque los valores actuales de las propiedades no están directamente visibles.

4.3. VARIABLES.

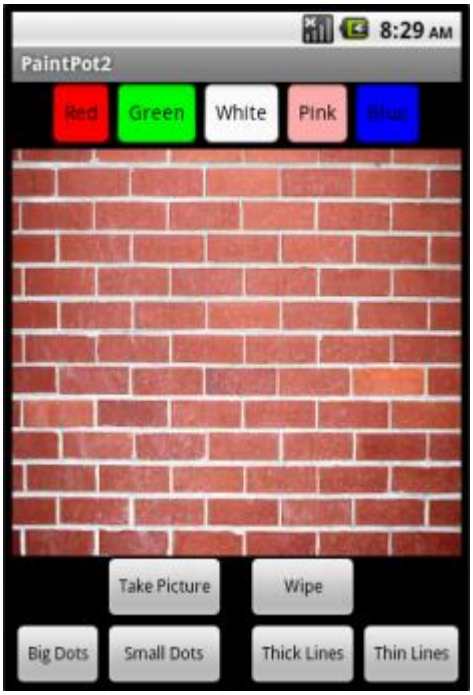
Al igual que las propiedades, las variables son posiciones de memoria identificadas con un nombre, con la diferencia de que no están asociadas a ningún componente en particular. En App Inventor definimos variables cuando la app necesita recordar un dato que no está almacenado en una propiedad. Por ejemplo, en un videojuego podríamos necesitar recordar la puntuación del usuario, o el nivel en el que se encuentra. Si solo queremos mostrar la puntuación en una etiqueta, probablemente no neceitemos una variable, porque podemos guardar ese dato en la propiedad "Text" de esa etiqueta. Pero si vamos a tomar una decisión basada en el valor de la puntuación, deberemos almacenar ese valor en una variable.

Las variables son *temporales*, lo que significa que si reiniciamos la app o apagamos el teléfono, dichas variables y la información que almacenan habrán desaparecido. En otras ocasiones querremos que la app recuerde información incluso después de haber cerrado la app o apagado el teléfono, como una lista de contactos o las puntuaciones más altas en un videojuego. A esto se le llama *persistencia*. Para hacer que los datos se guarden de forma persistente debemos guardar las variables en un dispositivo de almacenamiento o en una base de datos externa para poder volver a cargarlas en la memoria del teléfono la próxima vez que la app vuelva a ejecutarse. Hablaremos más de la persistencia en capítuos posteriores.

CREAR Y RECUPERAR VARIABLES.

El hecho de poder almacenar información en la memoria interna del teléfono no es suficiente. Además, necesitamos algún mecanismo para poder recuperarla. Por esta razón las variables tienen nombres. Puede que ya nos hayamos encontrado con esta idea en álgebra: Cuando escribimos $x = 100$, lo que estamos diciendo es que siempre que escribamos x, nos estaremos refiriendo a su valor asignado, que es 100. Las variables en programación funcionan de forma similar, con la diferencia de que pueden almacenar cualquier tipo de información (y no solo números). Además, las variables pueden tener nombres que posean algún significado (en vez de llamarse simplemente x). Algunos ejemplos serían $Score = 120$, $Sex = "F"$, $Surname = "Jones"$, y $ArrivedAtDestination = False$. Estos ejemplos muestran que las variables pueden almacenar datos de distintos tipos: $Score$ es un número; Sex y $Surname$ son variables de tipo texto (a veces llamadas **cadenas**); y $ArrivedAtDestination$ es una variable **booleana** que solo puede tomar dos valores: true y false.

En resumen, una variable es una especie de celda en la memoria RAM del teléfono en la que podemos almacenar de forma temporal datos que necesitaremos utilizar más adelante durante la ejecución de la app. En ocasiones resulta útil imaginar a las variables como cajas con nombre en la que podemos guardar datos que pueden variar, y cuyos valores podemos recuperar más tarde cuando sean necesarios.



En este capítulo empezaremos a usar variables modificando la app "PaintPot1" para permitir que el usuario pueda cambiar el tamaño de los puntos y el grosor de las líneas dibujadas en pantalla.

4.4. COMENZAR CON LA APP "PAINTPOT2".

En el capítulo previo construimos la app "PaintPot1", en la que el usuario podía dibujar puntos y líneas de distintos colores en la pantalla del móvil. Sin embargo, esta app tenía una limitación: El tamaño de los puntos y el grosor de las líneas dibujadas en pantalla

siempre eran los mismos, y el usuario no podía modificarlos. Por ello, vamos a crear una segunda versión de la app que le permita al usuario aumentar o disminuir el tamaño de los puntos y el grosor de las líneas en relación a sus valores iniciales.

Como esta nueva app será muy parecida a "PaintPot1", vamos a crearnos una copia de "PaintPot1", y a renombrarla como "PaintPot2". A continuación, usaremos esta nueva app "PaintPot2" para aplicar todos los cambios necesarios. Para ello, acudimos al menú "Projects" → "My projects", y abrimos la app "PaintPot1". Una vez abierta, vamos de nuevo a "Projects" y seleccionamos la opción "Save project as…" para crear una copia de esta app. En la ventana emergente, llamamos a esta copia "PaintPot2", y al clicar en el botón OK, App Inventor abrirá el nuevo proyecto "PaintPot2", en el que nos pondremos a trabajar a partir de ahora.

4.5. DISEÑAR LOS COMPONENTES DE "PAINTPOT2".

En la app "PaintPot1" el tamaño de los puntos dibujados en el lienzo "DrawingCanvas" quedó predeterminado por el bloque "call (DrawingCanvas).DrawCircle", ya que el parámetro "radius" se fijó a 5. Por su parte, el grosor de las líneas trazadas también quedó establecido por la propiedad "LineWidth" de "DrawingCanvas", que fijamos a 5.

Como programadores, podemos cambiar el tamaño de los puntos dándole al parámetro "radius" y a la propiedad "LineWidth" unos valores distintos a sus valores iniciales, por ejemplo, 3 y 8, respectivamente. Pero el hecho aquí es que el usuario se verá limitado por los valores que el *programador* haya fijado en el parámetro "radius" y en la propiedad "LineWidth". ¿Y si el *usuario* quisiera cambiar el tamaño de los puntos y/o el grosor de las líneas? Vamos a modificar el programa para permitir que el usuario, y no solo el programador, pueda cambiar estos dos valores.

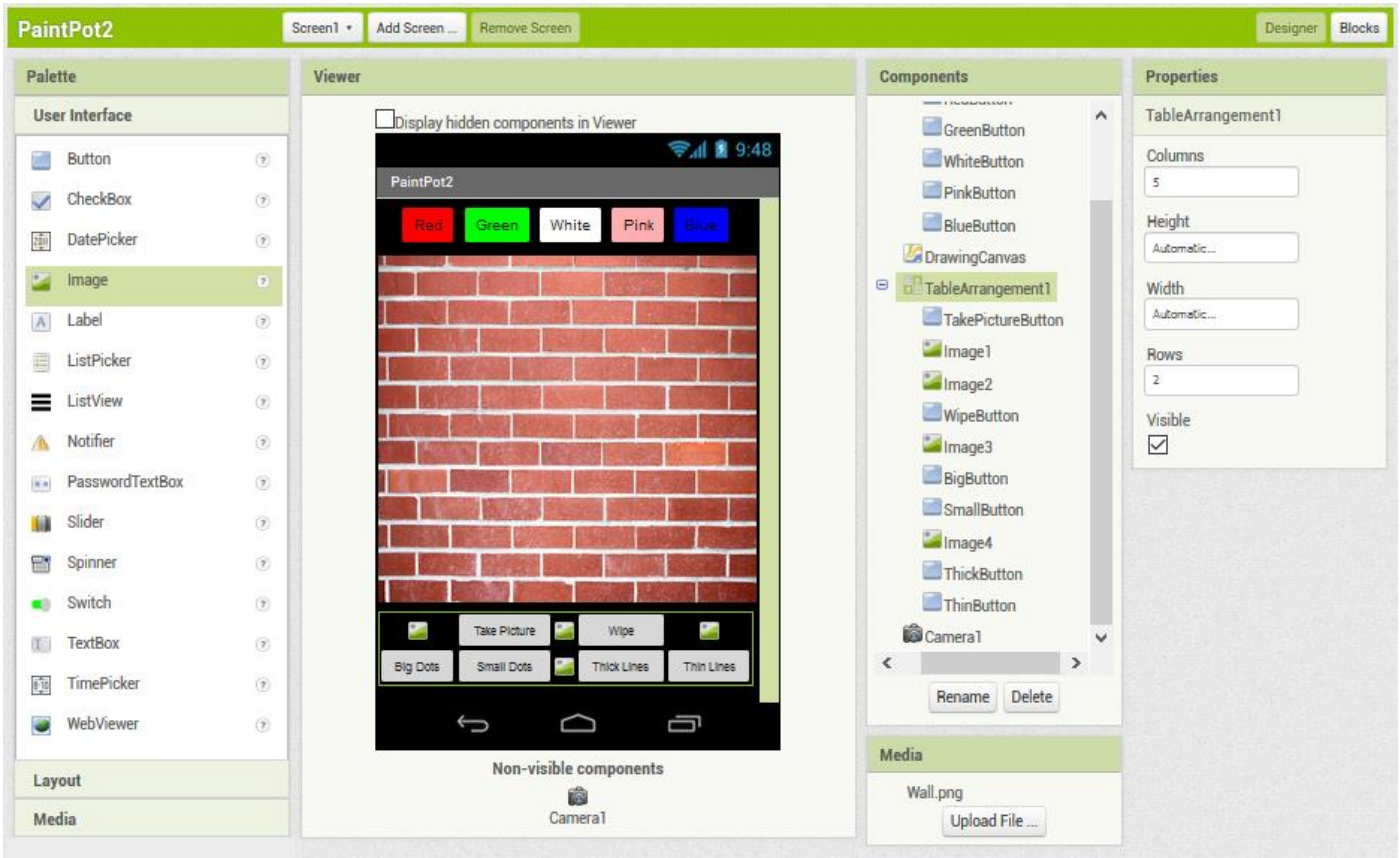
Para ello vamos a añadir cuatro botones más a nuestra app, que distribuiremos en sendos organizadores horizontales. Mediante dos botones llamados "Big Dots" y "Small Dots" cambiaremos el tamaño de los puntos de su valor inicial de 5 a unos nuevos valores de 8 y 2, respectivamente. Con otros dos botones llamados "Thick lines" y "Thin lines" cambiaremos el grosor de las líneas de su valor inicial de 5 a unos nuevos valores que también serán de 8 y 2, respectivamente.

Ello implica que hemos de añadir un total de cuatro botones nuevos, que ubicaremos justo bajo los botones "TakePictureButton" y "WipeButton". Como al final acabaremos con un teclado de seis botones bajo el lienzo "DrawingCanvas", para que este teclado quede más organizado nos conviene insertar todos estos botones en un "TableArrangement", que sustituirá al "HorizontalArrangement" que actualmente alberga a los botones "TakePictureButton" y "WipeButton". Para ello:

1) Agregamos un "TableArrangement" bajo el "HorizontalArrangement" que actualmente contiene a los botones para tomar una foto y limpiar la pantalla. Fijamos sus propiedades "Columns" y "Rows" a 5 y 2, respectivamente.
2) Para poder acceder a todos los componentes que ahora mismo existen en la pantalla, puede que necesitemos activar la propiedad "Scrollable" de "Screen1". Aprovechamos para cambiar la propiedad "Title" de la pantalla a "PaintPot2".
3) Insertamos tres componentes "Image" vacíos que ubicamos en las columnas 1, 3, y 5 de la primera fila de la tabla. (El primer y último componentes "Image" servirán para dejar sendos huecos vacíos al principio y al final de la primera fila de la tabla. El segundo componente "Image" actuará de separador). A continuación, movemos los botones "TakePictureButton" y "WipeButton" a las columnas 2 y 4 de la primera fila de la tabla, respectivamente.
4) Eliminamos el componente "HorizontalArrangement" que contenía a los botones "TakePictureButton" y "WipeButton", puesto que ya no es necesario. (Con esto también borraremos la imagen vacía que separaba a estos dos botones).

5) Creamos 4 nuevos botones, a los que llamaremos "BigButton", "SmallButton", "ThickButton" y "ThinButton". Además, cambiamos sus textos a "Big Dots", "Small Dots", "Thick lines", y "Thin lines", respectivamente.
6) Insertamos un componente "Image" vacío en la columna 3 de la segunda fila de la tabla, entre los botones "SmallButton" y "ThickButton". Este componente servirá para separar los botones de los puntos de los botones de las líneas.
7) Para que este nuevo teclado sea totalmente visible en la pantalla de la app, debemos reducir la altura del lienzo "DrawingCanvas" a 290 píxeles.
8) Cambiamos el tamaño del texto de los seis botones dentro del "TableArrangement" de 14 a 12, para que sean perfectamente visibles en pantalla.
9) Cambiamos la anchura de los componentes "Image" que separan "TakePictureButton" y "WipeButton" por un lado, y "SmallButton" y "ThickButton" por otro lado, a 15 píxeles. Con ello implementaremos las separaciones que vemos en la captura de pantalla de la app.

Finalmente, el aspecto de nuestra interfaz de usuario debería parecerse a la de la figura:



Probamos el aspecto que tiene la app en nuestro dispositivo. Conectamos nuestro teléfono para hacer pruebas en vivo, y observamos si los valores fijados para las distintas propiedades son los adecuados para mostrar correctamente la interfaz en la pantalla de nuestro dispositivo. De no ser así, tal vez tengamos que fijar otros valores mediante el método de prueba y error.

4.6. AÑADIR COMPORTAMIENTOS A LOS COMPONENTES DE "PAINTPOT2".

Vamos a definir los comportamientos de los nuevos botones añadidos a la app "PaintPot2", tal y como hemos indicado en la sección previa.

Como el parámetro "radius" y el atributo "LineWidth" podrán tomar distintos valores, y la app necesita saber qué valor tienen en cada momento. Cuando el usuario especifique los valores que les quiere dar a

"radius" y a "LineWidth", la app deberá almacenarlos (o recordarlos) para después poder usarlos. Siempre que necesitemos que nuestra app recuerde un valor que no sea una propiedad, deberemos definir una variable. En este caso, el único valor que debemos almacenar en una variable es el del parámetro "radius" (porque el valor del grosor de las líneas ya se almacena en la propiedad "LineWidth").

DEFINIR UNA VARIABLE.

Comenzamos definiendo una variable llamada "dotSize":

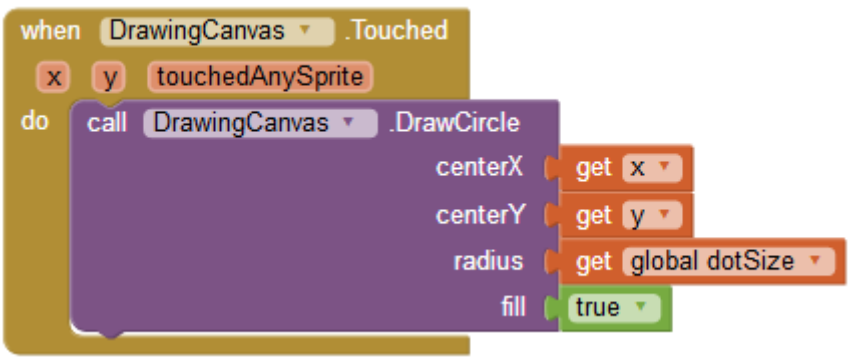
1) En el Editor de Bloques acudimos a la bandeja "Variables", y arrastramos el bloque "initialize global (name) to". Dentro del bloque, cambiamos el texto "name" por "dotSize".
2) Notar que el bloque "initialize global (dotSize) to" tiene a su derecha un conector abierto. Ahí es donde especificamos el valor inicial de la variable, esto es, el valor que toma la variable por defecto al arrancar la aplicación. En nuestro caso, vamos a inicializar la variable a un valor de 5. Para ello, creamos un bloque numérico con un 5. (Recordemos que pinchando en una zona en blanco del área de trabajo y escribiendo un "5" podemos crear automáticamente un bloque numérico con ese valor). A continuación, conectamos el bloque numérico al puerto abierto del bloque "initialize global (dotSize) to".

REFERENCIAR UNA VARIABLE EN UN BLOQUE.

Ahora queremos cambiar el valor del parámetro "radius" del bloque "call (DrawingCanvas).DrawCircle" dentro del manejador de evento "when (DrawingCanvas).Touched" para que use el valor de la variable "dotSize" en vez de usar siempre un valor fijo de 5. (Como veremos después, los botones "Big Dots" y "Small Dots" podrán cambiar el valor de esta variable, y por lo tanto, el tamaño de los puntos dibujados).

1) Sacamos un bloque "get ()" de "initialize global (dotSize) to". Deberíamos obtener un bloque "get (global dotSize)" que representa el valor actual de la variable "dotSize".

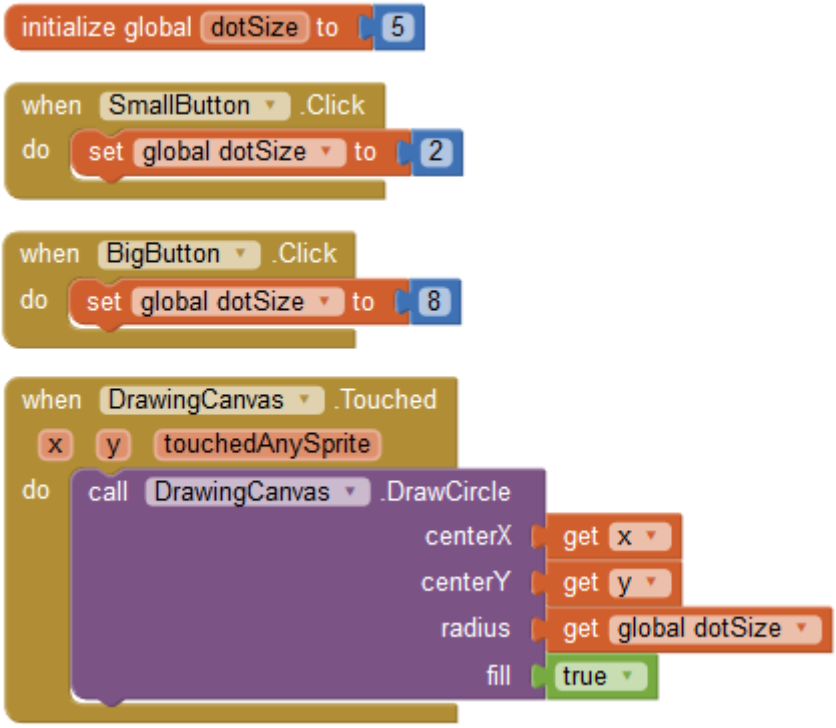
2) Acudimos al bloque "call (DrawingCanvas).DrawCircle", desconectamos el bloque numérico con el valor 5 del conector "radius", y lo enviamos a la papelera (lo borramos). Entonces lo reemplazamos por el bloque "get (global dotSize)". De esta forma, cuando el usuario toque el "DrawingCanvas", la app obtendrá el radio del punto a dibujar del valor almacenado en la variable "dotSize".



CAMBIAR EL VALOR DE UNA VARIABLE.

Ahora mismo, nuestra app puede dibujar círculos cuyo tamaño está basado en el valor de la variable "dotSize", pero todavía necesitamos programar la forma en la que "dotSize" cambia su valor de acuerdo con lo que elija el usuario. (Notar que, de momento, el valor de "dotSize" está fijo a su valor por defecto de 5). Vamos a implementar este comportamiento programando los manejadores de evento "when (SmallButton).Click" y "when (BigButton).Click":

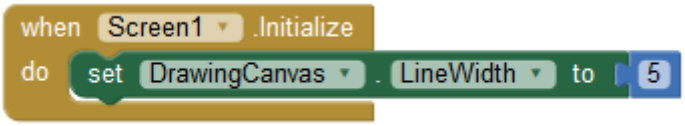
1) Sacamos un manejador de evento "when (SmallButton).Click" de la bandeja de bloques de "SmallButton". A continuación, pasamos el ratón sobre el argumento "dotSize" del bloque "initialize global (dotSize)", sacamos un bloque "set (global dotSize) to", y lo ponemos dentro del bloque "when (SmallButton).Click". Finalmente, creamos un bloque numérico con un 2, y lo conectamos al puerto abierto de "set (global dotSize) to".
2) Hacemos algo similar para el manejador de evento "when (BigButton).Click", pero esta vez fijando el valor de la variable "dotSize" a 8. El programa para determinar el tamaño del punto que se dibuja por pantalla debería quedar finalmente como:



NOTA: El "global" en los bloques asociados a la variable "dotSize" se refiere al hecho de que la variable puede usarse en todos los manejadores de eventos del programa (esto es, globalmente). En App Inventor también se pueden definir variables *locales*, esto es, variables propias y exclusivas de una cierta parte del programa. Hablaremos más sobre **variables globales** y **variables locales** en capítulos posteriores.

INICIALIZAR EL GROSOR DE LAS LÍNEAS.

En la app "PaintPot1" insertamos a la propiedad "LineWidth" de "DrawingCanvas" un valor de 5 directamente en el Diseñador. Sin embargo, este tipo de configuraciones iniciales también pueden hacerse con bloques de código.



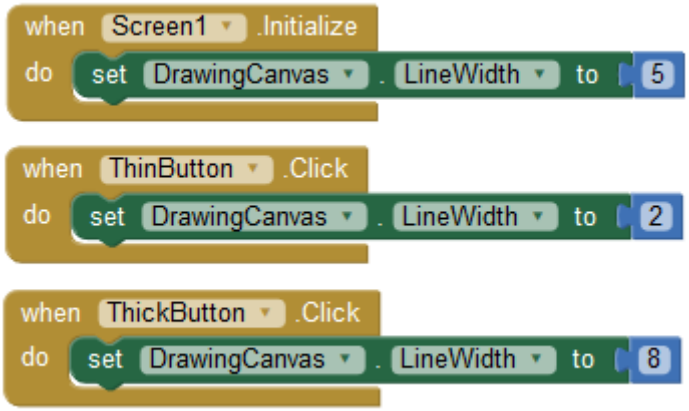
Aquí queremos fijar un valor inicial de 5 a la propiedad "LineWidth" de "DrawingCanvas" nada más arrancar la app. Tal vez recordemos que esto podemos hacerlo programando el manejador "when (Screen1). Initialize" como muestra la figura previa.

CAMBIAR EL GROSOR DE LAS LÍNEAS.

Tal y como está ahora, la app puede dibujar líneas cuyo grosor está fijado en el valor actual de la propiedad "LineWidth", que hemos inicializado a 5. Ahora vamos a programar el comportamiento de los botones "ThinButton" y "ThickButton" para cambiar el valor de esta propiedad a 2 y 8, respectivamente. (Como ya mencionamos antes, aquí no necesitaremos definir una variable para recordar el valor actual del grosor de las líneas, ya que dicho valor ya está almacenado en la propiedad "LineWidth". Las variables sirven para recordar valores que *no* sean propiedades).

Vamos a implementar este comportamiento programando los manejadores de evento "when (ThinButton).Click" y "when (ThickButton).Click":

1) Sacamos un manejador de evento "when (ThinButton).Click" de la bandeja de bloques de "ThinButton". A continuación, acudimos a la bandeja de "DrawingCanvas" y sacamos un bloque "set (DrawingCanvas).LineWidth to", y lo conectamos dentro del manejador "when (ThinButton).Click". Finalmente, creamos un bloque numérico con un 2, y lo conectamos al puerto abierto de "set (DrawingCanvas).LineWidth to".
2) Hacemos algo similar para el manejador de evento "when (ThickButton).Click", pero esta vez fijando el valor de la propiedad "LineWidth" a 8. El programa para fijar el grosor de las líneas que se dibujan por pantalla debería quedar finalmente como:



En este caso, no necesitamos modificar de ninguna forma el manejador "when (DrawingCanvas).Dragged", porque este bloque no necesita recibir como parámetro el grosor de las líneas dibujadas.

Con esto hemos terminado la app "PaintPot2". Vamos a probar su funcionamiento: Conectamos el dispositivo (o el emulador) para hacer pruebas en vivo. Al arrancar la app, los puntos y las líneas dibujadas en pantalla deberían tener un tamaño intermedio. Al presionar en los botones "BigDots" y "ThickLines", deberíamos pasar a dibujar líneas y puntos más grandes y gruesos que al principio. Al presionar en los botones "SmallDots" y "ThinLines" los puntos y líneas dibujadas por pantalla deberían ser más pequeños y finos que los dibujados al principio.

4.7. TOMA DE DECISIONES.

En la sección previa hemos aprendido a crear variables, almacenar en ellas información, y recuperar esa información más tarde accediendo al valor almacenado en dichas variables. Las variables también son especialmente útiles cuando las usamos en estructuras de toma de decisiones en nuestras apps. La toma de

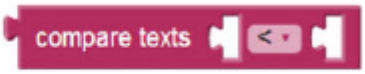
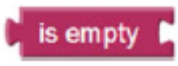
decisiones en un programa suele implicar dos pasos: (1) El programa se hace una pregunta, y (2) El programa toma una decisión basada en la respuesta.

Las preguntas que se hacen los programas para poder tomar una decisión son preguntas del tipo "¿El valor de esta variable es mayor que 5?", "¿Este texto comienza con la letra *A*?", "¿Ha realizado el usuario la misma acción 10 veces?" o "¿Ha colisionado la nave espacial con un asteroide?". Todas estas preguntas hacen comparaciones que comprueban si dos cosas son iguales o distintas.

OPERADORES DE COMPARACIÓN.

Los ordenadores realizan comparaciones usando **operadores de comparación**. Las siguientes tablas muestran los bloques para realizar las operaciones de comparación más habituales; todos estos bloques devuelven "true" o "false" dependiendo del resultado de la comparación. La primera tabla muestra los operadores de comparación que aplican tanto a números como a textos, la segunda tabla lista los operadores que solo funcionan con números, y la tercera tabla incluye los operadores que solo comparan textos.

Number comparison operators		
Comparison operator	**Block**	**What does it do?**
Greater than		Answers True if the number in the left socket is larger than the number in the right socket
Less than		Answers True if the number in the left socket is smaller than the number in the right socket
Greater than or equal to		Answers True if the number in the left socket is larger than *or* the same as the number in the right socket
Less than or equal to		Answers True if the number in the left socket is smaller than *or* the same as the number in the right socket
Is a number?		Answers True if the item in the socket is a number

Text comparison operators		
Comparison operator	**Block**	**What does it do?**
Text less than		Answers True if text 1 is alphabetically smaller than text 2 (meaning it's earlier in the alphabet). For example "Apple" < "Bumblebee" is True.
Text greater than		Answers True if text 1 is alphabetically greater than text 2. For example, "Inventor" > "App" is True.
Text contains		Answers True if the text in the top socket contains the text in the bottom socket anywhere in it. For example, [text "AppInventor" contains piece "Inventor"] is True.
Is text empty?		Answers True if the text is empty—for example, if a user hasn't entered any text in a notifier box.

Text and number comparison operators		
Comparison operator	**Block**	**What does it do?**
Equals		Answers True if the two numbers or pieces of text plugged into the sockets are the same
Does not equal		The opposite of equals—answers True if two numbers or pieces of text aren't the same

Estos operadores de comparación son la base de todas las preguntas que requieren los procesos de toma de decisiones. Por ejemplo, si queremos comprobar si la edad del usuario (dato alojado en la variable "UserAge") es mayor o igual que 18 años, podemos usar los bloques mostrados en la figura:

Con estos bloques hemos construido una pregunta que devolverá "true" o "false" dependiendo del valor de la variable "UserAge". Y ahora, ¿qué hacemos con esta información? ¿A qué otro bloque podemos conectar esta pregunta para tomar una decisión?

BLOQUES CONDICIONALES.

Los humanos tomamos decisiones usando sentencias condicionales como las siguientes:

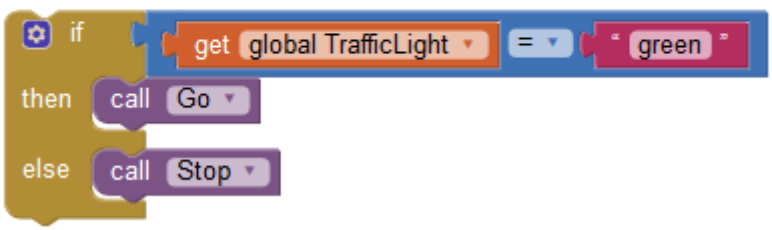
- *Si* el precio de la camisa es menor de 30€, *entonces* me la compro.
- *Si* la crítica de la película es buena, *entonces* voy al cine a verla.
- *Si* suena la alarma, *entonces* llamo a la policía inmediatamente.

En ocasiones las decisiones tienen varios resultados posibles:

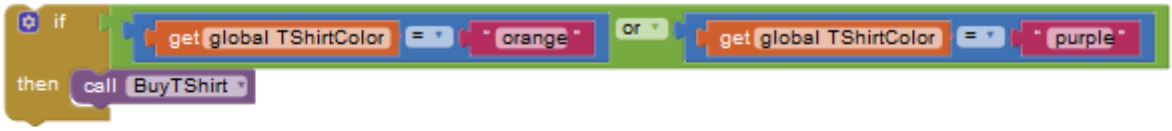
- *Si* la luz está verde, *entonces* salgo; *si no*, me paro.
- *Si* la pasta ya está lista, *entonces* la saco de la cacerola; *si no*, la dejo cociéndose un minuto más.

Casi todos los lenguajes de programación tienen su propia versión de la sentencia "Si … entonces …" (y de la sentencia "Si … entonces … Si no …"). App Inventor no es una excepción e incluye, dentro de la bandeja "Control", el bloque "if - then", al que nosotros llamaremos simplemente bloque "if".

He aquí un par de programas que muestran como quedarían estos bloques "if" aplicados a las sentencias que usamos previamente como ejemplo:

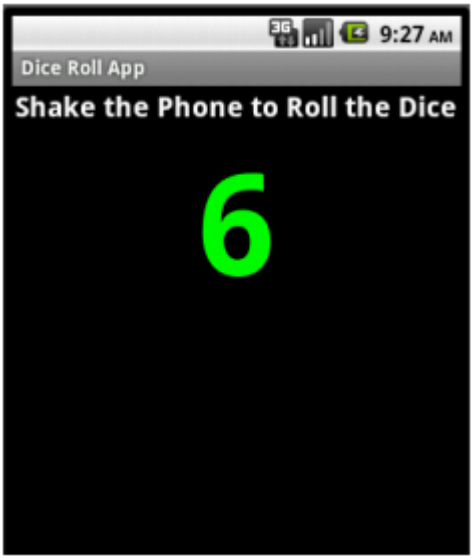


También podemos combinar decisiones usando los operadores lógicos "and" y "or", como muestran los siguientes ejemplos:



A continuación, vamos a poner en práctica todos estos conceptos construyendo una app que simula un dado en nuestro móvil. Además de servirnos para poner en práctica la toma de decisiones, esta app nos ayudará a entender la forma en la que App Inventor permite obtener números aleatorios.

4.8. COMENZAR CON LA APP "DICEROLL1".



Hay muchas ocasiones en las que querremos que una app se comporte de forma aleatoria. Por ejemplo, en un videojuego podríamos necesitar que un personaje se mueva de manera impredecible. Esta app es más sencilla que eso: Vamos a simular un dado en nuestro teléfono móvil, de forma que cuando el usuario agite el dispositivo, la pantalla muestre un número aleatorio entre 1 y 6. Después crearemos una segunda versión de esta app en la que, en vez de mostrar un número aleatorio, muestre una de las seis caras de un dado. Para empezar, nos conectamos a App Inventor y comenzamos un nuevo proyecto llamado "DiceRoll1". A continuación, conectamos el teléfono o el emulador para hacer pruebas en vivo.

4.9. DISEÑAR LOS COMPONENTES DE "DICEROLL1".

La tabla lista todos los componentes necesarios para construir la app "DiceRoll1", así como los valores que hemos de fijar para sus propiedades.

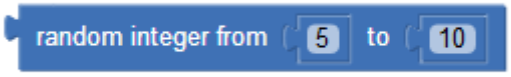
Dice Roll			
Screen1 properties	`AlignHorizontal`: Center `Title`: Dice Roll App `Icon`: dice_6.png (downloaded from our website) `ScreenOrientation`: Portrait `BackgroundColor`: Black		
Components	**What do I rename it?**	**What does it do?**	**What properties do I set?**
`AccelerometerSensor` Palette group: Sensors	`Accelerometer-Sensor1`	Detects if the user has shaken the phone	Enabled (selected)
`Label`	`InstructionLabel`	Tells the user what to do	`FontBold`: selected `Text`: "Shake the Phone to Roll the Dice" `TextColor`: White `TextAlignment`: Center
`Label`	`DiceLabel`	Displays the number that has been rolled	`FontBold`: selected `Text`: "6" `FontSize`: 100 `TextColor`: Green

Añade todos estos componentes al Diseñador de App Inventor hasta conseguir una interfaz de usuario como la mostrada en la figura previa.

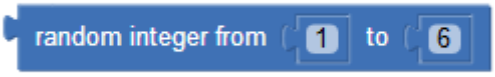
4.10. AÑADIR COMPORTAMIENTOS A LOS COMPONENTES DE "DICEROLL1".

NÚMEROS ALEATORIOS.

Esta app es muy sencilla de programar, porque App Inventor incluye dentro de la bandeja "Math" el bloque "random integer from () to ()", que permite generar un entero aleatorio entre los dos enteros que le pasamos como argumentos. Por ejemplo, el siguiente bloque serviría para generar un entero aleatorio entre 5 y 10, esto es, para elegir aleatoriamente un número de entre 5, 6, 7, 8, 9, y 10.



En nuestro caso, queremos simular el lanzamiento de un dado, experimento que da como resultado un entero aleatorio entre 1 y 6. Así pues, el bloque que necesitamos es:



MOSTRAR POR PANTALLA UN ENTERO ALEATORIO AL AGITAR EL DISPOSITIVO.

Para mostrar por pantalla el número aleatorio utilizaremos la etiqueta "DiceLabel". Como debemos mostrar este número aleatorio al agitar el dispositivo, debemos programar el manejador de eventos "when (AccelerometerSensor1).Shaking" como muestra la figura:

Y con esto ya hemos terminado. Ahora vamos a probar la app: Cada vez que agitemos nuestro dispositivo, la app debería ajustar la propiedad "Text" de "DiceLabel" a un número aleatorio entre 1 y 6, y mostrarlo por pantalla.

> NOTA: Si alguna vez necesitamos obtener un número aleatorio decimal, disponemos de un bloque análogo a "random integer" llamado "random fraction". Notar que este bloque no necesita dos valores límite entre los que obtener un fraccionario aleatorio; siempre genera un número decimal aleatorio entre 0 y 1.

4.11. COMENZAR CON LA APP "DICEROLL2".

Vamos a crear una segunda versión de la app "DiceRoll1", a la que llamaremos "DiceRoll2". En esta nueva versión no mostraremos el número aleatorio obtenido mediante una simple etiqueta, sino que mostraremos aleatoriamente una de las seis caras de un dado.

En primer lugar, abrimos el proyecto "DiceRoll1", y en el menú superior seleccionamos la opción "Projects" → "Save project as…". A continuación, en la ventana emergente, guardamos esta copia del proyecto como "DiceRoll2". Con ello tenemos ya dos copias del proyecto previo, llamadas "DiceRoll1" y "DiceRoll2". Para programar la segunda versión de la app, vamos a modificar la copia "DiceRoll2".

Para esta segunda versión necesitaremos seis archivos de imagen con las seis caras de un dado. Estas imágenes están disponibles en los archivos de los alumnos, con los nombres *dice_1.png*, *dice_2.png*, …, *dice_6.png*. Cargamos uno a uno estos seis archivos en nuestro proyecto usando el botón "Upload file" de la sección "Media" del Diseñador.

4.12. DISEÑAR LOS COMPONENTES DE "DICEROLL2".

Esta app es muy similar a la anterior. La única diferencia es que, en vez de mostrar un número aleatorio mediante la etiqueta "DiceLabel", vamos a elegir aleatoriamente y a mostrar una de las seis imágenes de las caras de un dado.

Para completar el nuevo diseño, basta con borrar la etiqueta "DiceLabel" y añadir un componente "Image" llamado "DiceImage". Además, vamos a fijar su propiedad "Picture" a *dice_1.png*, para que al arrancar la aplicación muestre por defecto la imagen de un dado con un solo punto.

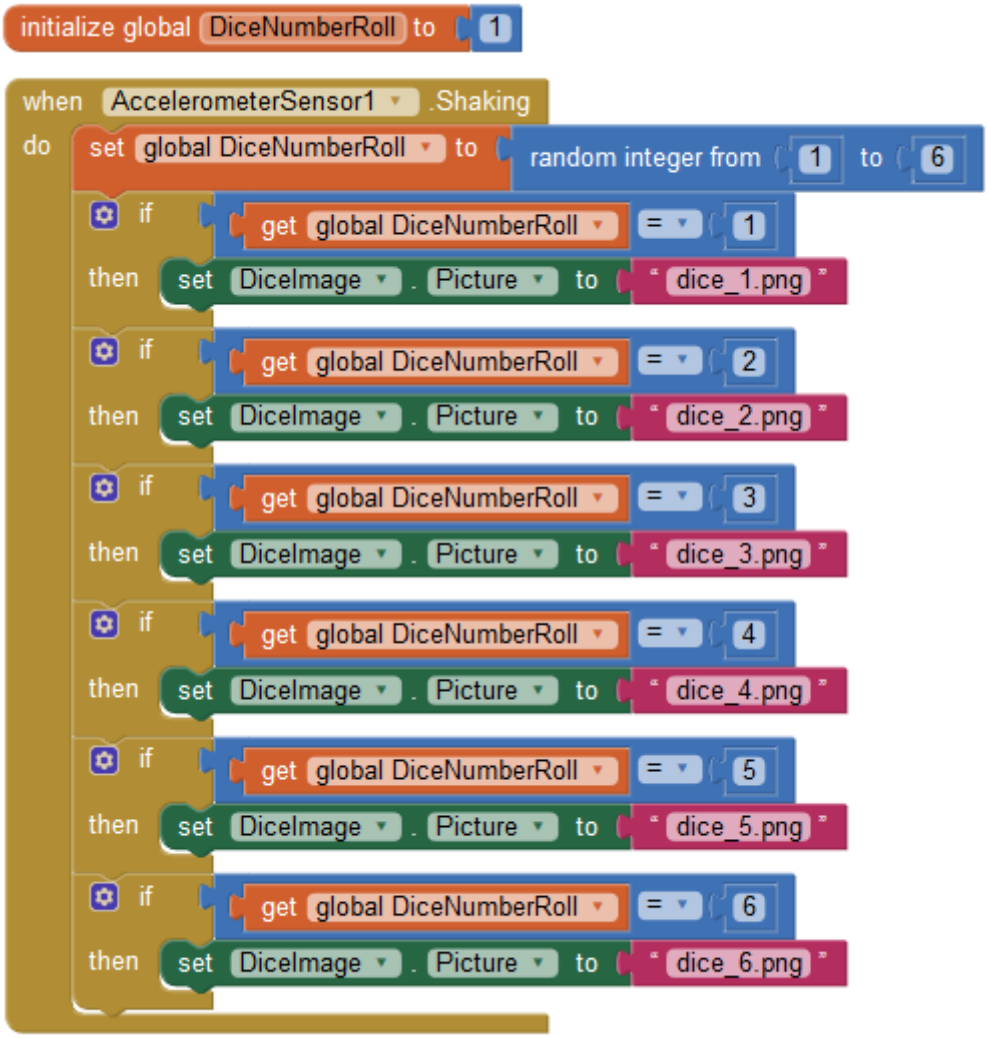
Con esto hemos terminado el diseño de la interfaz de usuario. Ahora hay (al menos) dos formas de programar el comportamiento de los bloques. Una es fácil de ver, pero un poco larga. La otra no es tan obvia, pero es mucho más eficiente. Este tipo de situaciones son muy habituales en programación, y siempre que podamos o sepamos, debemos optar por la solución más eficiente. Ello nos permitirá ahorrar tiempo, hará que el código sea más fácil de mantener, y normalmente facilitará que la aplicación se ejecute más rápidamente. Veamos cuáles son estas dos opciones.

4.13. AÑADIR COMPORTAMIENTOS A LOS COMPONENTES DE "DICEROLL2".

OPCIÓN 1: USAR MÚLIPLES BLOQUES "IF".

En primer lugar, definimos una variable llamada "DiceNumberRoll", y la inicializamos a 1. Al agitar el dispositivo, fijamos el valor de "DiceNumberRoll" a un número aleatorio entre 1 y 6. Seguidamente, y mediante múltiples bloques "if", comparamos el valor almacenado por "DiceNumberRoll" con $1, 2, 3, 4, 5,$ y 6, y en cada caso, fijamos la propiedad "Picture" del componente "DiceImage" al archivo de imagen adecuado (ver figura).

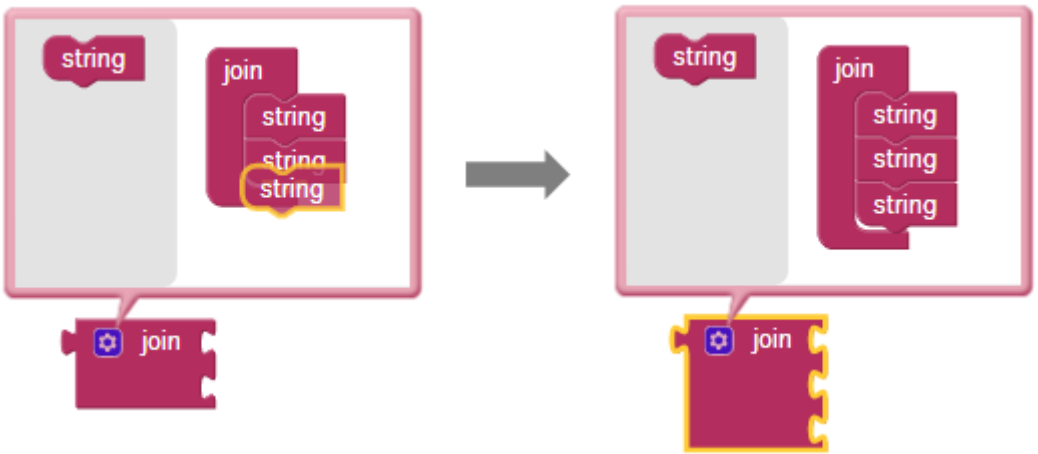
Este método es sencillo, pero requiere que la app procese las seis sentencias "if", incluso cuando la primera ya es cierta. Notar que todas las sentencias "if" son muy similares: Lo único que cambia cada vez es la prueba de comparación del valor de "DiceNumberRoll" y el nombre del archivo de imagen que debe mostrarse. Además, notar que hay dos cosas que coinciden: Si el valor almacenando por "DiceNumberRoll" es 2, la imagen que debemos mostrar es *dice_2.png*. Por consiguiente, tendría mucho más sentido programar algo parecido a lo siguiente: "Escoge un número aleatorio entre 1 y 6, y a continuación carga la imagen con ese número en su nombre de archivo". Eso es precisamente lo que haremos en la versión más eficiente de esta app.



OPCIÓN 2: CONCATENAR EL NOMBRE DEL ARCHIVO DE IMAGEN.

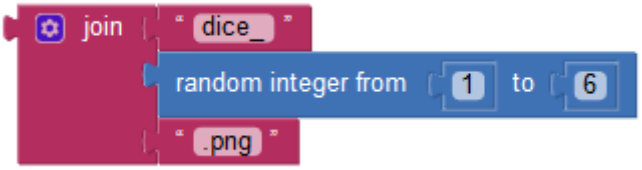
Concatenar significa unir en una sola cadena de texto dos o más elementos (por ejemplo, dos textos, un texto y el valor de una variable, un texto y el valor arrojado por una función, etc.). En nuestro caso, cada vez que agitemos el teléfono móvil, querremos que la app fije la propiedad "Picture" del componente "DiceImage" al nombre de archivo que combina el texto "dice_", un número aleatorio entre 1 y 6, y finalmente, el texto ".png". Para combinar estos tres elementos en una sola cadena de texto necesitamos usar el bloque "join":

1) Sacamos un bloque "join" de la bandeja "Text", y lo soltamos en el área de trabajo. Cada ranura abierta a la derecha del bloque "join" debe recibir uno de los elementos (textos, variables, funciones) que queremos combinar para formar la cadena de texto final.

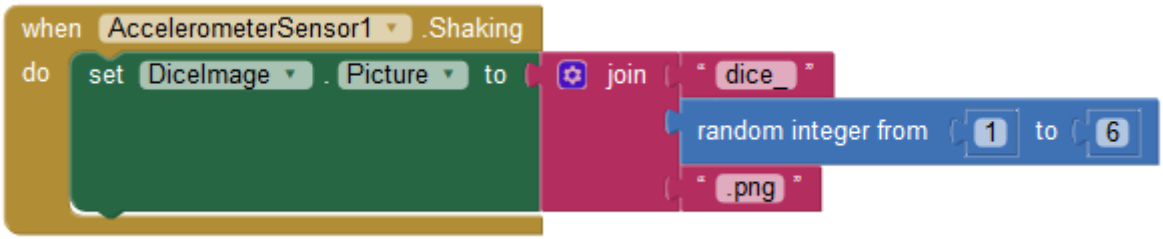


El cuadradito azul con el icono del engranaje significa que este bloque es *mutable*, esto es, que podemos cambiar el número de puertos abiertos que tiene. Por defecto, el bloque "join" aparece con dos ranuras abiertas, lo que nos permitiría concatenar dos elementos. En nuestro caso necesitamos combinar tres elementos, por lo que debemos añadir una ranura extra a la derecha del bloque. Para ello, pinchamos en el botón azul del bloque "join", y en la ventana emergente arrastramos los bloques "string" extra que necesitemos a la derecha del "join", ver figura.

2) De la bandeja "Text", extraemos dos bloques de texto vacío, y dentro de ellos escribimos las cadenas "dice_" y ".png", y las conectamos a la primera y tervera ranuras del bloque "join".
3) Ahora sacamos de la bandeja "Math" un bloque "random integer", lo configuramos para que seleccione un aleatorio entre 1 y 6, y conectamos este bloque en la segunda ranura abierta del bloque "join". La figura muestra cómo debería quedar el conjunto de bloques:



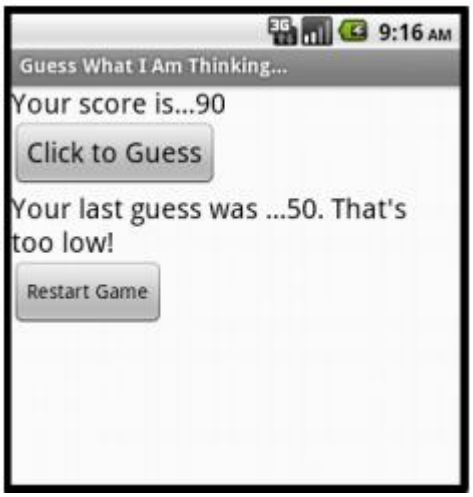
4) Ahora conectamos este bloque compuesto a la derecha del bloque "set (DiceImage).Picture to", y a continuación, metemos el bloque resultante dentro del manejador de evento "when (AccelerometerSensor1).Shaking". El programa debería quedar como en la figura:



Por supuesto, esta técnica no se limita a mostrar imágenes de dados, y tampoco tienen por qué ser solamente seis. Siempre que los nombres de archivo de las imágenes tengan todos el mismo formato y contengan un único número secuencial, este método nos permitirá mostrar cientos de imágenes aleatorias. En estos casos, veremos clara la ventaja de usar este método en lugar de usar cientos de bloques "if".

La app del dado ha sido bastante sencilla. Ahora vamos a construir una app mucho más compleja que cualquiera que las que hayamos realizado hasta ahora. Se trata de un ejercicio que nos permitirá combinar nuestros conocimientos de variables, números aleatorios, y estructuras de toma de decisión. Además, al programarla podremos presentar algunas ideas nuevas de gran importancia, como el uso de procedimientos, y la técnica de validación de datos. También aprenderemos a monitorizar los valores de las variables mientras la app se está ejecutando, para asegurarnos de que el programa funciona como esperábamos.

4.14. COMENZAR CON LA APP "GUESSINGGAME".



En esta sección construiremos un juego en el que la app piensa un número aleatorio, y el jugador tiene que adivinarlo. Después de que el jugador haya hecho su apuesta, la app le dice si el número secreto es mayor o menor que el número apostado. Cada apuesta errónea reduce la puntuación del jugador (que inicialmente es de 100) en 10 puntos, y si la puntuación cae a cero se acaba la partida. Para hacer las cosas más interesantes programaremos tres niveles de dificultad: (1) El nivel fácil obtiene un número aleatorio entre 1 y 100, (2) el nivel medio entre 1 y 200, y (3) el nivel avanzado entre 1 y 300.

Empezamos creando un nuevo proyecto, al que llamaremos "GuessingGame". Cambiamos el título de la pantalla a "Guess the number I am thinking of...". Además, conectamos el dispositivo o el emulador para hacer pruebas en vivo.

4.15. DISEÑAR LOS COMPONENTES DE "GUESSINGGAME".

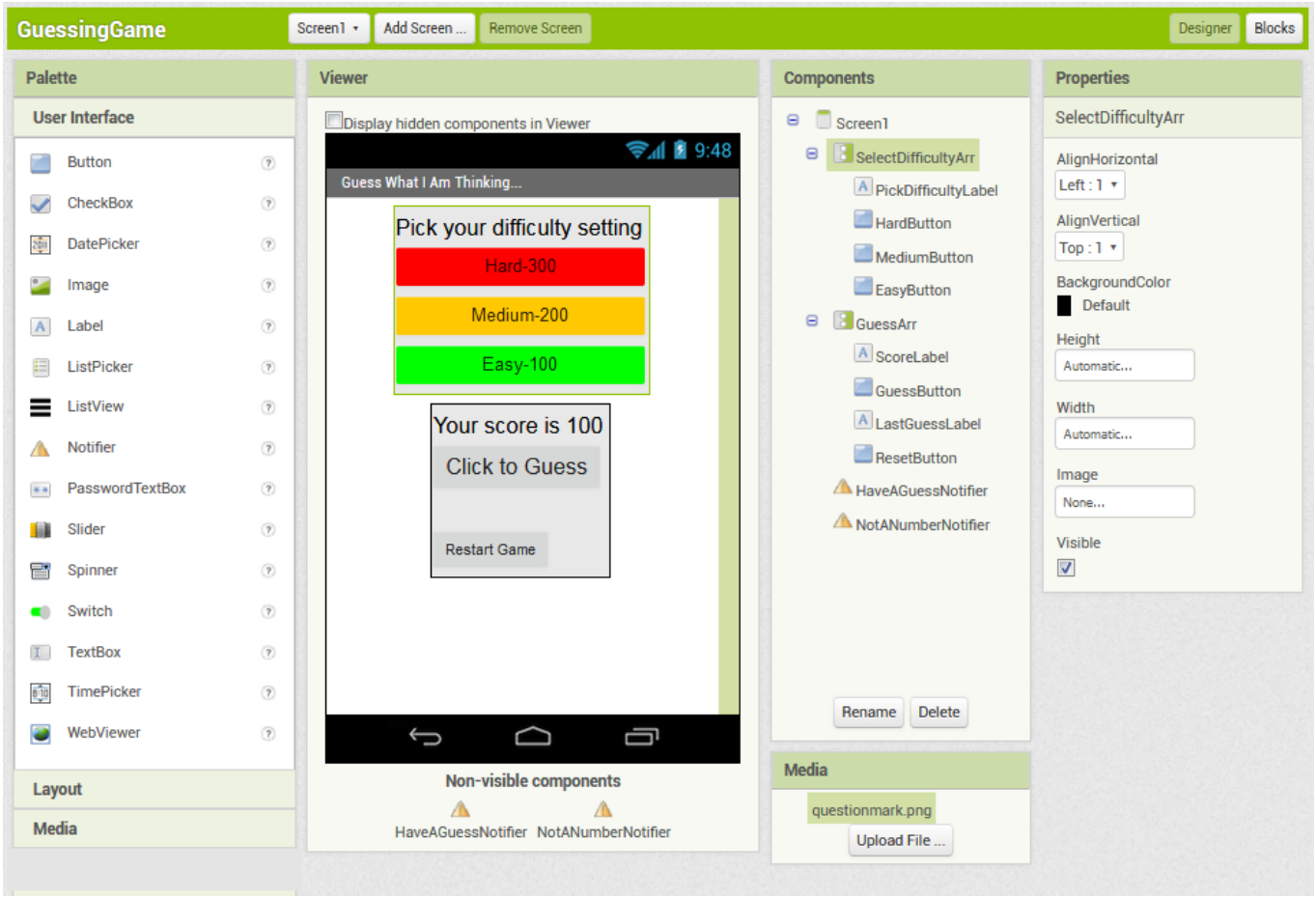
La tabla lista todos los componentes a añadir para la app, y los valores de sus propiedades.

Guess What I Am Thinking			
Screen1 properties	`AlignHorizontal`: Center `Scrollable`: Yes (selected) `Icon`= questionmark.png (downloaded from our website) `BackgroundColor`: Your choice `ScreenOrientation`: Portrait `Title`: Guess What I Am Thinking...		
Components	**What do I rename it?**	**What does it do?**	**What properties do I set?**
`Vertical-Arrangement` Palette group: Layout	`SelectDifficultyArr` (We've shortened `Arrangement` to `Arr` to save typing and to make you talk like a pirate.)	Contains the difficulty label and buttons	`Width` and `Height`: Automatic `Visible`: Showing
`Label`	`PickDifficultyLabel`	Asks the player to pick a difficulty setting	`FontSize`: 20 `Text`: "Pick your difficulty setting"

3 `Buttons`	`HardButton`
	`MediumButton`
	`EasyButton`
		The buttons the user clicks to choose difficulty	`BackgroundColor`
			`Hard`: Red, `Medium`: Orange, `Easy`: Green
			`Text`: "Hard - 300", "Medium - 200", "Easy - 100"
			`FontSize`: 16
			`TextAlignment`: center
			`Width`: Fill Parent
			`Height`: Automatic

`Vertical-Arrangement`	`GuessArr`	Contains `ScoreLabel`, `GuessButton`, `LastGuessLabel`, and `ResetButton`	`Width` and `Height`: Automatic `Visible`: hidden
`Label`	`ScoreLabel`	Displays the score at the beginning of the game and after every guess	`FontSize`: 20 `Text`: "Your score is 100"
`Button`	`GuessButton`	Button the player clicks to make a guess	`Text`: "Click to Guess" `FontSize`: 20 `TextAlignment`: Center
`Label`	`LastGuessLabel`	Displays the last guess and whether it's higher, lower, or correct.	`FontSize`: 20 `Text`: None (leave it blank)
`Button`	`ResetButton`	Resets the game	`Text`: "Restart Game" `FontSize`: 14 `TextAlignment`: Center
`Notifier` Palette group: User Interface	`HaveAGuessNotifier`	A pop-up notifier where the player inputs their guess	None
`Notifier`	`NotANumberNotifier`	Asks the player to try again if they type letters instead of numbers	None

La figura muestra cómo debería quedar la interfaz de usuario de la app al añadir y configurar adecuadamente estos componentes. En particular, es importante asegurarnos de que los dos "VerticalArrangements" (llamados "SelectDifficutyArr" y "GuessArr") contienen los componentes adecuados.



Para esta app queremos que, nada más empezar el juego, sólo esté visible el organizador de selección de dificultad, componente "SelectDifficultyArr". Una vez elegida la dificultad, queremos que solo esté visible el organizador donde el jugador hace sus apuestas, componente "GuessArr". Para hacer que una app solo muestre ciertas cosas en ciertos momentos, podemos modificar la propiedad "Visible" de los componentes involucrados. Como queremos que el juego comience mostrando solamente el organizador "SelectDifficultyArr", en el Diseñador desactivamos la propiedad "Visible" del organizador "GuessArr".

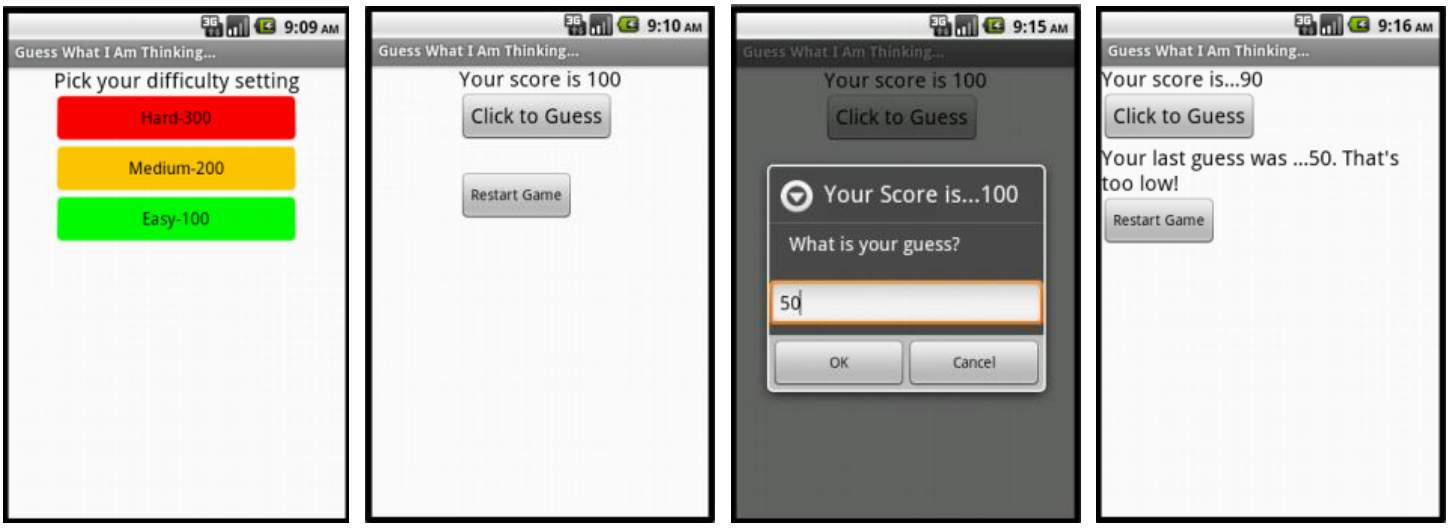
4.16. AÑADIR COMPORTAMIENTOS A LOS COMPONENTES DE "GUESSINGGAME".

Las distintas figuras muestran cómo se desarrolla el juego: La app comienza con una pantalla que solo muestra el organizador "SelectDifficultyArr" (ver figura 1). Cuando el jugador selecciona un nivel de dificultad, escondemos este organizador y mostramos el organizador "GuessArr", donde el jugador hace sus apuestas. Para hacer una apuesta el jugador debe pulsar el botón "Click to Guess", que hace aparecer un notificador donde puede insertar la apuesta, ver figuras 2 y 3. Entonces, la app compara esta apuesta con el número secreto e informa al jugador. Si el jugador presiona el botón de Reset, volvemos a la pantalla inicial de selección de dificultad.

La mecánica de esta app puede dividirse en 3 fases:

1) Inicialización: Es la fase en la que inicializamos las variables necesarias y establecemos el nivel de dificultad para obtener el número secreto adecuado.

2) Juego: Es la fase en la que el usuario hace sus apuestas, y éstas se comparan contra el número secreto.
3) Reseteo: Cuando el usuario pulsa el botón de Reset, la app reinicia el juego volviendo a la pantalla de selección de nivel de dificultad.



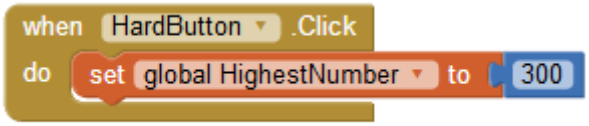
FASE DE INICIALIZACIÓN.

En esta app necesitaremos un total de 4 variables:

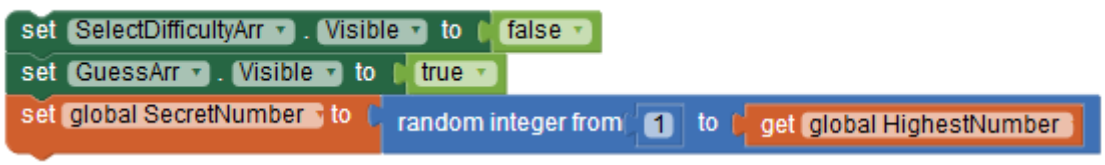
- "HighestNumber": Variable que almacena el número aleatorio más alto que la app podrá sacar, y cuyo valor depende del nivel de dificultad elegido.
- "Score": Variable que llevará la puntuación.
- "PlayerGuess": Variable donde se guarda la última apuesta del usuario.
- "SecretNumber": Variable que almacena el número aleatorio que el jugador debe adivinar.

Inicializamos estas variables en el Editor de Bloques, con unos valores de 100, 100, 0, y 0, respectivamente:

A continuación, cuando el usuario presione en uno de los tres botones de nivel de dificultad (a saber, "EasyButton", "MediumButton", y "HardButton"), debemos fijar el valor de la variable "HigestNumer" a 100, 200, o 300, según el botón pulsado. (Por ejemplo, al pulsar el botón "HardButton", debemos fijar el valor de "HigestNumer" a 300. Los otros dos botones funcionarán de forma similar).



¿Qué más deben hacer estos botones de selección de dificultad? Deben hacer que comience el juego ocultando el organizador "SelectDifficultyArr" (junto con todos sus componentes internos) y mostrando el organizador "GuessArr". Finalmente, deben elegir el número secreto como un número aleatorio entre 1 y "HighestNumber". Todo esto podemos hacerlo con los siguientes bloques de código:



Pero ahora debemos notar un hecho destacable: Los tres botones hacen todas estas tareas de forma idéntica. Detectar este hecho es importante, porque en vez de copiar tres veces el mismo código dentro de los manejadores de los tres botones, podemos crear un **procedimiento** que se encargue de implementar este código común para los tres botones.

Crear un procedimiento.

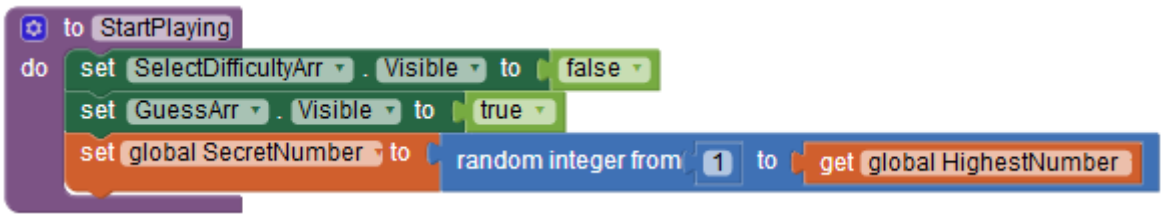
Los procedimientos no son más que conjuntos de bloques de código que realizan una tarea específica, y a los cuales les damos un nombre para poder llamarlos más adelante. En los programas que hemos escrito hasta ahora hemos llamado a procedimientos preconstruidos en App Inventor, a los que hemos denominado funciones. Algunos ejemplos son la función "(Sound1).Play", la función "(Sound1).Vibrate", y la función "(Canvas1).DrawCircle".

App Inventor también nos permite crear nuestros propios procedimientos y llamarlos siempre que los necesitemos. Al igual que las funciones, nuestros procedimientos también aparecerán en una bandeja, y podremos usarlos exactamente igual que ellas. Sin embargo, también sería posible escribir todos los programas de estos apuntes sin crear ni un solo procedimiento. Entonces, ¿cuál es su utilidad? Las ventajas derivadas del uso de procedimientos son múltiples, y entre ellas cabe destacar las siguientes:

- Ahorro de tiempo: Una vez que un programador ha definido en detalle el comportamiento de un procedimiento, pueden llamar a ese procedimiento siempre que se necesite efectuar las tareas que implementa. Además, si un programador comete un error a la hora de escribir un procedimiento, solo debe corregirlo en un lugar; por el contrario, los bloques copiados en múltiples sitios le obligaría a realizar la misma corrección en muchos lugares.
- Apps más fáciles de leer y de entender: Al igual que las variables, los procedimientos suelen tener nombres descriptivos que ayudan a entender qué cometido cumplen en un programa, y a hacer que el propio programa sea más fácil de leer y entender (especialmente, para una tercera persona que no haya escrito ni el procedimiento ni el programa).
- Eficiencia: Las apps que usan procedimientos son más pequeñas, y habitualmente se ejecutan más rápido que las apps que usan un montón de código replicado.
- Reusabilidad: Si hemos escrito un procedimiento que puede ser útil en multitud de apps, basta con copiar y pegar ese procedimiento en todas aquellas apps que puedan necesitarlo.

En esta app necesitamos crear un procedimiento, al que llamaremos "StartPlaying", que se encargará de ocultar el organizador "SelectDifficultyArr", de mostrar el organizador "GuessArr", y de elegir un número aleatorio entre 1 y "HighestNumber". Para crear el procedimiento "StartPlaying" hacemos lo siguiente:

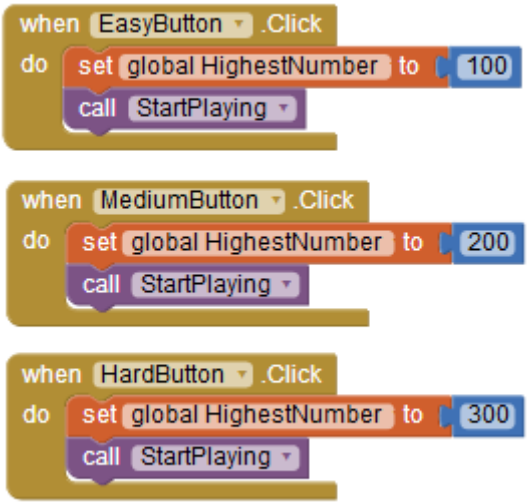
1) En el Editor de Bloques, pinchamos en la bandeja "Procedure".
2) Arrastramos el bloque "to (procedure)", pero aquel que incluye la sección "do", y no el que incluye la sección "result".
3) En este bloque, pinchamos en el texto "procedure" y escribimos "StartPlaying" para establecer el nombre del procedimiento.
4) Dentro de la sección "do" del procedimiento insertamos los bloques para esconder el organizador "SelectDifficultyArr", para mostrar el organizador "GuessArr", y para obtener un número aleatorio entre 1 y "HighestNumber". El código debería quedar como en la figura:



Llamar a un procedimiento.

Ahora que hemos escrito el procedimiento "StartPlaying", vamos a usarlo. Como los tres manejadores de los botones "EasyButton", "MediumButton", y "HardButton" deben realizar las tareas implementadas en este procedimiento, los tres manejadores han de incluir una llamada al procedimiento.

Para hacer una llamada al procedimiento "StartPlaying" pinchamos en la bandeja "Procedures", en la que ahora veremos un bloque "call (StartPlaying)", que es el bloque para llamar al procedimiento que acabamos de crear. Ponemos este bloque dentro de los manejadores de evento de los botones "EasyButton", "MediumButton", y "HardButton", como muestra la figura:

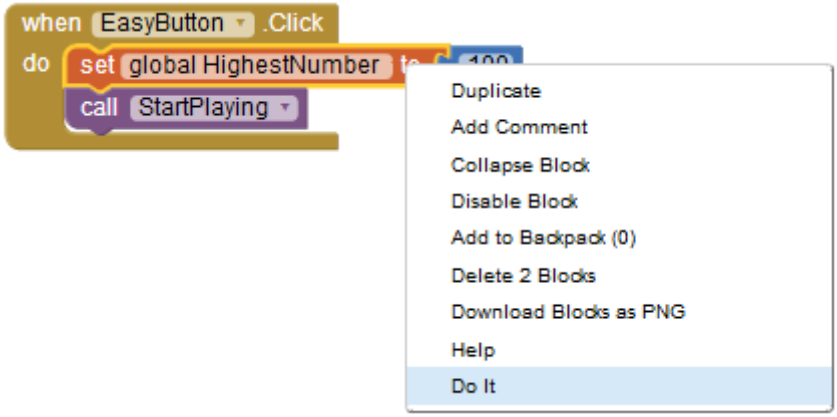


Al clicar en cualquiera de estos tres botones, además de fijar el valor adecuado para la variable "HighestNumber", la app llama al procedimiento "StartPlaying".

Depuración de errores (debugging).

Vamos a probar la app: Conectamos el teléfono (o el emulador) para hacer pruebas en vivo. Nada más arrancar la app, solo debería aparecer el organizador que muestra los botones de selección de dificultad. Al pulsar cualquiera de los tres botones deberíamos ver que este organizador desaparece, y que aparece el organizador donde el usuario hace sus apuestas.

El problema aquí es que no podemos saber si la app ha fijado correctamente el valor de "HighestNumber", y si ha elegido un valor apropiado para "SecretNumber". Afortunadamente App Inventor incluye una característica que nos permite activar bloques individuales para comprobar su funcionamiento, y ver los valores que toman las variables mientras la app se está ejecutando. Se trata de la herramienta llamada "Do it".



Para activar esta herramienta la app la app debe estar en ejecución. Al pinchar sobre cualquier bloque con el botón derecho del ratón y seleccionar la opción "Do it", veremos que ese bloque se ejecuta en el teléfono. Suponer que queremos comprobar si el manejador "when (EasyButton).Click" funciona correctamente. Pinchamos con el botón derecho del ratón sobre el bloque "set (global HighestNumber) to (100)" y elegimos "Do it". Después repetimos la misma operación sobre el bloque "call (StartPlaying)". Veremos que el teléfono muestra la pantalla donde el usuario hace su apuesta.

Otros bloques también proporcionan comentarios emergentes que nos indican cuál es su valor actual. Así, para comprobar cuál es el valor actual de las variables "HighestNumber" y "SecretNumber" podemos crear temporalmente dos bloques "get (variable)" para ambas variables y activar la opción "Do it" en ellos. Es importante entender que los resultados ofrecidos por estos bloques solo se actualizan al volver a seleccionar "Do it". Si el usuario presiona un botón que cambia los valores actuales de estas variables, no veremos el cambio en los valores hasta que volvamos a activar la opción "Do it".

En este ejemplo, parece que todo funciona como esperábamos. Pulsamos el botón "EasyButton", y la app fija el valor de "HighestNumber" a 100. A continuación, comprobamos que la app ha elegido aleatoriamente un valor para "SecretNumber" de 58, que está dentro del rango de 1 a 100.

La técnica de monitorizar los valores de las variables es un método muy útil de depurar programas (práctica a la que en programación se le denomina **debugging**). Si no sabemos qué está pasando en un programa, basta con añadir algunos bloques "get (variable)" de forma temporal, activar la opción "Do it", y comprobar si los bloques están haciendo lo que deben y las variables están tomando los valores adecuados. Pero es importante acordarnos de borrar estos bloques "get" una vez hayamos acabado la depuración del programa.

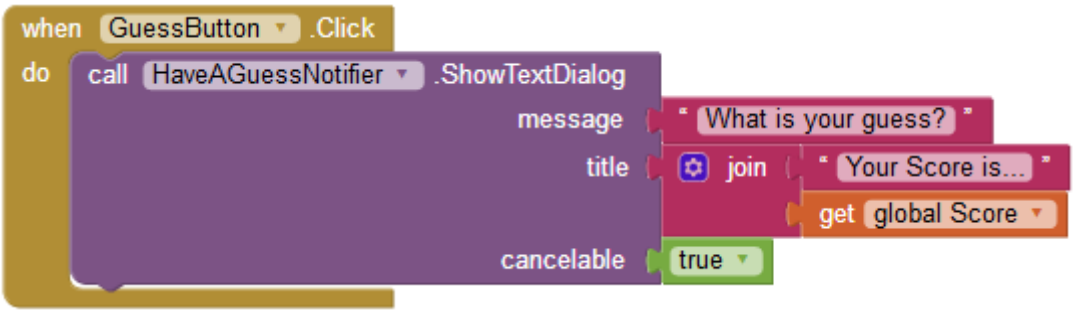
FASE DE JUEGO.

Programar el comportamiento del botón "GuessButton".

Una vez que el jugador llega a la pantalla de las apuestas, al pulsar en el botón "GuessButton" debería abrirse el notificador "HaveAGuessNotifier" para permitirle insertar su apuesta. Esto lo haremos con la función "call (HaveAGuessNotifier).ShowTextDialog", la cual permite mostrar un notificador con caja de texto para insertar datos y un botón de cancelación opcional.

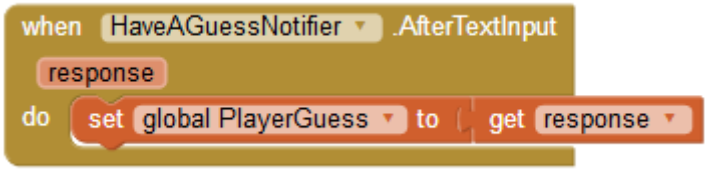
La función "ShowTextDialog" requiere de tres parámetros para funcionar: El parámetro "message" es un texto con el mensaje que debe mostrar el notificador, el parámetro "title" es un texto con el título del notificador, y el parámetro "cancelable" indica si el notificador añadirá un botón para cancelar.

Creemos que para esta app es una buena idea que el notificador también muestre la puntuación actual del jugador. Para ello, en el título del notificador vamos a mostrar un texto del tipo "Your score is … 100" concatenando una cadena ("Your score is …") y el valor actual de la variable "Score" mediante un bloque "join".



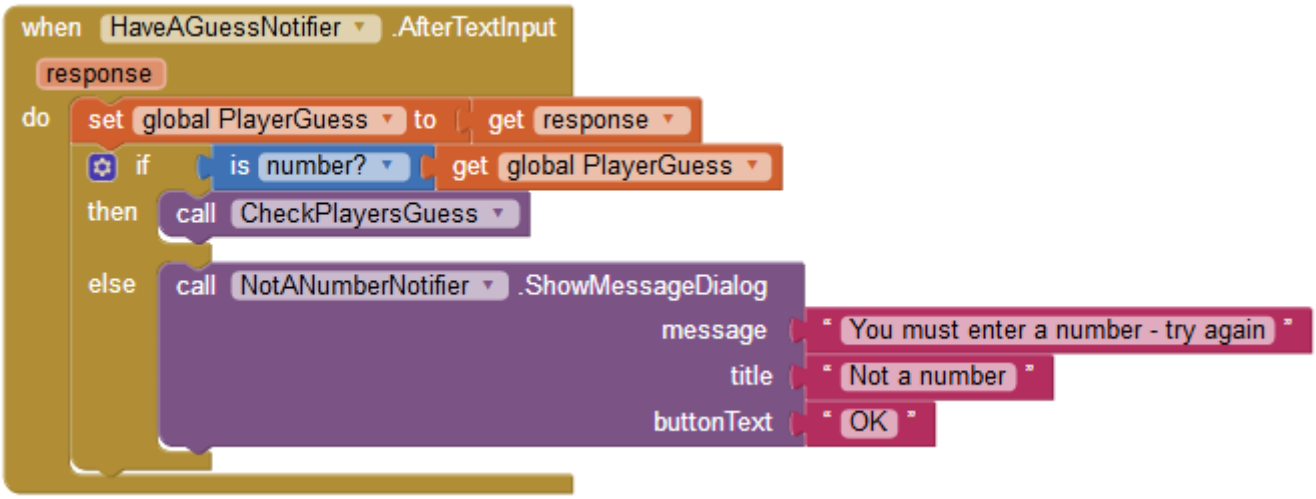
Programar el comportamiento del notificador "HaveAGuessNotifier".

Cuando el jugador escribe su apuesta y pulsa el botón OK, el notificador lanza el evento "AfterTextInput", que puede gestionarse con el manejador "when (Notifier).AfterTextInput". La respuesta del usuario queda almacenada en el argumento "response" del manejador. La app debe guardar esa respuesta en la variable "PlayerGuess", como muestra la figura:



A continuación podríamos ir directamente a comprobar si esta respuesta es igual, mayor, o menor que el número secreto, e indicárselo al jugador. Pero antes de hacer esa comprobación, vamos a asegurarnos de que el jugador ha escrito un número, y no una repuesta en blanco, un texto, o cualquier otra respuesta incoherente. Al proceso de comprobar si los datos insertados por el usuario son correctos se le llama **validación**.

Para hacer nuestra validación, vamos a chequear *si* el usuario ha proporcionado como respuesta un número, mediante el bloque comparador "is a number?" de la bandeja "Math". Si la respuesta del usuario es un número, llamaremos a un procedimiento denominado "CheckPlayerGuess" (que construiremos a continuación). Pero si el usuario no escribió un número, mostramos un nuevo notificador (llamado "NotANumberNotifier") que le indica que la respuesta no es válida, y que debe insertarla otra vez:



(Notar que para poder añadir la llamada "call (CheckPlayerGuess)", previamente tendremos que haber construido el procedimiento "CheckPlayerGuess". Por el momento vamos a crear este procedimiento vacío; lo programaremos más adelante).

Vamos a probar la app: Al llegar a la pantalla de las apuestas, pulsamos el botón "GuessButton", y debería emerger el notificador "HaveAGuessNotifier". Ahora intentamos escribir una apuesta errónea, por ejemplo, "apple", y presionamos OK. Debería aparecer un notificador que nos indique que ese dato no es correcto. Volvemos a intentar pasar una respuesta errónea, por ejemplo, dejando la caja de texto en blanco. Al presionar en el botón OK, la app debería volver a indicarnos con un notificador que esa apuesta no es correcta. Finalmente escribimos un número, por ejemplo 55. La app debería aceptar ese dato como válido y no mostrar el notificador de error.

Ahora, cuando el usuario inserta una apuesta válida debemos compararla con el número secreto. Esta comparación puede arrojar tres resultados: El usuario propone un número demasiado alto, un número demasiado bajo, o el número correcto. Podemos comprobar estas posibilidades con tres bloques "if". Si el usuario falla en su apuesta debemos reducir su puntuación en 10 puntos, y además debemos controlar si la puntuación cae a cero (en cuyo caso, el jugador pierde la partida).

Vamos a dividir todas estas tareas en dos procedimientos: El procedimiento "CheckPlayerGuess" compara el número secreto con la apuesta del jugador, y le indica si el número apostado es demasiado alto, demasiado bajo, o es el número correcto. Por su parte, el procedimiento "DisplayScore" actualizará la puntuación en pantalla y comprobará si el usuario ha perdido.

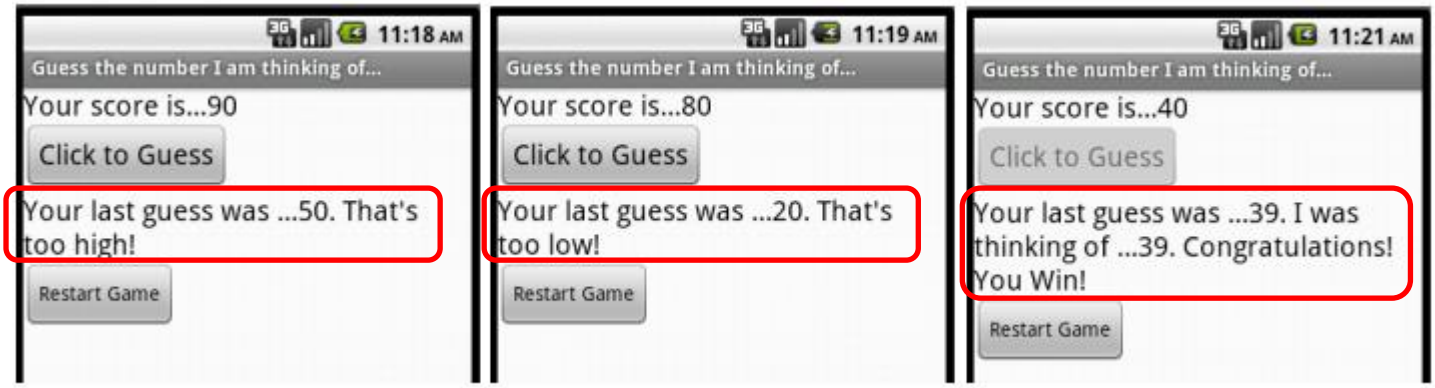
Escribir el procedimiento "CheckPlayerGuess".

Éstas son las tres posibilidades que debemos comprobar, escritas en lenguaje llano:

- Si el valor de la variable "PlayerGuess" es mayor que el valor de la variable "SecretNumber", la app indica "Too High" (demasiado alto) y reduce la puntuación del jugador en 10 puntos.
- Si el valor de "PlayerGuess" es menor que el valor de "SecretNumber", la app indica "Too Low" (demasiado bajo) y reduce la puntuación del jugador en 10 puntos.
- Si "PlayerGuess" es igual a "SecretNumber", la app indica "You win!" (ganaste!).

Vamos a escribir el procedimiento "CheckPlayerGuess" encargado de implementar estas comparaciones, teniendo en mente estas indicaciones:

1) Para informar al usuario del resultado de la comparación, fijaremos la propiedad "Text" de la etiqueta "LastGuessLabel", como muestra la figura. Para poder mostrar esas cadenas combinadas, probablemente necesitaremos concatenar varios textos mediante el bloque "join".

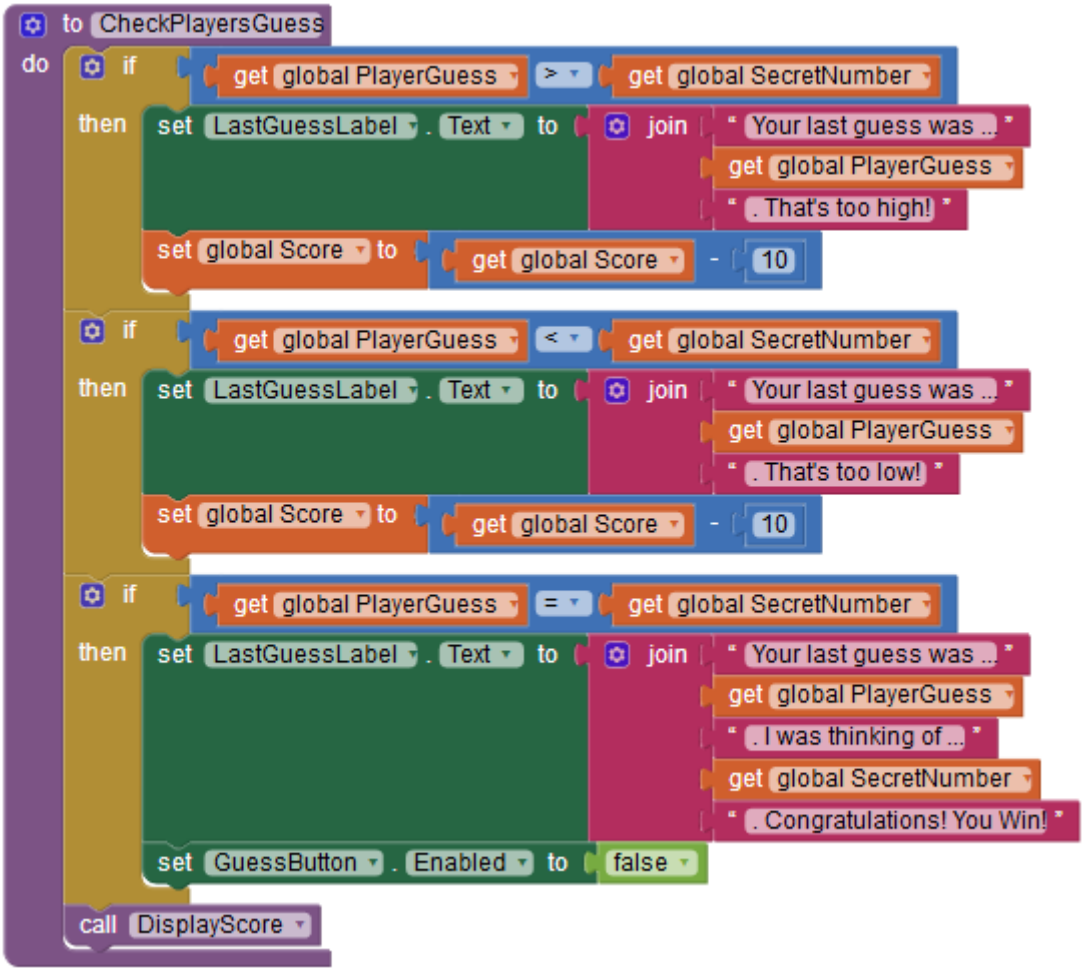


2) Para reducir la puntuación del usuario en 10 puntos, debemos fijar el nuevo valor de la variable "Score" a su valor actual menos 10, como muestra la figura:

3) Tras realizar las tres comparaciones, fijar el texto adecuado de la etiqueta "LastGuessLabel", y actualizar la puntuación del jugador, el procedimiento "CheckPlayerGuess" debería llamar al procedimiento "DisplayScore" para mostrar la puntuación actualizada. (Este procedimiento lo construimos vacío para poder llamarlo, y lo completaremos más adelante).

A modo de ejemplo, la figura muestra el primero de los tres bloques "if" necesarios para definir el procedimiento "CheckPlayerGuess":

Intenta construir tú mismo el contenido del procedimiento "CheckPlayerGuess" utilizando tres bloques "if" análogos, y compara tu código con el de la figura:



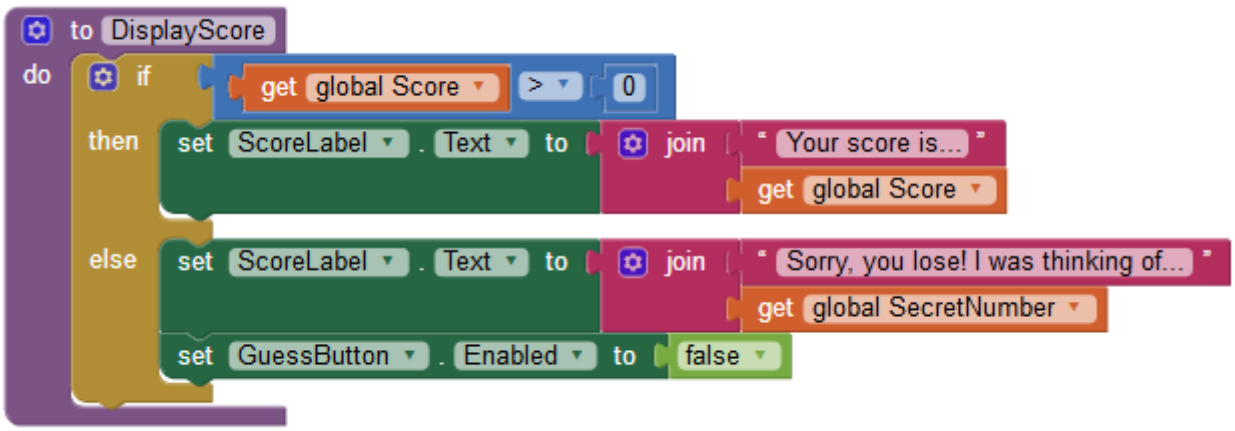
Vamos a probar la aplicación: Una vez hemos llegado a la pantalla de las apuestas, cuando el usuario escribe su apuesta, la app debería indicar si esa apuesta es demasiado alta, demasiado bajo, o la correcta. En caso

de no acertar, la puntuación irá bajando poco a poco, pero esa puntuación todavía no se muestra en la pantalla de apuestas (aún no hemos implementado el procedimiento "DisplayScore"), y solo se mostrará en el notificador donde el usuario escribe su apuesta.

Escribir el procedimiento "DisplayScore".

En nuestra app resulta muy conveniente tener un procedimiento independiente que actualice la puntuación y que compruebe si el usuario ha perdido la partida si su puntuación ha caído a cero. De esta forma, será muy fácil modificar el juego en el caso de que queramos cambiar la puntuación de alguna otra forma. (Por ejemplo, podríamos añadir un temporizador de forma que si el usuario tarda demasiado en dar su apuesta, sea penalizado con una pérdida adicional de puntos).

Por consiguiente, vamos a crear un nuevo procedimiento llamado "DisplayScore", que se encargue de comprobar si el valor de la variable "Score" es mayor que cero. En ese caso, el procedimiento ajustará el valor de la propiedad "Text" de la etiqueta "ScoreLabel" al valor actual de la puntuación (almacenado en la variable "Score"). En caso contrario (esto es, si la puntuación ya ha caído a cero), dirá que el jugador ha perdido y revelará el valor de "SecretNumber" (fijando adecuadamente el valor de la propiedad "Text" de "ScoreLabel"), y evitará que el jugador haga una nueva apuesta (fijando la propiedad "enabled" de "GuessButton" a "false"). Recordar que para implementar condicionales del tipo "si – entonces - si no" necesitaremos un bloque "if – then - else":

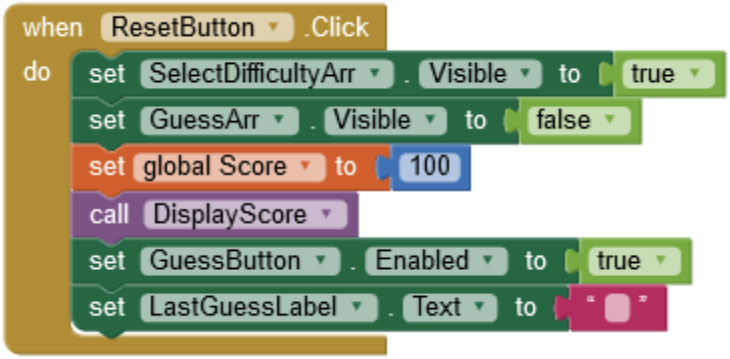


FASE DE RESETEO.

Para hacer que el juego vuelva a la pantalla de selección del nivel de dificultad, debemos revertir los ajustes de las propiedades "Visible" de los organizadores "SelectDifficultyArr" y "GuessArr". Además, debemos acordarnos de:

- Volver a fijar al valor de "Score" a 100.
- Volver a habilitar el botón "GuessButton".
- Limpiar el texto de la etiqueta "LastGuessLabel".

Intenta escribir el programa por ti mismo. El resultado debería parecerse al siguiente:



Ahora nuestro juego de adivinar el número está plenamente operativo. Por supuesto, los valores usados aquí para las distintas variables no son obligatorios. Por ejemplo, podríamos fijar el valor de "HighestNumber" en la pantalla de selección de dificultad a unos valores distintos de 100, 200, y 300. O podríamos cambiar el valor inicial de la puntuación, la cantidad de puntos que restamos por cada fallo, etc. (Por supuesto, si cambiamos algunos de estos valores, no debemos olvidar reflejar esos cambios en las etiquetas que los referencien).

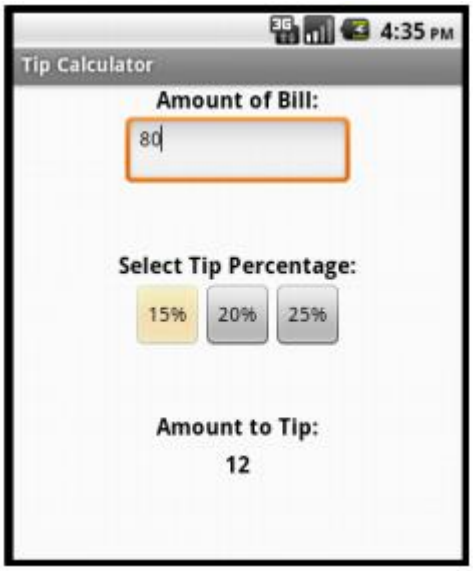
Algunas modificaciones que podríamos implementar para este programa serían:

- Cambiar el sistema de puntuación para que los niveles de mayor dificultad le den al usuario una mayor puntuación inicial.
- Añadir notificadores para que el usuario pueda insertar el valor inicial de "Score" y el valor de "HighestNumber" para cada nivel de dificultad.
- Añadir algún tipo de ocurrencia especial en el caso de que el usuario acierte el número secreto a la primera (reproducir un sonido de trompetas o fuegos artificiales), hacer que aparezca una imagen con unos laureles o una copa, etc.

4.17. EJERCICIOS DEL CAPÍTULO 4.

Ejercicio 4.1. Calculadora de propinas.

A menudo, las apps necesitan que el usuario introduzca algunos datos para poder funcionar. Las **cajas de texto** (componente "TextBox" en la bandeja "User Input") es un área rectangular en la que el usuario puede introducir los datos solicitados a través del teclado del dispositivo. La propiedad más importante de un "TextBox" es su propiedad "Text", a través de la cual podemos acceder al texto escrito por el usuario. Normalmente, los datos proporcionados por el usuario a través de una caja de texto se suelen almacenar en variables. Por ejemplo, el siguiente código inicializaría la variable "userName" a una cadena vacía, y después almacenaría en esa variable el nombre que el usuario haya escrito en la caja de texto "TextBox1":



La tarea de este ejercicio es crear una app llamada "TipCalculator" que nos permita calcular la propina a dejar en un restaurante. La interfaz de usuario debería tener una caja de texto para insertar el importe de la cuenta (sin incluir propina), tres botones para calcular una propina equivalente a un 15%, a un 20%, o a un 25% del importe de la cuenta (dependiendo del servicio recibido y de lo generosos que nos sintamos), y una etiqueta donde mostrar la propina calculada. Añade también las etiquetas explicativas que creas necesarias. Un ejemplo de interfaz sencilla sería la ilustrada en la figura.

Ejercicio 4.2. Dale al huevo.

En este ejercicio vas a construir una app que consista en darle golpes a un huevo hasta romperlo. Para poder golpear al huevo, lo implementarás como un botón (con la imagen de un huevo) sobre el que podrás tocar. Cuando hayas golpeado al huevo un total de 5 veces, el huevo se romperá mostrando la imagen de un huevo roto, la app reproducirá un sonido, y el móvil vibrará. En ese momento aparecerá un botón para reiniciar la app y volver a la imagen del huevo sin romper. Para construir esta app necesitarás tres archivos multimedia: las imágenes *egg.png* y *crackedEgg.png*, y el sonido *crackedEgg.mp3*.

PISTA: Para llevar la cuenta del número de golpes, necesitarás incrementar en una unidad el valor de la variable que almacena el número actual de golpes cada vez que se presione el botón del huevo.

Ejercicio 4.3. Programa una calculadora de área de rectángulos. Amplía su funcionalidad para que también pueda usarse en el cálculo de las áreas (y volúmenes) de otras figuras geométricas.

Ejercicio 4.4. Luz.
Crea una app que simule una luz que pueda encenderse y apagarse con un interruptor. Inicialmente, la luz está apagada y el interruptor desconectado. Cuando el usuario pulsa en el interruptor, su imagen cambia a la de un interruptor conectado, y la luz se enciende. Si el usuario vuelve a presionar en el interruptor, su imagen cambia de nuevo a la de un interruptor desconectado, y la luz se apaga. Para hacer esta app necesitarás los archivos *lightBulbOn.png*, *lightBulbOff.png*, *lightSwitchOn.png*, *lightSwitchOff.png*, y *switch.mp3*.

PISTA: Es recomendable crear sendos procedimientos para encender y apagar la luz (y para cambiar la imagen del interruptor).

PISTA 2: Para el interruptor, es necesario utilizar un componente botón. Para la luz, basta con usar un componente "Image".

Ejercicio 4.5. Moneda al aire.
El objetivo de este ejercicio es construir un simulador del lanzamiento de una moneda. Cuando el usuario presiona un botón la app "lanza una moneda al aire", y muestra por pantalla la imagen de una moneda mostrando la cara, o de una moneda mostrando la cruz, según el resultado. Además, una etiqueta bajo la imagen indica si ha salido cara o cruz. Para este ejercicio necesitarás dos archivos, uno con la imagen de la cara (*heads.png*) y otro con la imagen de la cruz (*tails.png*).

Ejercicio 4.6. Contador de monedas.
En este ejercicio crearás una app para un contador de monedas, llamada "ChangeCounter". La app debe mostrar las imágenes de cuatro monedas: una de 5 céntimos, otra de 10 céntimos, otra de 20 céntimos, y otra de 50 céntimos. (Busca las imágenes tú mismo en internet, pero procurando que tengan todas el mismo tamaño). La app tendrá una variable global llamada "Total" que comienza con un valor de 0. Cada vez que el usuario pulse en una de las imágenes de las monedas, el valor de esa moneda se deberá sumar al valor de "Total". La app también debe mostrar por pantalla el valor actual de esa variable.

Ejercicio 4.7. Deslizadores.
El componente "Slider" (bandeja "User Interface") consiste en una pista horizontal con un **deslizador** que el usuario puede mover hacia la izquierda o hacia la derecha para ajustar un cierto valor dentro de un rango de valores posibles. Sus propiedades más destacadas son "MaxValue" y "MinValue", que permiten especificar los valores máximo y mínimo que se podrán ajustar con el deslizador, y "ThumbPosition", que indica la posición actual del deslizador (y por consiguiente, el valor actual fijado por el usuario).

Cada vez que el usuario mueve el deslizador, se activa el evento "PositionChanged". Por consiguiente, podemos usar el manejador de eventos "when (Slider).PositionChanged" para realizar una acción cada vez

que el usuario desplace el deslizador. Notar que este manejador posee un parámetro llamado "thumbPosition" que almacena y nos permite acceder a la posición actual del deslizador.

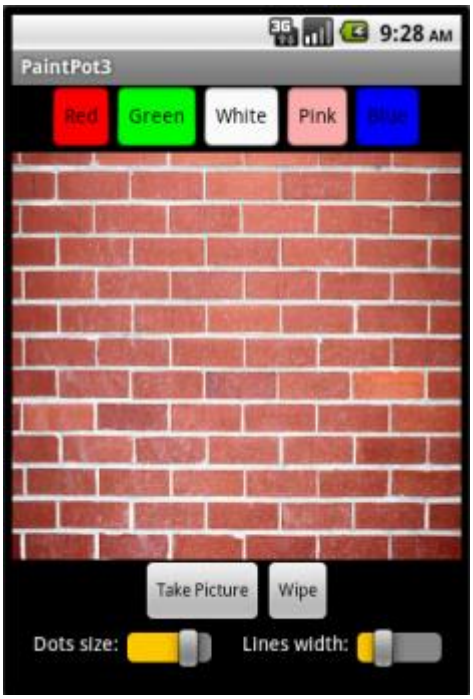
La tarea de este ejercicio es crear una app (llamada "SliderDemo") que nos permita cambiar dinámicamente el tamaño del texto estático de una etiqueta mediante un deslizador. Mediante otra etiqueta la app mostrará el valor actual del deslizador. La app debe funcionar como muestra la figura:

PISTA: Para cambiar el tamaño del texto de una etiqueta, podemos acceder a su propiedad "FontSize". Para cambiar el texto que muestra una etiqueta, debemos acceder a su propiedad "Text". Ambas cosas pueden hacerse con código mediante los bloques "set (Label).FontSize to" y "set (Label).Text to".

NOTA: Utiliza los componentes y fija sus propiedades como se indica en la tabla.

Component	Relevant Property Settings
`Screen1`	AlignHorizontal = *Center* Title = *Slider Demo*
`Slider1`	MaxValue = 100 MinValue = 0 ThumbPosition = 50 Width = *Fill parent*
`LabelSampleText`	Text = *Hello* FontSize = 50
`LabelSliderPosition`	Text = *50.0*

Ejercicio 4.8. PaintPot(3): Deslizadores.
Modifica la app "PaintPot2" para permitir que el usuario pueda elegir el tamaño de los puntos y el grosor de las líneas dibujadas sobre la pared mediante sendos deslizadores (a los que llamaremos "DotSizeSlider" y "LineWidthSlider").



Recordar que el tamaño de los puntos que dibujamos sobre un lienzo puede cambiarse modificando el valor del parámetro "radius" de la función "call (Canvas).DrawCircle". Por su parte, el grosor de las líneas dibujadas sobre un lienzo puede cambiarse modificando la propiedad "LineWidth" del componente "Canvas". Para nuestro programa, inicializa el tamaño de los puntos y el grosor de las líneas a un valor por defecto de 5 píxeles, y permite que el usuario pueda alterar estos valores por defecto dentro de un rango de 2 a 8 píxeles.

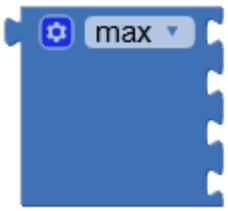
PISTA: Probablemente te convendrá definir una variable, por ejemplo "DotSize", para alojar el valor actual del tamaño de los puntos. No es necesario definir una variable para guardar el valor actual del grosor de línea, porque la propiedad "LineWidth" del componente "Canvas" ya hace esa tarea por nosotros.

Ejercicio 4.9. Saludos internacionales.
Construye una app en la que el usuario pueda presionar en cuatro botones de banderas (Reino Unido, Alemania, Francia, e Italia). Al pulsar en cualquiera de las banderas la app lanza un notificador, que le pide al usuario que inserte su nombre. A continuación, y en una etiqueta bajo las banderas, el programa muestra un saludo personalizado para el usuario en el idioma de la bandera que ha seleccionado. (Por ejemplo, "Guten morgen, Roberto", "Good morning, Roberto", "Buongiorno, Roberto", o "Bonjour, Roberto"). NOTA: Los archivos de imagen para las banderas los tienes disponibles en los archivos de los alumnos, con los nombres *Germany_flag.png*, *France_flag.png*, *Italy_flag.png*, y *UK_flag.png*.

CUIDADO: El saludo no debería aparecer al pulsar el botón de la bandera, sino después de introducir el nombre en el notificador emergente. La pulsación de la bandera únicamente le indica a la app el idioma del saludo a mostrar ("Guten Morgen", "Good Morning", "Buongiorno", o "Bonjour"), pero el texto de la etiqueta solo debería actualizarse tras introducir el nombre en el notificador (evento "AfterTextInput").

Ejercicio 4.10. Añade un seguimiento estadístico a la app "SoundFX" que muestre cuántas veces se pulsa cada botón, mostrando una etiqueta contadora bajo los distintos botones. (Es muy recomendable definir múltiples variables para almacenar el número de pulsaciones de cada botón). Además, añade otra etiqueta que indique cuál es el sonido más popular, y que se actualice tras cada pulsación de botón (usa un procedimento).

PISTA: Para detectar qué sonido es el más popular, necesitarás utilizar el bloque "max" de la bandeja de bloques matemáticos, el cual te permite calcular el valor máximo de un conjunto de número (o variables). Después, deberías comparar ese valor con el valor de todas las variables que almacenan el número de pulsaciones de cada botón.



Ejercicio 4.11. Pares o nones.
Pares o nones es un juego tipo "piedra, papel, o tijera" para dos jugadores. Un jugador elige pares y el otro nones (impares). A continuación, cada jugador muestra su mano derecha, sacando tantos dedos como crea conveniente (de 0 a 5). Se suman los dedos sacados por ambos jugadores, y si el número es par gana el jugador que eligió pares, y si es impar, gana el jugador que eligió nones.

Construye una app que simule este juego, en el que ambos usuarios compartirán el mismo dispositivo. Tienes total libertad para elegir la interfaz de usuario, pero es recomendable construir una botonera del 0 al 5 para el jugador de los pares, y otra botonera del 0 al 5 para el jugador de los nones. Los jugadores insertan su apuesta, y después uno de los dos pulsa un botón para mostrar el resultado. También es buena idea que la app pregunte el nombre del jugador de los pares y del jugador de los nones, mediante notificadores o cuadros de texto. La app debe indicar claramente cuál de los dos jugadores ha ganado la ronda.

AMPLIACIÓN: El jugador que va con los pares tiene ventaja (6 resultados favorables) frente al que va con los nones (5 resultados favorables). Completa la app para que, si ambos jugadores sacan 5 dedos (puntuación total de 10), la app indique que hay empate y les pida a los jugadores que vuelvan a jugar.

5. TEMPORIZADORES.

Si pensamos en el reloj de un horno (o de un microondas), entenderemos que cumple dos funciones:

- Nos da la hora.
- Cuando el horno está encendido, muestra una cuenta atrás con el tiempo que queda, y realiza alguna acción cuando el temporizador llega a cero (apaga el horno, y hace sonar un timbre).

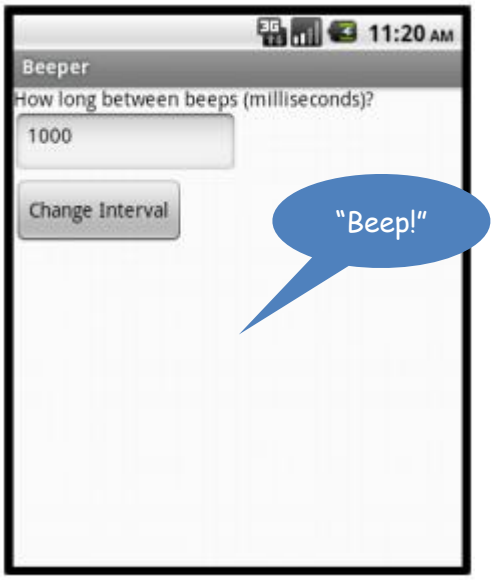
Este reloj también podría programarse para hacer algo más, como encender el horno a la hora programada, o cambiar la temperatura en ciertos instantes de tiempo marcados de antemano. ¿Qué tiene que ver todo esto con App Inventor? Bueno, App Inventor incorpora un componente "Clock" que funciona de forma similar al reloj de un horno. Este componente es capaz de:

- Proporcionar información sobre la hora (y la fecha) actuales.
- Iniciar una temporización y hacer algo útil cuando el tiempo se ha agotado.

Nosotros utilizaremos el componente "Clock" exclusivamente como temporizador. Una diferencia importante entre el reloj de App Inventor y el reloj de un horno es que el temporizador del componente "Clock" se reinicia automáticamente una y otra vez mientras no lo apaguemos. Afortunadamente esto no ocurre en los hornos ni en los microondas, ya que correríamos el riesgo de quemar la comida o incluso incendiar la casa.

5.1. TEMPORIZADORES.

Casi todas las apps que hemos construido hasta ahora cuentan con que el usuario realice una acción que obligue a la app a responder. Esta acción podría ser presionar un botón, arrastrar un dedo por la pantalla, o sacudir el teléfono. Por el contrario, los temporizadores permiten que la app realice ciertas acciones a intervalos de tiempo regulares, incluso aunque el usuario no haya interaccionado con el dispositivo.



Para mostrar el uso de los temporizadores vamos a construir una app que pite a intervalos regulares. La app comenzará pitando a cada segundo, pero el usuario podrá cambiar este intervalo de tiempo escribiendo un número en milisegundos y presionando un botón.

Para construir esta app, necesitaremos el audio *beep.wav*, que podremos encontrar en los archivos de los alumnos.

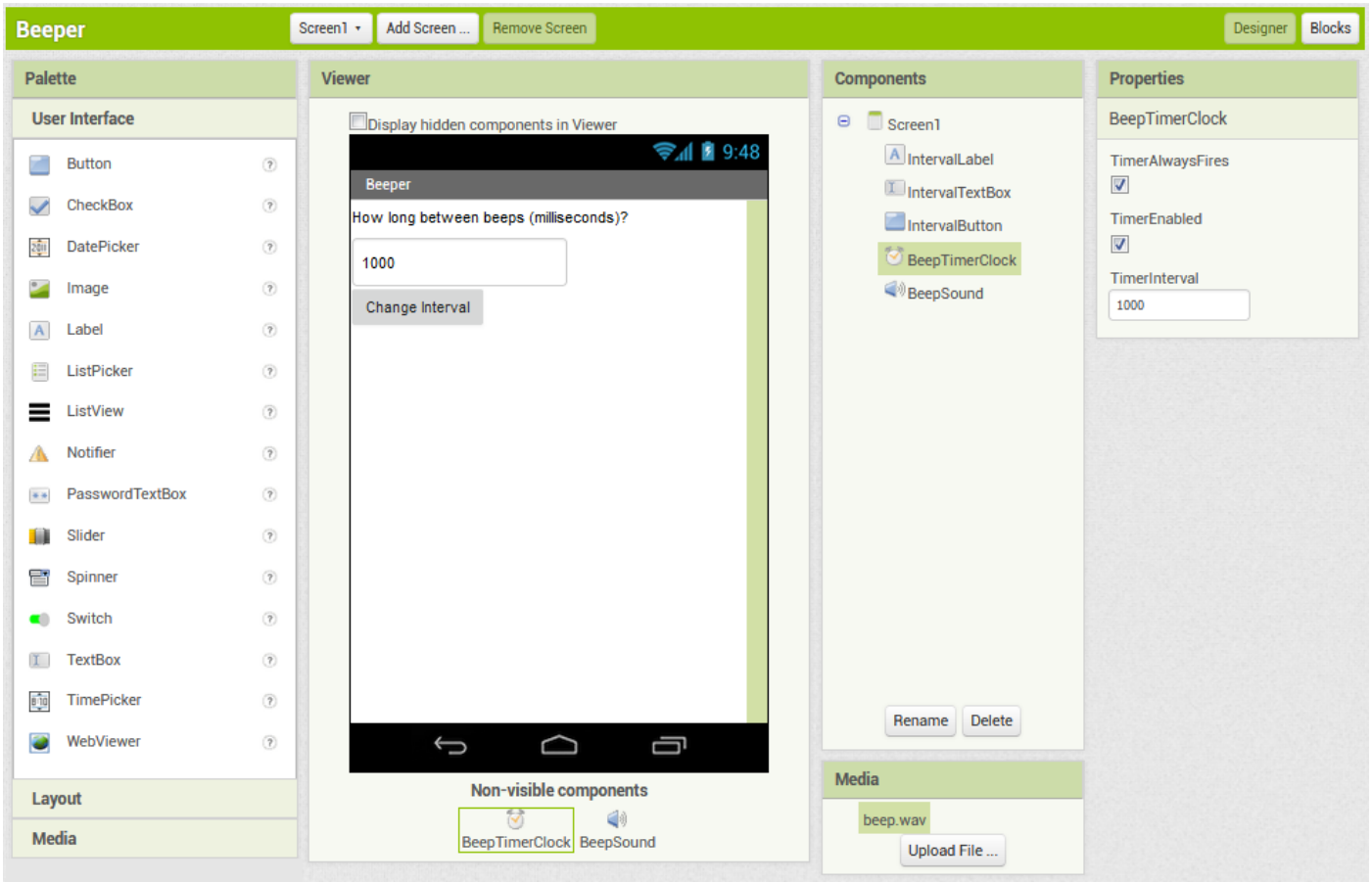
5.2. COMENZAR CON LA APP "BEEPER".

Comenzamos un nuevo proyecto llamado "Beeper". Para este nuevo proyecto usaremos unos componentes muy similares a los que ya usamos en la app "WeekdayCalculator", por lo que podemos abrir esta última y elegir "Projects" → "Save Project as…".

5.3. DISEÑAR LOS COMPONENTES DE "BEEPER".

La tabla lista todos los componentes necesarios para esta app, junto con los valores a fijar para sus propiedades, y la figura muestra el diseño de la interfaz de usuario.

Beeper			
Screen1 properties	`Title`: Beeper		
Components	**What do I rename it?**	**What does it do?**	**What properties do I set?**
`Label`	`IntervalLabel`	Instructs the user	`Text`: "How long between beeps (milliseconds)?"
`TextBox`	`IntervalTextBox`	Where the user enters the time to wait between beeps	`Hint`: "1000 = 1 sec" `NumbersOnly` = Yes (selected) `Text`: "1000"
`Button`	`IntervalButton`	Sets the time between beeps to the value typed in `IntervalTextBox`	`Text`: "Change Interval"
`Sound`	`BeepSound`	Plays the beep.wav sound	`MinimumInterval`: 1 `Source`: beep.wav
`Clock` Palette group: User Interface	`BeepTimerClock`	When the timer fires, plays a beep sound	`TimerAlwaysFires`: Yes (selected) `TimerEnabled`: Yes (selected) `TimerInterval`: 1000
Media files			
1 sound file downloaded from our website: beep.wav			



El componente "Clock" llamado "BeepTimerClock" tiene tres propiedades de importancia, que pasamos a describir a continuación:

- "TimerAlwaysFires": Al activar esta casilla le decimos al temporizador que siga funcionando, incluso si el teléfono pasa a una pantalla diferente, o si el usuario pone el teléfono en modo No Molestar, Avión, etc.

En el caso de una aplicación como una alarma, queremos que el temporizador compruebe todo el tiempo si ha finalizado su cuenta atrás para hacer sonar la alarma, incluso aunque el teléfono reciba una llamada o el usuario lo apague porque se ha ido a dormir. Pero en un juego probablemente querremos que se pause si recibimos una llamada o nos vamos a la cama. Así pues, para una alarma fijaríamos "TimerAlwaysFires" a "true" (casilla activada), pero en el caso de un videojuego la fijaríamos a "false" (casilla desactivada).

- "TimerEnabled": Al seleccionar esta casilla le decimos al temporizador que empiece a contar en cuanto arranque la app. Si deseleccionamos esta casilla el temporizador retrasará su activación hasta que lo arranquemos manualmente en un programa.

  Al ajustar esta propiedad debemos decidir si queremos que el temporizador se dispare[7] nada más empezar la aplicación, o solo cuando finalice la cuenta atrás del reloj que el usuario haya activado como resultado de alguna acción. Por ejemplo, una app de reloj de noche que simplemente muestre la hora probablemente necesite disparar el temporizador nada más empezar la aplicación, pero una app para un cronómetro solo necesita activar el temporizador cuando el usuario presione el botón de Empezar.

- "TimerInterval": Esta propiedad almacena el valor por defecto del intervalo temporal que usará el temporizador para dispararse. En esta app el temporizador se dispara cada segundo ($1000\,ms = 1\,s$). Como ocurre con todas las propiedades, podemos cambiar este valor por defecto mediante bloques de código.

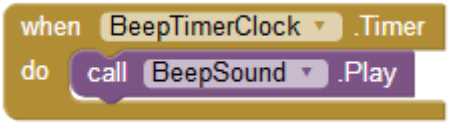
  Con esta propiedad podemos establecer la frecuencia con la que se dispara el temporizador. Para una app de cronómetro probablemente querremos que se dispare muchas veces a cada segundo (digamos, cada $100\,ms$), para poder capturar al menos hasta las décimas de segundo. Pero si queremos que un timbre suene cada 60 segundos, el valor de "TimerInterval" puede ser mucho más amplio (por ejemplo, $60000\,ms$).

La otra novedad con la que nos hemos encontrado en esta app es la propiedad "MinimumInterval" del componente de sonido "BeepSound". Aquí la hemos ajustado a 1 (milisegundos). Su valor por defecto es de $1000\,ms$ (1 segundo), y para la mayoría de las apps ese valor es correcto. Pero en nuestro caso, si el usuario deseara reproducir un sonido con mucha frecuencia (digamos, cada 20 milisegundos), querremos que la app sea capaz de reproducir sonidos consecutivos de forma que se perciba el corte entre una reproducción y la siguiente.

5.4. AÑADIR COMPORTAMIENTOS A LOS COMPONENTES DE "BEEPER".

REPRODUCIR UN SONIDO AL DISPARARSE EL TEMPORIZADOR.

Cuando el temporizador se dispara (esto es, cuando su cuenta atrás llega a cero) genera un evento "Timer" (de la misma forma que cuando pulsamos un botón, éste genera un evento "Click"). En el Editor de Bloques acudimos a la bandeja de "BeepTimerClock", y seleccionamos el boque manejador de evento "when (BeepTimerClock).Timer". En la sección "do" del manejador insertamos un bloque "call (BeepSound).Play", para reproducir el sonido *beep.wav* asociado a la propiedad "Source" del componente "BeepSound".



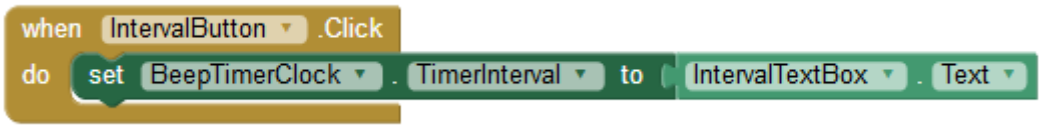
[7] Al decir que un temporizador se "dispara" nos estamos refiriendo al momento en el que la cuenta atrás del temporizador llega a cero, y lanza la acción para la que está programado.

Si ejecutamos la app escucharemos un pitido cada segundo justo después de arrancar la aplicación.

PERMITIR AL USUARIO CAMBIAR EL INTERVALO DE REPETICIÓN.

Ahora mismo, si el usuario escribe un número (en milisegundos) en la caja de texto "IntervalTextbox" y pulsa el botón, la app no hace nada. Lo que queremos es que el valor de la propiedad "TimerInterval" del componente "BeepTimerClock" cambie en función del número escrito por el usuario. (Por ejemplo, si el usuario escribe 10000, el pitido debería repetirse cada 10 segundos).

Para programar este comportamiento, debemos añadir un manejador de eventos "when (IntervalButton).Click" y cambiar la propiedad "TimerInterval" de "BeepTimerClock" al valor escrito por el usuario en la caja de texto "IntervalTextBox":



Y con esto hemos terminado la app. Probamos a escribir unos cuantos valores en la caja de texto para ver cómo responde el temporizador. Si escribimos intervalos de tiempo muy pequeños, puede que oigamos que los pitidos se solapan, y por consiguiente, que suenen con un mayor volumen.

En la siguiente sección vamos a construir una app para jugar al escondite usando el teléfono. Esto nos permitirá aprender cómo se activa y se desactiva un temporizador.

5.5. COMENZAR CON LA APP "WHERE ARE YOU HIDING?".

Vamos a construir una app que reproduzca un sonido (en este caso, el balido de una oveja) a intervalos de tiempo aleatorios. El usuario puede elegir un tiempo máximo entre balidos consecutivos y un tiempo de retardo después de que comience el juego. El juego empieza al tocar sobre una oveja, y finaliza al presionar un botón de Stop. ¿Para qué podríamos usar esta app? He aquí algunas ideas:

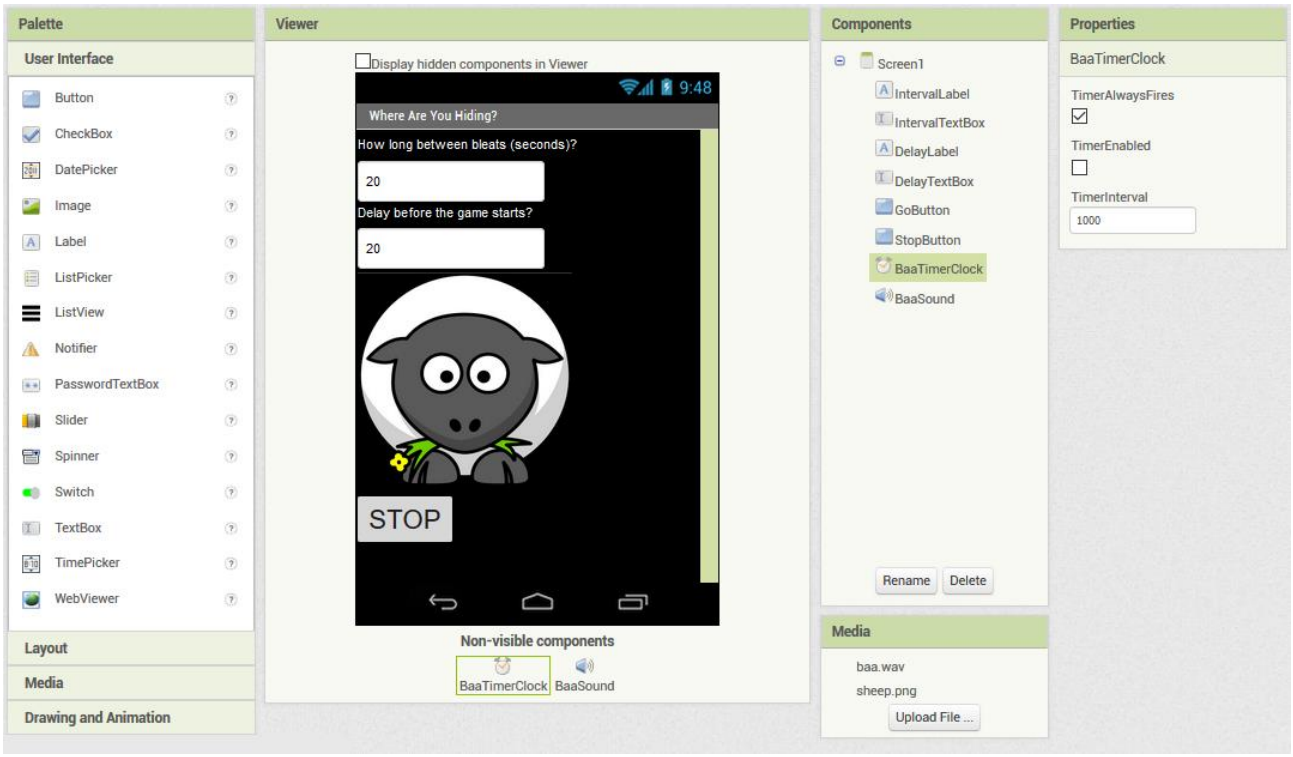
- Jugar al escondite: Los jugadores ponen el volumen de sus teléfonos al máximo, ejecutan la app, y se esconden. Los jugadores no saben cuándo sonará un balido que delate su posición.
- Jugar a Marco Polo: Todos los jugadores excepto el que amaga ejecutan esta app en sus teléfonos. El jugador que amaga intenta pillar (con los ojos tapados) a los demás jugadores en la habitación. Si su teléfono suena, el jugador debe quedar inmóvil 3 segundos.
- Jugar a la búsqueda del tesoro: Los jugadores ejecutan la app y esconden sus teléfonos. El jugador que busca tiene 60 segundos para encontrar tantos teléfonos como pueda.

Para esta app necesitaremos dos archivos multimedia: El audio *baa.wav* y la imagen *sheep.png*. Ambos recursos están disponibles en los archivos de los alumnos.

5.6. DISEÑAR COMPONENTES DE "WHERE ARE YOU HIDING?".

La tabla muestra los componentes necesarios para crear la interfaz de usuario, junto con los ajustes requeridos para sus propiedades.

Where Are Ewe Hiding			
Screen1 properties	`Title`: Where Are Ewe Hiding? `BackgroundColor`: Black		
Components	**What do I rename it?**	**What does it do?**	**What properties do I set?**
`Label`	`IntervalLabel`	Instructs the user	`Text`: "How long between Bleats (seconds)?" `TextColor`: White
`Label`	`DelayLabel`	Instruct the user	`Text`: "Delay before the game starts?" `TextColor`: White
2 `TextBoxes`	`IntervalTextBox` `DelayTextBox`	Where the user enters the times for the interval between beeps and the delay before the game starts	For both text boxes: `Hint`: "Seconds" `NumbersOnly` = Yes (selected) `Text`: "20"
`Button`	`GoButton`	Starts the game	`Text`: None `Image`: sheep.png `Width` and `Height`: 200 pixels
`Button`	`StopButton`	Stops the game	`FontSize`: 28 `Text`: "STOP" `Width` and `Height`: 200 pixels
`Sound`	`BaaSound`	Plays the baa.wav sound	`MinimumInterval`: 500 `Source`: baa.wav
`Clock` Palette group: User Interface	`BaaTimerClock`		`TimerAlwaysFires`: Yes (selected) `TimerEnabled`: No (unselected) `TimerInterval`: 1000



La figura muestra el diseño de la interfaz de usuario. La interfaz de usuario por la que hemos optado aquí es bastante sencilla. (Tal vez prefieras añadir algunos organizadores para ordenar un poco más los distintos componentes). Notar que en este caso hemos fijado la propiedad "TimerEnabled" del componente "BaaTimerClock" a "false" (esto es, desactivada). Esto significa que el temporizador no se disparará hasta

que el usuario no toque el botón de la oveja "GoButton". Cuando construimos nuestras propias apps de forma autónoma es fácil olvidarse de esto, así que vale la pena habituarse a desactivar los temporizadores si no queremos que se disparen nada más arrancar la app.

Además, notar que en nuestra app le estamos permitiendo al usuario insertar los intervalos de tiempo en segundos, en vez de hacerlo en milisegundos. Ello nos obligará a hacer la conversión, multiplicando el número de segundos introducidos en las cajas de texto por 1000.

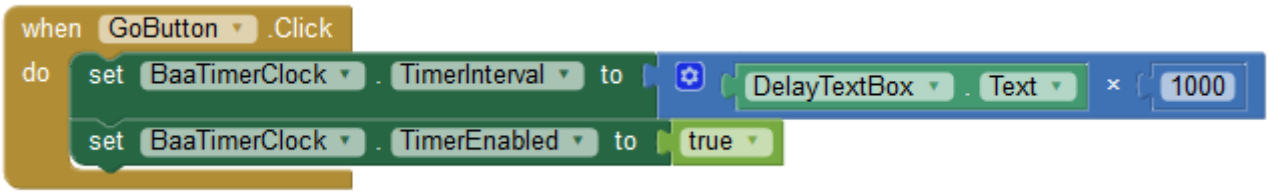
5.7. AÑADIR COMPORTAMIENTOS A LOS COMPONENTES DE "WHERE ARE YOU HIDING?".

PROGRAMAR EL BOTÓN DE ARRANQUE.

Cuando el usuario toque la oveja (botón "GoButton") para iniciar el juego, queremos que la app haga lo siguiente:

1) Convertir el valor introducido por el usuario en "DelayTextbox" de segundos a milisegundos.
2) Fijar el intervalo del temporizador al valor obtenido en el paso 1 para que la oveja no empiece a balar nada más presionar el botón, sino cuando se agote el tiempo introducido en "DelayTextbox".
3) Activar el temporizador.

El programa que implementa este comportamiento es el siguiente:



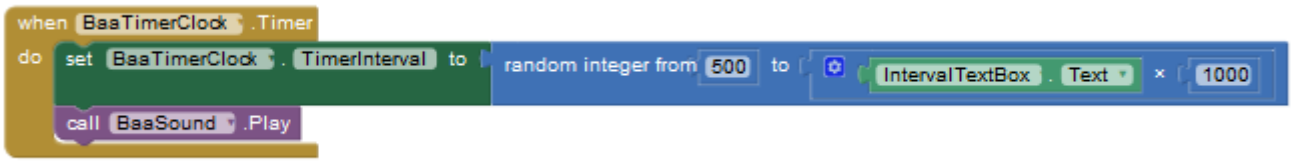
¿Cómo funciona este programa? Al pulsar en el botón de la oveja, fijamos el intervalo del temporizador "BaaTimerClock" al valor escrito por el usuario en la caja de texto "DelayTextbox" (valor que previamente hemos convertido de segundos a milisegundos, tal y como requiere la propiedad "TimerInterval"). A continuación activamos el temporizador, el cual comenzará su cuenta atrás y se disparará al llegar a cero.

PROGRAMAR EL TEMPORIZADOR.

Cada vez que el temporizador "BaaTimerClock" se dispare, queremos cambiar su propiedad "TimerInterval" a un número aleatorio entre, digamos, medio segundo y el tiempo máximo insertado por el usuario en "IntervalTextbox", y transcurrido este tiempo, que reproduzca el sonido del balido. La app continuará reproduciendo ese sonido a intervalos de tiempo aleatorios hasta que el usuario presione el botón "StopButton".

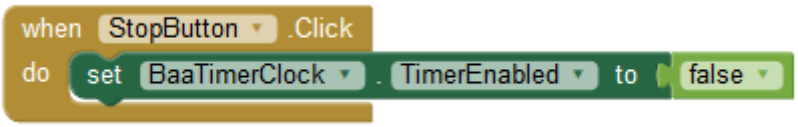
Así, supongamos que el usuario fija el valor de "DelayTextBox" a 20 segundos y el valor de "IntervalTextBox" a 15 segundos. El primer balido sonará transcurridos los 20 segundos del retardo inicial, y el resto de balidos podrían sonar tras cualquier intervalo temporal entre 0,5 y 15 segundos. Así, el segundo balido podría sonar a los 6 segundos y medio, el tercero 10 segundos después, el cuarto 2 segundos más tarde, etc. Incluso podríamos oír dos balidos consecutivos, y tener que esperar 15 segundos hasta oír el tercero. Cuanto más largo sea el tiempo máximo, más probable es que tengamos periodos de silencio más prolongamos. Podríamos usar esta libertad para fijar distintos tiempos máximos a los distintos jugadores: Por ejemplo, los chicos de mayor edad tendrían un tiempo máximo de 10 segundos, y los más pequeños un tiempo máximo de 30 segundos.

La figura muestra el programa que implementa este comportamiento:



PROGRAMAR EL BOTÓN DE PARADA.

El último bloque de código simplemente indica que, cuando el usuario pulsa el botón "StopButton", la app debe dejar de reproducir sonidos de balidos. Para ello, dentro del manejador de evento "when (StopButton).Click", desactivamos el temporizador "BaaTimerClock":



5.8. TEMPORIZADORES MÚLTIPLES.

Las apps complejas como los videojuegos necesitan temporizar múltiples eventos al mismo tiempo. Por ejemplo, en el juego Pac – Man se necesitan temporizadores para:

- Controlar las animaciones, como por ejemplo, Pac – Man abriendo y cerrando la boca y los fantasmas cambiando de apariencia y titilando.
- Insertar pausas entre los niveles.
- Realizar una cuenta atrás cuando Pac – Man se come una píldora.

Sería imposible hacer todo esto con un solo reloj. Afortunadamente en App Inventor podemos añadir a nuestros proyectos tantos relojes como queramos. Además, al usar nombres específicos para los distintos relojes haremos que nuestras apps sean más fáciles de entender. En el siguiente ejemplo aprenderemos a usar tres emporizadores en una sola app.

5.9. COMENZAR CON LA APP "SPLAT THE RAT".

Este videojuego muestra por pantalla una rata a intervalos de tiempo aleatorios. Cuando aparece, el jugador debe tocar a la rata rápidamente tantas veces como pueda antes de que vuelva a desaparecer, para aplastarla. Cada vez que el jugador toca a la rata gana 1 punto. El juego está limitado a un tiempo total de 30 segundos.

Para esta app necesitaremos varios archivos multimedia, llamados *splat.wav*, *squeak.wav*, *rat.png*, y *storm_drain_cover.png*. Todas estas imágenes y sonidos están disponibles en la carpeta de archivos de los alumnos.

Abre un nuevo proyecto en App Inventor llamado "SplatTheRat". A continuación, conecta tu dispositivo o el emulador para hacer pruebas en vivo.

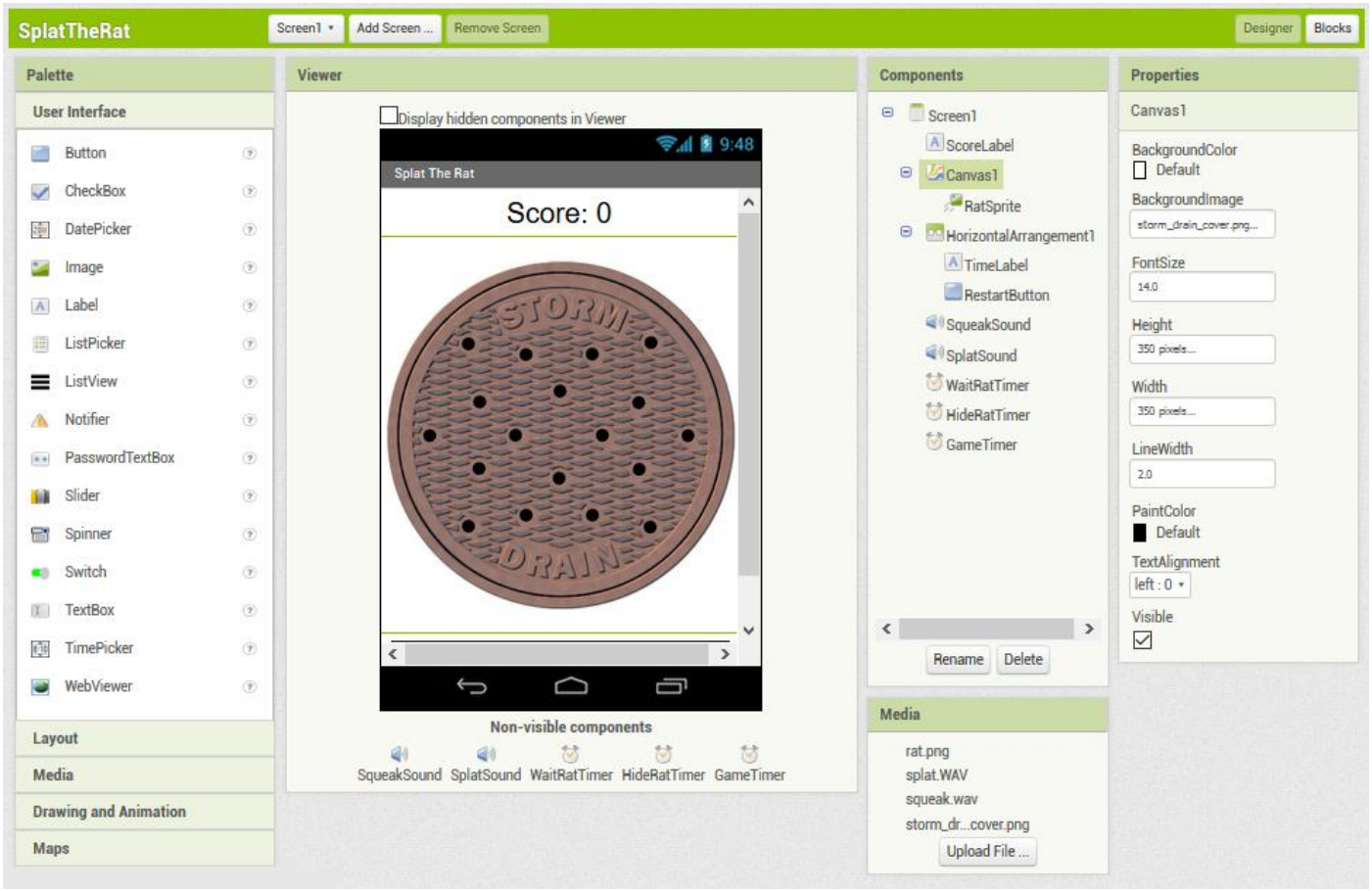
Ahora vamos a diseñar la pantalla de la app.

5.10. DISEÑAR LOS COMPONENTES DE "SPLAT THE RAT".

La tabla desglosa todos los componentes que necesitamos para construir la interfaz de usuario de esta app, junto con los valores que debemos fijar para sus propiedades. La figura que le sigue muestra el diseño de la pantalla.

Splat the Rat			
Screen1 properties	`Title`: Splat The Rat `AlignHorizontal`: Center `BackgroundColor`: White		
Components	**What do I rename it?**	**What does it do?**	**What properties do I set?**
`Label`	`ScoreLabel`	Shows the game score	`Text`: "Score: 0" `FontSize`: 28
`Label`	`TimeLabel`	Shows the time remaining	`Text`: "Time Remaining: 30" `FontSize`: 28
`Button`	`RestartButton`	Restarts the game	`Text`: Restart
`Horizontal-Arrangement`	`HorizArr`	Contains `TimeLabel` and `RestartButton` at the bottom of the screen	None
2 `Sounds`	`SqueakSound` `SplatSound`	Play a "squeak" sound when the rat appears and a "splat" sound when you touch it	`Source`: squeak.wav and splat.wav (one for each `Sound`)
`Canvas` Palette group: Drawing and Animation	`Canvas1`	The game play area	`BackgroundImage`: storm_drain_cover.png `Width` and `Height`: 350 pixels

Components	What do I rename it?	What does it do?	What properties do I set?
`ImageSprite` Palette group: Drawing and Animation	`RatSprite`	Displays an image of a rat	`Picture`: rat.png `Rotates`: Yes (selected) `Visible`: No (deselected) `Width`: 75 pixels `Height`: 50 pixels
`Clock` Palette group: User Interface	`WaitRatTimer`	Counts down a random time between 1 and 5 seconds; when it fires, the rat appears.	`TimerAlwaysFires`: Yes (selected) `TimerEnabled`: Yes (selected) `TimerInterval`: 5000
`Clock` Palette group: User Interface	`HideRatTimer`	Starts when the rat appears, counts down 2 seconds, and then hides the rat	`TimerAlwaysFires`: Yes (selected) `TimerEnabled`: No (unselected) `TimerInterval`: 2000
`Clock` Palette group: User Interface	`GameTimer`	Fires once a second; when it fires, it reduces the time remaining by 1 second and checks to see whether the game is over	`TimerAlwaysFires`: Yes (selected) `TimerEnabled`: Yes (selected) `TimerInterval`: 1000
Media files			
2 sound files downloaded from our website: splat.wav and squeak.wav 2 image files downloaded from our website: rat.png and storm_drain_cover.png	 		



Para empezar podemos ubicar la figura animada "RatSprite" en cualquier lugar sobre la alcantarilla. Cuando empiece el juego, la rata se moverá a lo largo de la pantalla.

Ahora, puede que nos preguntemos por qué necesitamos tres temporizadores. De hecho, hay muchas formas distintas de programar este videojuego. Tal vez pensemos que otra opción sería utilizar un solo temporizador y un bloque "if – then - else". La lógica que aplicaría esta versión alternativa es la siguente: Al dispararse el temporizador, la app debería:

- Fijar el intervalo del temporizador a un número aleaorio (por ejemplo, entre 1 y 10 segundos).
- Si la rata está visible, la escondemos. Si la rata está escondida, la mostramos.

Este método tiene sentido, pero la mecánica del juego ya no sería la correcta, porque no tendría final: La rata seguiría apareciendo y desapareciendo para siempre. Además, la rata también estaría la misma cantidad de tiempo mostrada y escondida, y en nuestro caso, queremos que permanezca escondida más tiempo, y que sólo aparezca por pantalla en pequeñas ráfagas, para que el juego sea más difícil.

Para resolver estos problemas hemos considerado que lo mejor es usar tres temporizadores, que funcionarían como indicamos:

1) Para asegurarnos de que el juego termina, el temporizador "GameTimer" se dispara una vez por segundo. Cuando se haya disparado 30 veces (es decir, tras 30 segundos), el juego termina.
2) Para hacer que la rata aparezca y se esconda durante cantidades de tiempo distintas, usaremos los temporizadores "WaitRatTimer" y "HideRatTimer". Estos temporizadores deben operar como un equipo de relevos:

   - "WaitRatTimer" controla cuándo puede aparecer la rata (pero no durante cuánto tiempo). Se dispara a intervalos de tiempo aleatorios entre 1 y 5 segundos, muestra la rata, y le cede el control a "HideRatTimer".

- "HideRatTimer" le permite a la rata aparecer durante únicamente durante 2 segundos. A continuación esconde a a rata y le vuelve a pasar el control a "WaitRatTimer".

NOTA: Temporizadores que arrancan otros temporizadores.

Hacer que un temporizador active otro temporizador es un truco muy útil en multitud de apps. Pero esta técnica puede producir efectos indeseados cuando un temporizador rápido (la liebre) se encarga de activar un temporizador más lento (la tortuga). Esto es lo que ocurre si la liebre (hare) se dispara a cada segundo y la tortuga (tortoise) solo se dispara cada dos segundos:

Time	0s	1s	2s	3s
Action	Hare timer starts	Hare fires and starts the tortoise timer	Hare fires and *restarts* the tortoise timer	Hare fires and *restarts* the tortoise timer

Esto seguirá ocurriendo todo el tiempo, y el resultado será que el temporizador de la tortuga nunca se dispara. Este error haría que el programa no funcionase. Podemos tener problemas similares si una parte de nuestra app tarda más tiempo en ejecutarse que el temporizador que la activa. En ese caso ocurrirían cosas raras, como por ejemplo, que solo funcione una parte de la app. ¿Cómo podemos corregir este error? Cuando la liebre se dispara, se deshabilita a sí misma, y activa a la tortuga. A continuación, cuando la tortuga ya ha terminado, se deshabilita a sí misma y activa a la liebre. Esta nueva secuencia seguiría repitiéndose, con la liebre y la tortuga trabajando conjuntamente:

Time	0s	1s	2s	3s
Action	Hare timer starts	Hare fires, disables itself, and starts the tortoise timer		Tortoise fires, disables itself, and starts the hare timer

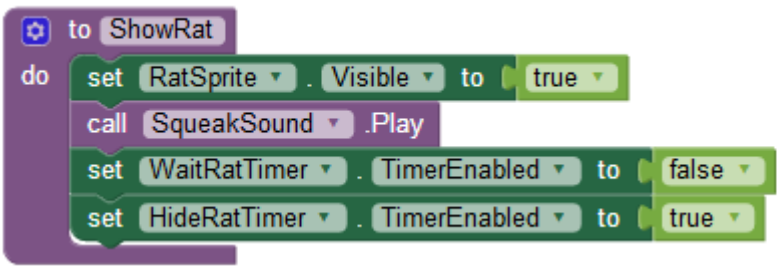
5.11. AÑADIR COMPORTAMIENTOS A LOS COMPONENTES DE "SPLAT THE RAT".

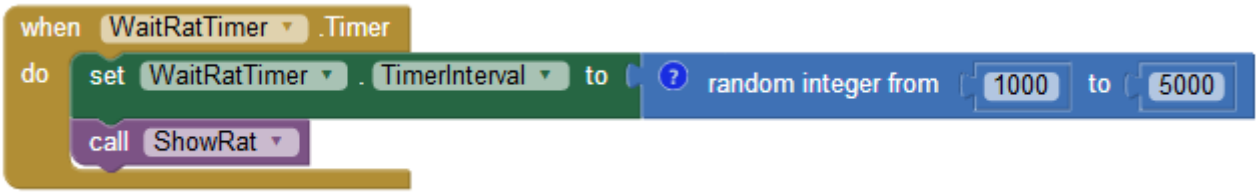
MOSTRAR Y ESCONDER A LA RATA.

Cuando "WaitRatTimer" se dispare por primera vez después de 5 segundos, queremos que elija un nuevo intervalo, de forma que el comportamiento de la rata sea impredecible. Podemos programar esta funcionalidad de forma muy similar a como lo hicimos en la app "WhereAreYouHiding". A continuación debemos mostrar a la rata durante dos segundos, y reproducir el sonido *squeak.wav*. Todo esto implica programar las siguientes acciones:

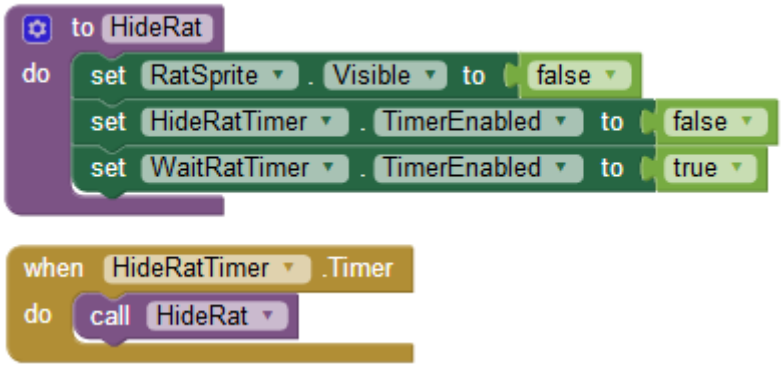
1) Mostrar la figura animada "RatSprite".
2) Reproducir un sonido *sqeak.wav*.
3) Deshabilitar el temporizador "WaitRatTimer".
4) Habilitar el temporizador "HideRatTimer".

El código es el siguiente:





Llegados a este punto la rata está en la pantalla y chillando. Sin embargo, queremos que la rata solo esté en pantalla durante dos segundos. Por consiguiente, tan pronto como "HideRatTimer" se dispare vamos a revertir todo lo que hemos hecho antes con el código que mostramos a continuación:



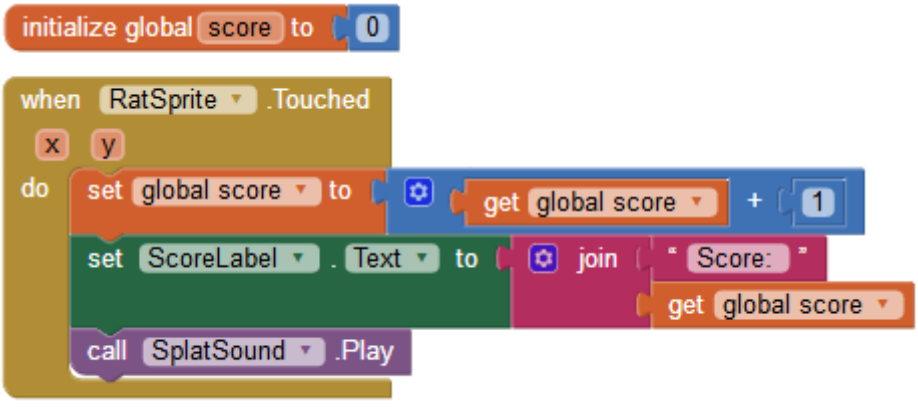
Aquí escondemos a la rata, y "WaitRatTimer" vuelve a iniciar una cuenta atrás durante una cantidad de tiempo aleatoria antes de volver a reiniciar todo el proceso de volver a mostrar a la rata.

Esta app siempre muestra a la rata en pantalla durante 2 segundos, por lo que no debemos cambiar su propiedad "TimerInterval". Pero una vez la tengamos plenamente operativa, podemos probar a ajustar un tiempo en pantalla aleatorio mediante el temporizador "WaitRatTimer", para ver cómo afecta esto al desarrollo del juego.

Vamos a probar la aplicación tal y como está ahora mismo: Lanzamos las pruebas en vivo en nuestro dispositivo o en el emulador, y deberíamos ver que la rata aparece y desaparece de la pantalla. A continuación, vamos a programar la acción de aplastarla.

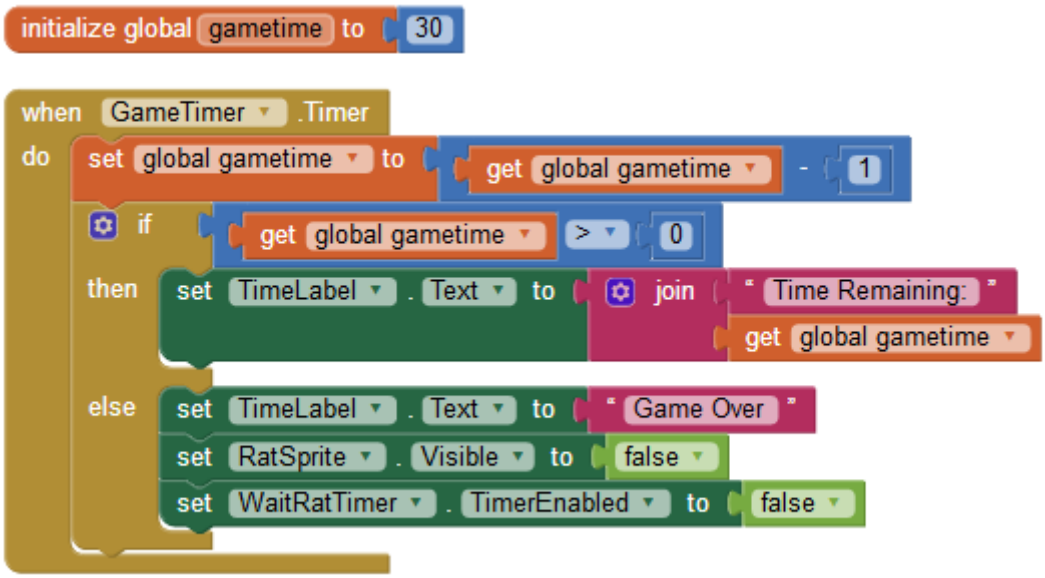
APLASTAR LA RATA Y CAMBIAR LA PUNTUACIÓN.

Necesitamos una variable para almacenar la puntuación, que se inicializa a cero. Podemos detectar si el usuario toca la rata con el evento "(RatSprite).Touched". Al activarse este evento, hemos de incrementar la puntuación en una unidad, cambiar esta puntuación en pantalla, y reproducir un sonido de aplastamiento (*splat.wav*).



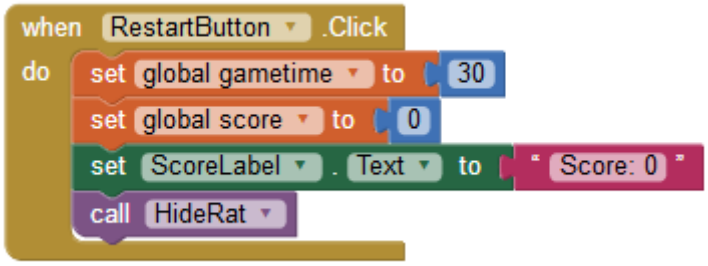
TEMPORIZAR, TERMINAR, Y REINICIAR EL JUEGO.

El código que mostramos a continuación se encarga de disminuir el tiempo de la partida segundo a segundo (partiendo de 30 segundos), y de terminar la partida cuando finaliza esta temporización. Como veremos, estos comportamientos se impementan con variables y un bloque "if – then - else", conceptos ya cubiertos en el capítulo previo. La única novedad aquí es que ejecutamos estas acciones con el temporizador "GameTimer", que se dispara una vez por segundo.



Si probamos la aplicación tal y como está ahora veremos que el temporizador con la cuenta atrás funciona, que al aplastar la rata nuestra puntuación se incrementa, y que el juego finaliza acabada la temporización de 30 segundos.

Empezar una nueva partida es muy fácil: Al pulsar el botón "RestartButton" debemos reinicializar la variable "GameTime" a 30 segundos y la variable "Score" a 0 puntos. Además, hemos de esconder la rata y retornar los temporizadores a sus estados iniciales (para ello, ya tenemos un procedimieto llamado "HideRat"; esto es lo bueno de los procedimientos). El programa es el siguiente:



El juego ya está casi terminado. Pero si lo probamos, veremos que no es muy divertido: La rata aparece y desaparece en intervalos de tiempo aleatorios, pero no se mueve, por lo que es muy fácil aplastarla. Vamos a hacer el juego más interesante haciendo que la rata se mueva.

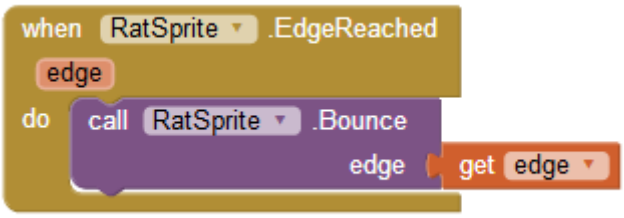
MOVER A LA RATA.

En el capítulo 3 aprendimos a mover una figura animada (concretamente, un componente "Ball"). Aquí haremos exactamente lo mismo dándole al objeto "RatSprite" unos ciertos valores a sus propiedades "Heading" y "Speed".

Para ello, volvemos al Diseñador de App Inventor, y fijamos los siguientes valores para las propiedades de "RatSprite":

- "Heading": 25.
- "Interval": 100.
- "Speed": 20.

Esto hará que la rata empiece corriendo en una cierta dirección, pero al llegar al borde del lienzo sobre el que se mueve, la rata se parará. Podemos hacer que la rata rebote al llegar al borde añadiendo el programa que aquí mostramos. Notar que el bloque "get (edge)" lo hemos obtenido pasando el ratón sobre el argumento "edge" del bloque manejador de evento. En el capítulo sobre animaciones explicaremos en mayor detalle todos estos bloques y propiedades.



5.12. AMPLIACIONES A LA APP "SPLAT THE RAT".

He aquí algunas ideas para modificar, mejorar, o ampliar la app "SplatTheRat":

- Podemos intentar hacer la rata más grande, más pequeña, más rápida, o más lenta, cambiando sus propiedades en el Diseñador. También podríamos cambiarlas de forma aleatoria cada vez que se dispare un temporizador. Si usamos el temporizador "GameTimer", la rata cambiaría de tamaño y de velocidad a cada segundo.
- Cuando estudiemos el capítulo de animaciones, podremos añadir una animación a la rata para que cambie su imagen a una rata aplastada durante 1 segundo si el jugador consigue tocarla.
- Otra opción es hacer varias ratas, y proporcionar puntos adicionales si el usuario aplasta más de una rata. Incluso podríamos añadir una especie de super rata que solo aparezca brevemente muy de vez en cuando, y cuyo aplastamiento proporcione muchos puntos extra.

5.13. EJERCICIOS DEL CAPÍTULO 5.

Ejercicio 5.1. Programa un temporizador de cocina: El usuario inserta el tiempo de la temporización, y el temporizador pita cuando el tiempo programado se ha agotado.

PISTA 1: Ten en cuenta que los temporizadores de cocina, como el de un horno, reciben un tiempo en minutos, pero el parámetro "TimerInterval" del componente "Clock" necesita un tiempo en milisegundos, por lo que necesitarás realizar una conversión entre unidades temporales (de minutos a minisegundos).

PISTA 2: Cuidado, porque los temporizadores de cocina no deben arrancar automáticamente, como ocurre con los temporizadores de app inventor cuando iniciamos la app. Este programa debería incluir algún componente para fijar la temporización en minutos, y un botón para arrancarla.

NOTA: La interfaz de usuario es libre: Diseña y construye una interfaz atractiva e intuitiva. (Por ejemplo, la selección del tiempo de temporización se podría implementar con un deslizador entre 0 y 60 minutos, con un valor preestablecido de 1 minuto). Se valorará positivamente la originalidad en el diseño.

PISTA 3: Los temporizadores de cocina solo suenan una vez, al terminar la temporización. Por el contrario, los temporizadores de App Inventor vuelven a iniciar otra temporización tras terminar la temporización

previa. Obviamente, nosotros queremos que la alarma pite una sola vez, al terminar la temporización. No queremos que el timbre suene una y otra cada vez que se agote el tiempo de temporización fijado.

6. MOVIMIENTO Y ANIMACIÓN DE SPRITES.

En este capítulo trataremos los métodos que ofrece App Inventor para crear apps con animaciones sencillas (esto es, con objetos que se mueven). Aprenderemos a crear sencillos videojuegos bidimensionales y nos familiarizaremos con el uso de las figuras animadas y con el manejo de eventos como las colisiones entre sprites.

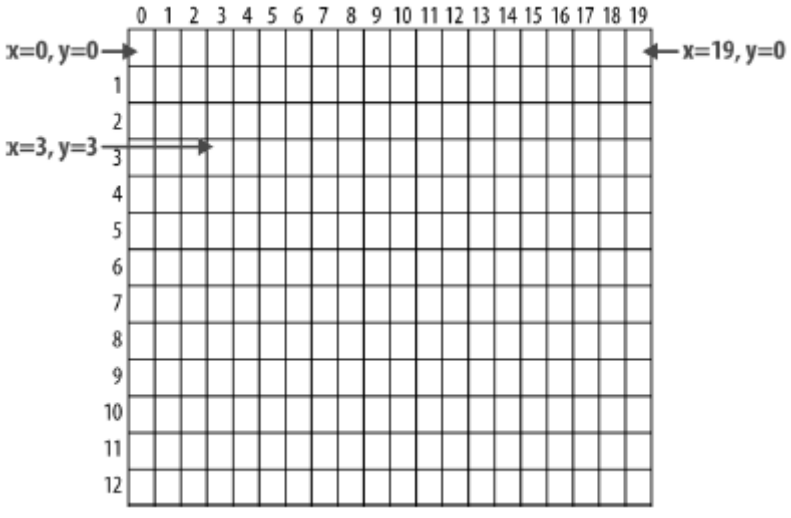
Cuando vemos un objeto moviéndose suavemente en la pantalla de un ordenador, lo que realmente estamos viendo es una rápida sucesión de imágenes en las que el objeto está en una posición ligeramente distinta cada vez. Se trata de una ilusión de movimiento no muy distinta a la de los libros animados o folioscopios, en los que percibimos que una figura se mueve cuando pasamos rápidamente las páginas.

En App Inventor podemos programar animaciones ubicando componentes "Ball" e "ImageSprite" dentro de un componente "Canvas", y moviendo o transformando dichos objetos en sucesivas fracciones de segundo. En este capítulo usaremos el evento "Clock.Timer" para activar los sucesivos movimentos de un objeto. También aprenderemos a mover un objeto actuando sobre las propiedades que determinan la rapidez, la dirección, y la frecuencia con la que se mueve. Por último veremos cómo mover un objeto permitiendo que sea el usuario quien lo controle mediante botones, arrastrándolo con su dedo, o inclinando el teléfono móvil. Pero si no queremos que parezca que el personaje está congelado o flotando también deberíamos animar sus piernas, cambiando la imagen del personaje a cada fracción de segundo.

6.1. AÑADIR UN COMPONENTE "CANVAS" A NUESTRA APP.

Para poder animar objetos, necesitamos añadir un componente "Canvas" a nuestra app. Después de agregarlo, especificamos sus propiedades "Width" y "Height". A menudo querremos que la anchura del lienzo se extienda a lo largo de toda la pantalla del dispositivo, y para ello, debemos fijar su propiedad "Width" al valor "Fill parent". Podríamos hacer exactamente lo mismo para la propiedad "Height", pero generalmente será preferible fijarla a algún número (por ejemplo, 300 píxeles) para dejar espacio para otros componentes que pudieran estar encima o debajo del lienzo.

Un dibujo en un lienzo es realmente una matriz de píxeles, donde un píxel es el punto de color más pequeño que se puede mostrar por pantalla. La localización de cada píxel está definida por un sistema de coordenadas x e y similar al mostrado en la figura. En este sistema coordenado, x define una localización en una línea horizontal (comenzando con el 0 en el extremo izquierdo e incrementándose conforme nos movemos hacia la derecha a lo largo de la pantalla), e y define una localización en una línea vertical (comenzando con el 0 en el extremo superior e incrementándose conforme nos movemos hacia *abajo* a lo largo de la pantalla).



La posición en la esquina superior izquierda tiene un valor de 0 para ambas cooordenadas, y esta posción se representa como $(x, y) = (0, 0)$, o simplemente $(0, 0)$ para abreviar. Al movernos hacia la derecha se incrementa la coordenada x, y al movernos hacia abajo se incrementa a coordenada y. Así, la posición inmediatamente a la derecha de la esquina superior izquierda tiene unas cordenadas $(1, 0)$. La esquina superior derecha tiene una coordenada x igual a la anchura del "Canvas" menos 1 unidad. La mayoría de los teléfonos móviles tiene una anchura de unos 300 píxeles, pero en el ejemplo mostrado en la figura, las coordenadas de la esquina superior derecha serían $(19, 0)$.

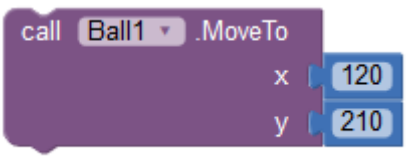
Podemos cambiar la apariencia de un lienzo de dos formas: Pintando sobre él, o ubicando objetos móviles dentro de él. Este capítulo se centra principalmente en la segunda posibilidad (la primera ya se discutió sobradamente en el capítulo 3). Como ya hemos mencionado antes, App Inventor nos permite ubicar sobre un lienzo los componentes "Ball" e "ImageSprite". Un **sprite** es un objeto gráfico localizado dentro de un escenario de mayor tamaño (en App inventor, este "escenario" es el componente "Canvas"). Tanto el componente "Ball" como el componente "ImageSprite" son sprites; la única diferencia entre ellos es su apariencia. Un componente "Ball" es un círculo cuya apariencia solo puede modificarse cambiando su color o su radio, mientras que un "ImageSprite" puede tomar una apariencia cualquiera, que viene definida por un archivo de imagen. Los componentes "Ball" e "ImageSprite" solo pueden existir dentro de un componente "Canvas"; no podemos ubicarlos en la interfaz de usuario fuera de un lienzo.

6.2. MOVER SPRITES USANDO EVENTOS TEMPORIZADORES.

Una forma de programar una animación en App Inventor es cambiar la posición o la apariencia de un objeto en respuesta a un evento temporizador. Habitualmente querremos mover sprites a diferentes localizaciones del lienzo en intervalos de tiempo fijos. Una forma de definir estos intervalos temporales es usar eventos temporizadores ("Clock.Timer"). En la siguiente sección discutiremos un método alternativo de programar animaciones usando las propiedades "Heading", "Interval", y "Speed" de los componentes "Ball" e "ImageSprite".

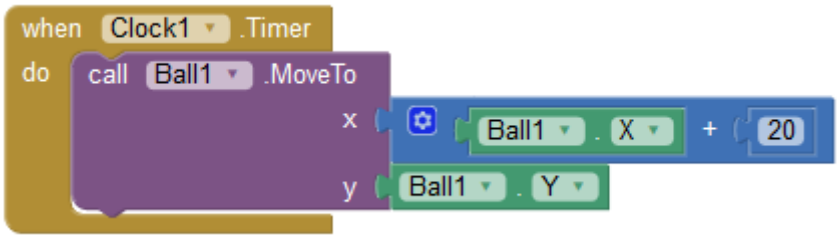
Las pulsaciones de botón y otros eventos iniciados por el usuario son fáciles de entender: El usuario hace algo, y la app responde realizando ciertas operaciones. Pero los eventos de temporizador son distintos porque no los activa el usuario, sino el transcurso del tiempo. Para definir un evento temporizador primero debemos acudir al Diseñador y añadir un componente "Clock" (bandeja "Sensors") a la app. El componente "Clock" tiene una propiedad llamada "TimerInterval" cuyo valor se especifica en milisegundos. Si fijamos un valor de 500, App Inventor lanzará un evento "Clock.Timer" cada vez que pasen 500 milisegundos (esto es, medio segundo) en el reloj. Así, cuanto más pequeño sea el valor del intervalo, más rápido será el ritmo de la animación.

Después de añadir un componente "Clock" y fijarle un valor a su propiedad "TimerInterval", debemos añadir el manejador de evento "when Clock.Timer" en el Editor de Bloques. Dentro de este manejador podemos incluir los bloques que queramos, y serán ejecutados a cada intervalo. Sin embargo, para hacer que un sprite se mueva a lo largo del tiempo, podemos usar la función "MoveTo" de los componentes "Ball" y/o "ImageSrite".



La función "MoveTo" mueve un objeto a una posición *absoluta* en el lienzo, y no en una cantidad relativa a su posición previa. Por ejemplo, el bloque de la figura (dentro del manejador correspondiente) movería a la pelota "Ball1" a la posición $(120, 210)$, independientemente de su localización previa. Para mover a un sprite en una cantidad relativa, debemos darle a los parámetros "x" e "y" de "MoveTo" las coordenadas actuales x

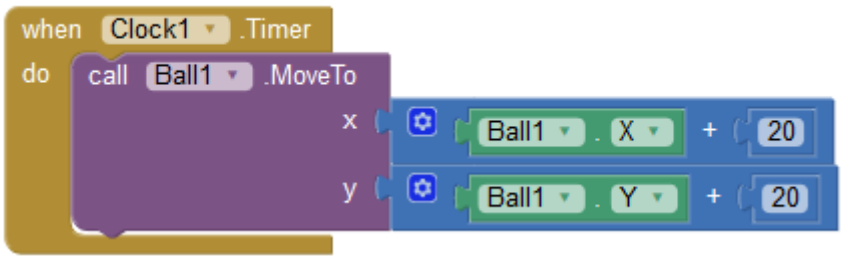
e y del sprite, más el desplazamiento deseado. Por ejemplo, para mover una pelota *horizontalmente* a lo largo de la pantalla, podríamos usar los bloques mostrados en la figura:



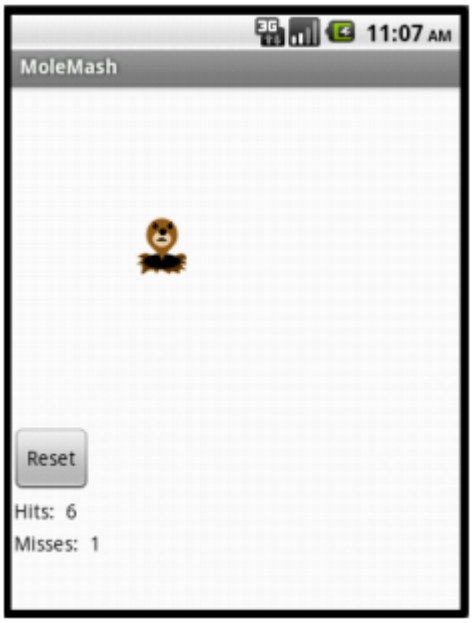
¿Cómo funcionaría este código? Imaginar que hemos fijado la propiedad "TimerInterval" del componente "Clock1" a 1000 (milisegundos). Cada 1000 milisegundos (esto es, cada segundo) se disparará el evento "Clock1.Timer", y App Inventor moverá la pelota a una nueva posición que está 20 píxeles a la derecha de su posición previa.

La rapidez del movimiento resultante depende tanto del valor que fijamos a la propiedad "TimerInterval" como de los valores que le proporcionemos a los parámetros "x" e "y" del bloque "MoveTo". Como hemos fijado "TimerInterval" a 1000 milisegundos, y la pelota se desplaza hacia la derecha 20 píxeles a cada intervalo, la pelota se moverá con una rapidez de $20\ px/1000\ ms = 20$ píxeles por segundo. Por ejemplo, si la pelota comenzase en la posición $(20, 10)$, tras 1 segundo se moverá a la posición $(40, 10)$, tras otro segundo se irá a la posición $(60, 10)$, y así sucesivamente… Pero un "TimerInterval" de 1 segundo no proporcina una animación demasiado suave: La pelota solo se desplaza una vez por segundo, y el movimiento parecerá entrecortado. Si queremos lograr un movimiento más suave deberíamos fijar un valor más pequeño para "TimerInterval". Por ejemplo, fijando un valor de 100 (milisegundos), la pelota se moverá 20 píxeles a la derecha cada décima de segundo, esto es, a una rapidez de $20\ px/100\ ms = 20\ px/0{,}1\ s = 200$ píxeles por segundo.

Ahora, ¿Y si quisiéramos mover la pelota verticalmente hacia abajo? Para ello, deberíamos modificar únicamente su coordenada y, dejando la coordenada x fija. Y si quisiéramos mover la pelota diagonalmente, deberíamos actuar tanto sobre la coordenada x como sobre la coordenada y, como muestra la figura:



Para poner en práctica estos conceptos, vamos a programar un videojuego inspirado en el juego clásico "Whac-A-Mole", en el que unos topos mecánicos salen de unos agujeros, y los jugadores suman puntos cuando los golpean con un mazo.



6.3. COMENZAR CON LA APP "MOLEMASH".

Nuestra app, a la que llamaremos "MoleMash", funcionará de la siguiente forma: Un topo ("mole") aparece en localizaciones aleatorias en la pantalla, cambiando de posición después de cada segundo. Si el usuario toca con el dedo al topo, el dispositivo vibra, el número de aciertos aumenta en una unidad, y el topo se mueve inmediatamente a una nueva localización. Si el usuario toca la pantalla

pero no toca al topo se incrementa en una unidad el número de fallos. Un botón de "Reset" reinicia los contadores de aciertos y de fallos.

Para comenzar el proyecto nos conectamos a App Inventor y seleccionamos "Projects" → "Start new project". Lo llamamos "MoleMash", y cambiamos el título de la pantalla a "MoleMash". Pinchamos en "Connect" y conectamos nuestro dispositivo o el emulador para hacer pruebas en vivo. Para esta app necesitaremos la imagen del topo, *mole.png*, que encontraremos en los archivos de los alumnos.

6.4. DISEÑAR LOS COMPONENTES DE "MOLEMASH".

Para construir la app "MoleMash" usaremos estos componentes:

- Un componente "Canvas" que actuará como terreno de juego.
- Un componente "ImageSprite" que mostrará la imagen de un topo, y que podrá moverse por el lienzo y detectar cuando le tocan.
- Un componente "Sound" que vibra cuando el jugador toca al topo.
- Sendas etiquetas "Label" que muestren los aciertos ("Hits") y los fallos ("Misses"), junto con el número actual de aciertos y fallos.
- Unos organizadores de tipo "HorizontalArragement" para posicionar correctamente las etiquetas.
- Un componente "Button" para reiniciar a cero el número de aciertos y fallos.
- Un componente "Clock" para hacer que el topo se mueva cada segundo.

La tabla lista todos los componentes que usaremos para la app:

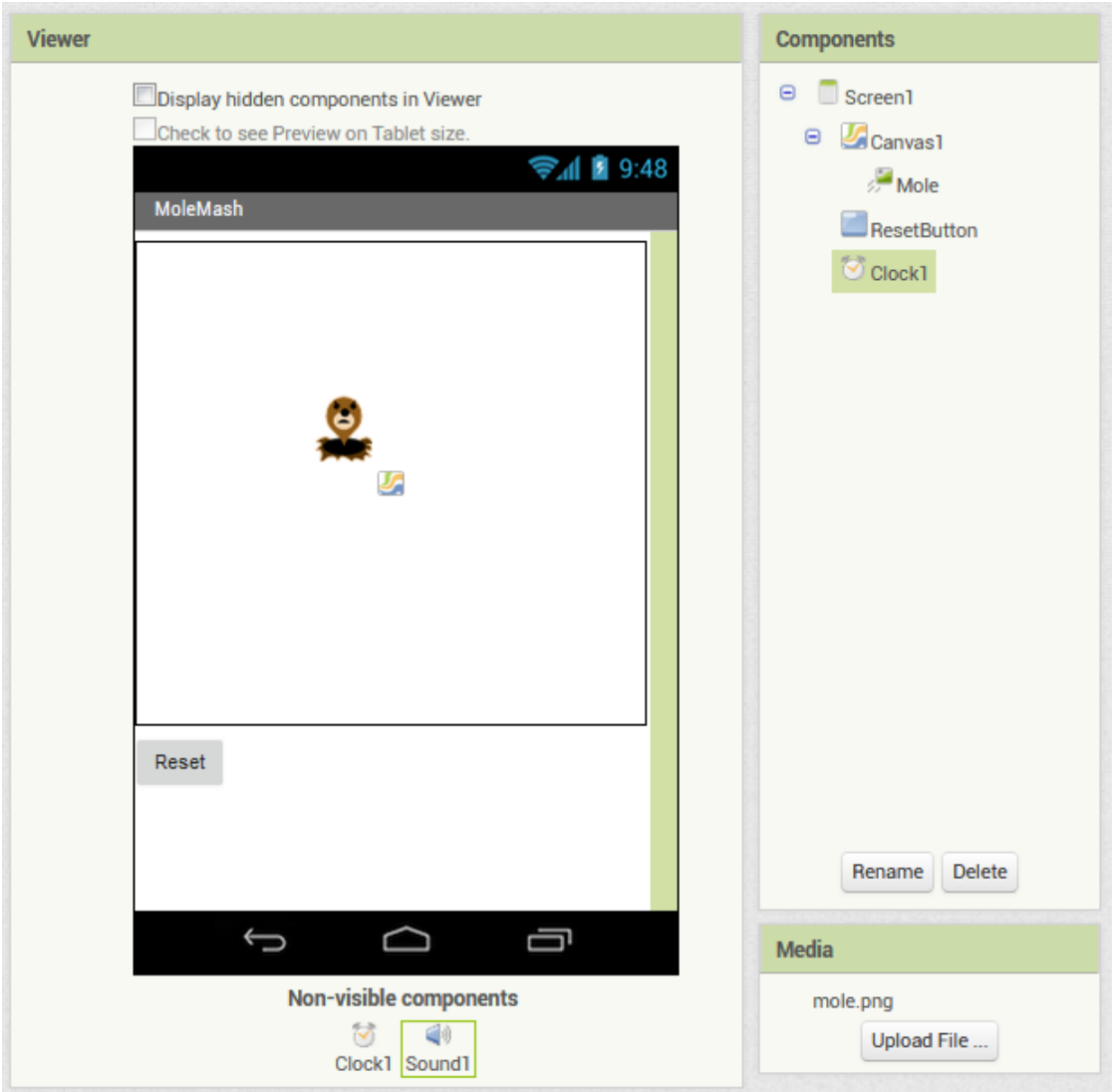
Component type	Palette group	What you'll name it	Purpose
`Canvas`	Drawing and Animation	`Canvas1`	The container for `ImageSprite`.
`ImageSprite`	Drawing and Animation	`Mole`	The user will try to touch this.
`Button`	User Interface	`ResetButton`	The user will press this to reset the score.
`Clock`	User Interface	`Clock1`	Control the mole's movement.
`Sound`	Media	`Sound1`	Vibrate when the mole is touched.
`Label`	User Interface	`HitsLabel`	Display "Hits: ".
`Label`	User Interface	`HitsCountLabel`	Display the number of hits.
`HorizontalArrangement`	Layout	`Horizontal Arrangement1`	Position `HitsLabel` next to `HitsCountLabel`.
`Label`	User Interface	`MissesLabel`	Display "Misses: ".
`Label`	User Interface	`MissesCountLabel`	Display the number of misses.
`HorizontalArrangement`	Layout	`Horizontal Arrangement2`	Position `MissesLabel` next to `MissesCountLabel`.

AÑADIR LOS COMPONENTES DE ACCIÓN.

En esta sección añadiremos los componentes necesarios para que se desarrolle la acción del videojuego. En la siguiente sección añadiremos los componentes para llevar la cuenta de la puntuación y para mostrarla:

1) De la bandeja "Drawing and Animation" sacamos un componente "Canvas" y lo soltamos en el Visor. Lo dejamos con su nombre por defecto, "Canvas1". Fijamos su propiedad "Width" a "Fill Parent", para que su anchura sea igual a la de la pantalla, y ajustamos su propiedad "Height" a 300 píxeles.
2) De nuevo, de la bandeja "Drawing and Animation" sacamos un componente "ImageSprite", y lo ubicamos dentro del componente "Canvas1". En la parte inferior de la lista de componentes cambiamos su nombre a "Mole". A continuación, fijamos su propiedad "Picture" al archivo *mole.png*, esto es, al archivo de imagen que cargamos previamente al proyecto.
3) De la bandeja "User Interface" arrastramos un componente "Button" hasta dejarlo bajo "Canvas1". Cambiamos su nombre a "ResetButton" y fijamos su propiedad "Text" a "Reset".
4) De la bandeja "Sensors" sacamos un componente "Clock". Al dejarlo sobre el Visor, aparecerá en la sección de componentes no visibles.
5) De la bandeja "Media" sacamos un componente "Sound" y los soltamos en el Visor. Este componente también aparecerá en la sección de componentes no visibles.

La pantalla de nuestra app debería parecerse a la de la figura. (Puede que nuestro topo esté en una posición distinta sobre el lienzo).

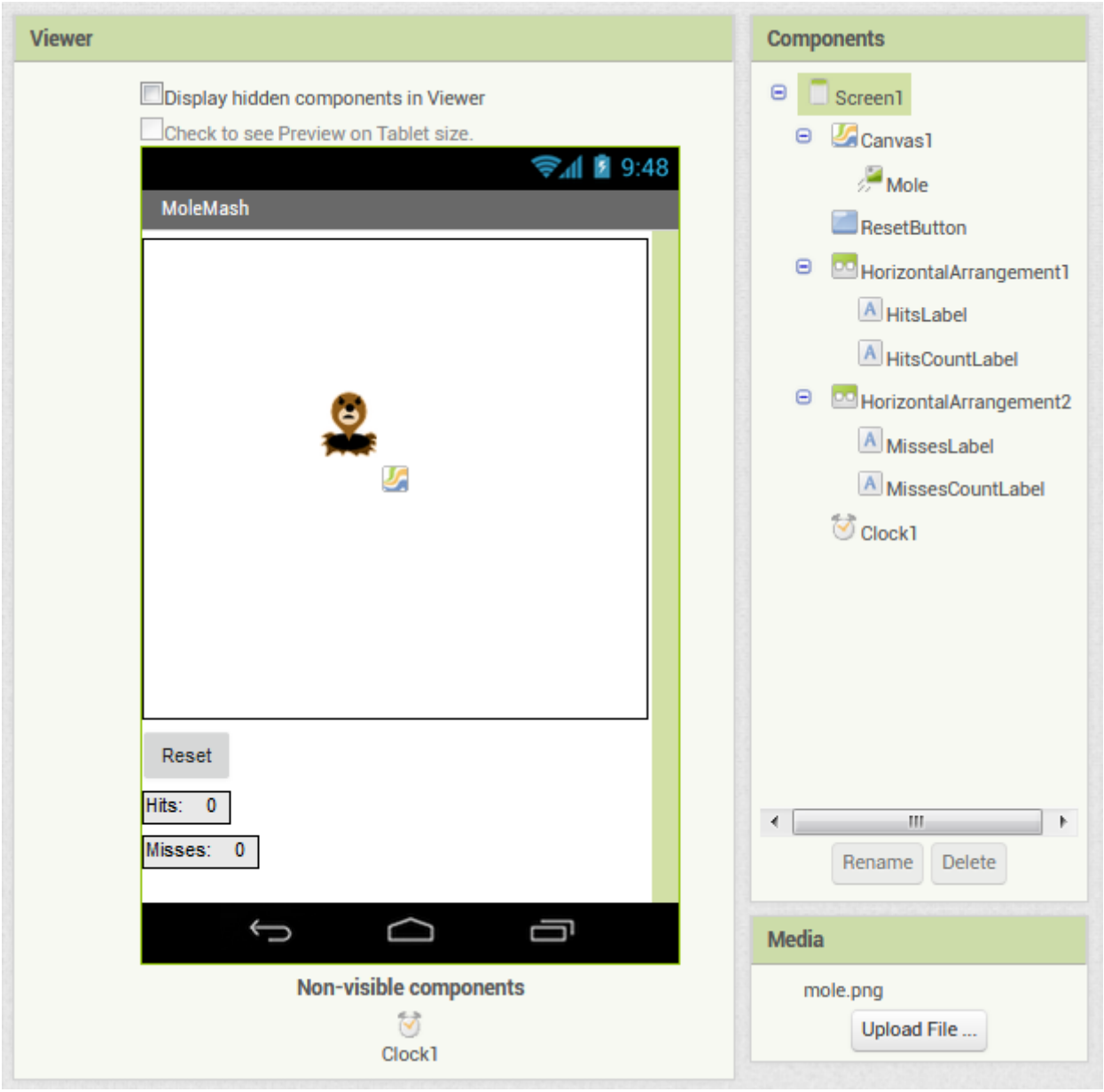


AÑADIR LOS COMPONENTES DE PUNTUACIÓN.

Ahora vamos a añadir los componentes para mostrar la puntuación del usuario, específicamente, el número de aciertos y de fallos.

1) De la bandeja "Layout", tomamos un componente "HorizontalArrangement" y lo colocamos bajo el botón de reseteo. Mantenemos su nombre por defecto a "HorizontalArrangement1".
2) De la bandeja "User Interface" sacamos dos etiquetas y las soltamos dentro del organizador horizontal.
   - Llamamos a la etiqueta de la izquierda "HitsLabel", y fijamos su propiedad "Text" a "Hits: ". (Notar que hemos puesto dos puntos, seguidos de un espacio en blanco).
   - Llamamos a la etiqueta de la derecha "HitsCountLabel", y fijamos su propiedad "Text" al número 0.
3) Sacamos un segundo organizador horizontal, y lo colocamos bajo el primero.
4) Arrastramos dos etiquetas a este segundo organizador:
   - Llamamos a la etiqueta de la izquierda "MissesLabel", y fijamos su propiedad "Text" a "Misses: ". (De nuevo, notar que hemos puesto dos puntos, seguidos de un espacio en blanco).
   - Llamamos a la etiqueta de la derecha "MissesCountLabel", y fijamos su propiedad "Text" al número 0.

La pantalla debería parecerse a la mostrada en la figura:



6.5. AÑADIR COMPORTAMIENTOS A LOS COMPONENTES.

Después de añadir los componentes nos vamos al Editor de Bloques para definir su comportamiento. En concreto queremos que, tras cada segundo, el topo se mueva a una posición aleatoria en el lienzo. El objetivo del usuario es tocar al topo allá donde aparezca, y la app mostrará el número de veces que el usuario cierta o falla el golpe. El botón de reseteo reinicia el número de aciertos y de fallos a cero.

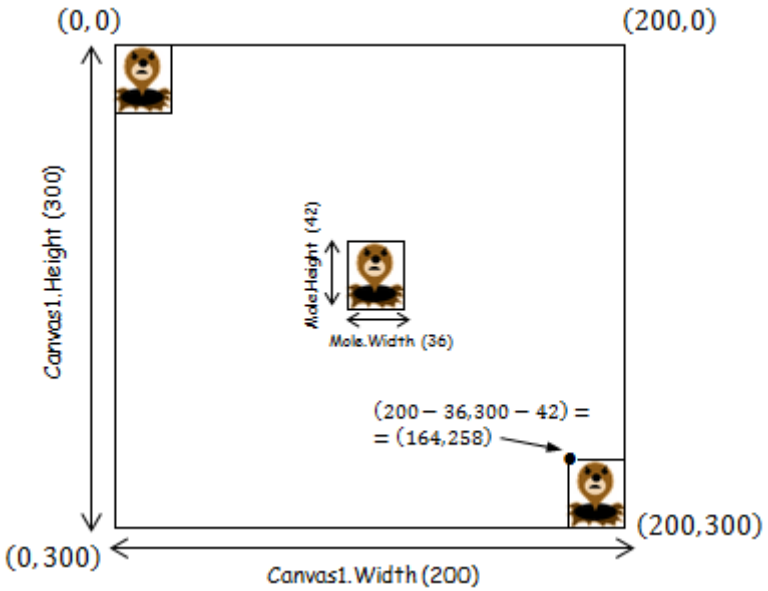
MOVER EL OBJETO.

En los programas que hemos escrito hasta ahora hemos llamado a funciones preconstruidas en App Inventor, como por ejemplo, las funciones "Sound.Vibrate" y "Canvas.DrawCircle". Sería estupendo que App Inventor también proporcionase una función para mover un componente "ImageSprite" a una localización aleatoria en la pantalla. Desgraciadamente, App Inventor no incluye esta función. Por suerte, sí que nos permite crear nuestros propios **procedimientos**. Al igual que las funciones preconstruidas de App Inventor, nuestros procedimientos también aparecerán en una bandeja, y podremos usarlos exactamente igual que las funciones.

En esta app necesitaremos crear un procedimiento para mover al topo a una posición aleatoria en pantalla. A este procedimiento lo denominaremos "MoveMole". Llamaremos a este procedimiento al principio del juego, cada vez que el usuario toque al topo, y también una vez por segundo.

CREAR EL PROCEDIMIENTO "MOVEMOLE".

Como ya sabemos, el lienzo es como una matriz de píxeles con unas coordenadas x (horizontal) e y (vertical), donde las coordenadas (x, y) de la esquina superior izquierda son $(0, 0)$. (Es decir, el origen de coordenadas se sitúa en la esquina superior izquierda del lienzo). La coordenada x se incrementa conforme nos movemos hacia la derecha, y la coordenada y se incrementa conforme nos movemos hacia abajo, como muestra la figura. Las propiedades "X" e "Y" de un componente "ImageSprite" indican dónde está ubicado su extremo superior izquierdo. Por consiguiente, el topo en la esquina superior izquierda de la figura tiene unos valores de "X" e "Y" iguales a 0.



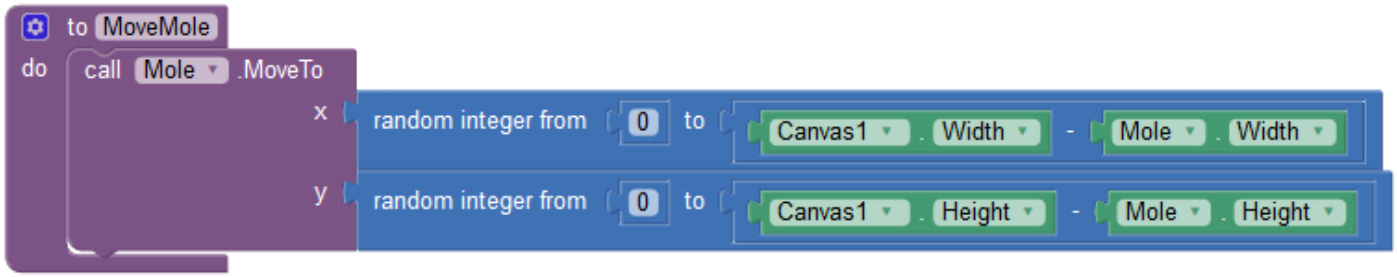
Para determinar los valores máximos para x e y de la nueva localización del topo, de forma que siempre quepa en la pantalla, debemos hacer uso de las propiedades "Width" (anchura) y "Height" (altura) de los componentes "Mole" y "Canvas1". (La anchura y la altura del topo vienen dadas por el tamaño horizontal y vertical de la imagen "mole.png" que cargamos en el proyecto. Cuando creamos el componente "Canvas1" fijamos su altura a 300 píxeles, y su anchura a "Fill parent", lo cual copia la anchura de su componente

padre, que es este caso es la pantalla). Si el topo tiene una anchura de 36 píxeles y el lienzo de 200 píxeles, la coordenada x del lado izquierdo del topo puede variar desde 0 (completamente a la izquierda) hasta 164 (esto es, $200 - 36$, o bien, Canvas1.Width − Mole.Width). Con ello nos aseguramos que la imagen del topo nunca se saldrá por el extremo derecho de la pantalla. De forma similar, la coordenada y del extremo superior del topo puede variar desde 0 hasta Canvas1.Height − Mole.Height.

Para ubicar al topo en una posición aleatoria, hemos de elegir al azar una coordenada x dentro del rango desde 0 hasta Canvas1.Width − Mole.Width. De forma análoga, hemos de elegir al azar una coordenada y en el rango desde 0 hasta Canvas1.Height − Mole.Height. Podemos generar un número entero aleatorio mediante el bloque "random integer from () to ()" de la bandeja "Math".

Para crear el procedimiento "MoveMole" hacemos lo siguiente:

1) En el Editor de Bloques, pinchamos en la bandeja "Procedure".
2) Arrastramos el bloque "to (procedure)", pero el que incluye la sección "do", y no el de la sección "result".
3) En este bloque, pinchamos en el texto "procedure" y escribimos "MoveMole" para establecer el nombre del procedimiento.
4) Como queremos mover el topo pinchamos en la bandeja "Mole", y arrastramos el bloque "call (Mole).MoveTo" a la sección "do" del procedimiento. Notar que los conectores abiertos en el lado derecho de la función "(Mole).MoveTo" nos indican que debemos proporcionar las coordenadas x e y de la nueva posición a la que queremos llevar al topo.
5) Para especificar que la nueva coordenada x del topo debería ser un valor aleatorio entre 0 y Canvas1.Width − Mole.Width hacemos lo siguiente:
   - De la bandeja "Math" sacamos el bloque "random integer from () to ()", y lo pegamos al conector "x" del bloque "call (Mole).MoveTo".
   - Cambiamos a 0 el valor del bloque numérico dentro del conector "from" del bloque "random integer from () to ()".
   - Borramos el bloque numérico del 100 dentro del conector "to" del bloque "random integer from () to ()".
   - Sacamos un bloque de resta de la bandeja "Math", y lo ponemos dentro del conector "to".
   - De la bandeja "Canvas1" sacamos el bloque "(Canvas1).(Width)" y lo ponemos en el conector de la izquierda del bloque de resta.
   - Análogamente, de la bandeja "Mole" sacamos el bloque "(Mole).(Width)" y lo ponemos en el conector de la derecha del bloque de resta.
6) Seguimos un procedimiento similar para especificar que la coordenada y debería ser un entero aleatorio en el rango desde 0 hasta Canvas1.Height − Mole.Height. El código debería quedar como muestra la figura:

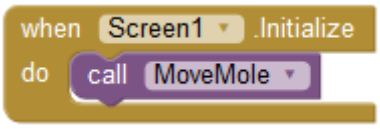


LLAMAR AL PROCEDIMIENTO "MOVEMOLE".

Llamar al procedimiento al comenzar la app.

Ahora que hemos escrito el procedimiento "MoveMole", vamos a usarlo. Como queremos llamar al procedimiento "MoveMole" nada más arrancar la app, recurrimos al manejador "when (Screen1).Initialize" y hacemos lo siguiente:

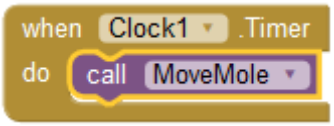
1) Pinchamos en la bandeja "Screen1" y sacamos el bloque "when (Screen1).Initialize".
2) Vamos a la bandeja "Procedures", donde ahora veremos un bloque "call (MoveMole)", que es el bloque para llamar al procedimiento que hemos creado antes. Ponemos este bloque dentro del manejador de evento "when (Screen1).Initialize", como muestra la figura.



Llamar al procedimiento tras cada segundo.

Para hacer que el topo se mueva tras cada segundo necesitamos un componente "Clock". En el Diseñador dejamos la propiedad "TimerInterval" de "Clock1" a su valor por defecto de 1000 (milisegundos), o 1 segundo. Esto significa que, cualquier cosa que especifiquemos dentro del manejador de evento "when (Clock1).Timer", ocurrirá tras cada segundo. Así es como configuramos el programa:

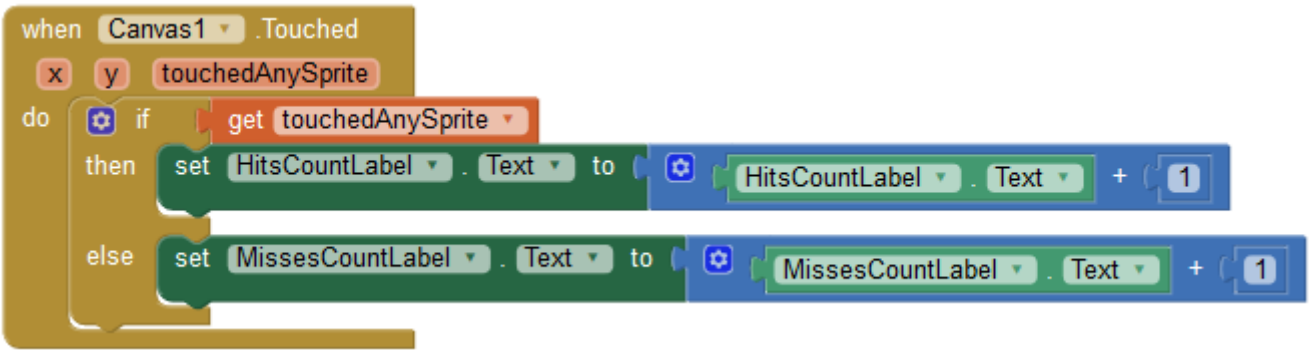
1) De la bandeja "Clock1" sacamos manejador de evento "when (Clock1).Timer".
2) De la bandeja "Procedures" sacamos un bloque "call (MoveMole)", y lo ponemos dentro del bloque "when (Clock1).Timer", como muestra la figura.



Si este tiempo de 1 segundo es muy rápido o muy lento para nuestro gusto, podemos cambiar la propiedad "TimeInterval" de "Clock1" en el Diseñador, de forma que el topo se mueva con mayor o menor frecuencia.

LLEVAR LA PUNTUACIÓN.

Recordar que hemos creado dos etiquetas, llamadas "HitsCountLabel" y "MissesCountLabel", que inicialmente tenían usos valores de 0. Queremos que estos valores se incrementen siempre que el usuario golpee con su dedo al topo (un acierto), y cuando toque la pantalla sin alcanzar al topo (un fallo), respectivamente. Para hacerlo, vamos a usar el manejador de evento "when (Canvas1).Touched", el cual indica que el usuario ha tocado en el lienzo, las coordenadas x e y donde ha tocado (algo que no necesitamos saber en esta app), y si ha tocado a algún objeto en el lienzo (algo que sí debemos saber en esta app). La figura muestra el código que vamos a escribir:



El código de la figura dice lo siguiente: Siempre que el usuario toque el lienzo "Canvas1", comprobamos si ha tocado un objeto. Como solo hay un objeto en nuestro programa, el objeto tocado debe ser "Mole". Si el usuario ha tocado a "Mole", sumamos 1 a la propiedad "HitsCountLabel.Text"; en caso contrario, sumamos 1 a la propiedad "MissesCountLabel.Text".Así es como se crea este programa:

1) Pinchamos en la bandeja "Canvas1" y sacamos el manejador de evento "when (Canvas1).Touched".

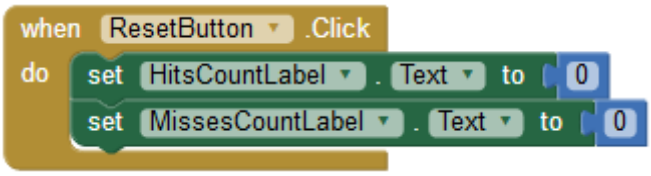
2) Vamos a la bandeja "Control" y sacamos un bloque "if () then ()". Pinchamos en el icono azul del engranaje para acceder a la configuración del bloque, y añadimos una rama "else ()". A continuación, ponemos este bloque dentro del manejador de evento "when (Canvas1).Touched".
3) Pasamos el ratón sobre el argumento "touchedAnySprite" del bloque "when (Canvas1).Touched", y sacamos un bloque "get (touchedAnySprite)". Conectamos este bloque "get ()" al conector de condición del bloque "if () then () else ()"
4) Como queremos que "HitsCountLabel.Text" se incremente en una unidad si la condición es cierta (esto es, si el usuario ha tocado al topo), hacemos lo siguiente:
   - De la bandeja "HitsCountLabel" sacamos el bloque "set (HitsCountLabel).Text to", y lo conectamos a la derecha del "then".
   - De la bandeja "Math" sacamos un bloque de suma, y lo ubicamos en el conector "to" de "set (HitsCountLabel) (Text) to".
   - De la bandeja "HitsCountLabel" sacamos un bloque "HitsCountLabel.Text" y lo insertamos en el conector de la izquierda del bloque de suma.
   - De la bandeja "Math" sacamos un bloque numérico, cambiamos su valor a 1, y lo insertamos en el conector de la derecha de bloque de suma.
5) Repetimos el paso 4 para "MissesCountLabel" en la sección "else" del bloque "if () then () else()"

Ahora probamos nuestro código en el dispositivo. Para ello tocamos sobre el lienzo, a veces acertando en el golpeo del topo, y otras veces fallando. Deberíamos ver cómo cambia la puntuación de ambas etiquetas.

RESETEAR LA PUNTUACIÓN.

Si un amigo nos ve jugar con el "MoleMash", probablemente querrá probarlo. Por lo tanto, necesitamos poder resetear el número de aciertos y fallos a 0. A estas alturas ya deberíamos ser capaces de añadir esta funcionalidad sin tener que seguir las instrucciones que vienen a continuación. Intenta hacerlo antes de seguir leyendo.

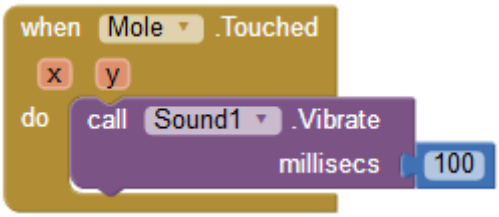
Lo que necesitamos es un manejador de evento "when (ResetButton).Click" que fije los valores de "HitsCountLabel.Text" y "MissesCountLabel.Text" a 0. Para ello, creamos el código mostrado en la figura:



Llegados a este punto probablemente ya no necesitamos las instrucciones paso a paso para construir este código. Solo recordar que en vez de obtener el bloque numérico con el 0 de la bandeja "Math", podemos acelerar el proceso simplemente escribiendo un cero, y el bloque debería crearse automáticamente.

HACER QUE EL DISPOSITIVO VIBRE AL TOCAR EL OBJETO.

Ya hemos comentado antes que queremos que el dispositivo vibre cuando el usuario consiga tocar al topo. Esto podemos hacerlo con el bloque "call (Sound1).Vibrate" de la bandeja de "Sound1".



Para terminar probamos la aplicación. Debemos observar que al presionar el botón de reseteo, las puntuaciones vuelven a 0. Además, cuando conseguimos golpear al topo, el dispositivo debería vibrar 100 milisegundos. (Si la vibración nos parece muy breve, podemos prolongarla cambiando el valor del bloque numérico a 150, 200, 250, … milisegundos).

6.6. AMPLIACIONES A LA APP "MOLEMASH".

Aquí presentamos algunas ideas para mejorar nuestra app "MoleMash":

- Añadir botones para permitir que el topo se mueva más rápido o más lento.
- Añadir una etiqueta que lleve la cuenta y muestre el número de veces que ha aparecido (que se ha movido) el topo.
- Añadir un segundo componente "ImageSprite" con la imagen de algo que el usuario no debería tocar, como una trampa, o una bomba. Si el usuario lo toca, lo penalizamos reduciendo su puntuación o terminando el juego.
- En vez de usar la imagen de un topo, podemos dejar que sea el usuario el que seleccione una imagen con el componente "ImagePicker".

6.7. DETECCIÓN DE COLISIONES.

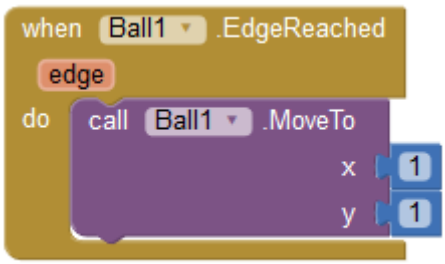
Al crear juegos y tras apps animadas generalmente necesitaremos funcionalidades más complejas que el simple movimiento. Afortunadamente, App Inventor proporciona algunos bloques de alto nivel para tratar con eventos de animación tales como objetos llegando al borde de la pantalla, u objetos colisionando unos contra otros.

Por "bloques de alto nivel" nos referimos a que es App Inventor quien se encarga de los detalles de bajo nivel a la hora de determinar si se produce un evento como una colisión. De no ser así, deberíamos comprobar manualmente si se producen este tipo de ocurrencias, monitorizando constantemente las propiedades de los sprites y del lienzo, lo cual requeriría una lógica bastante compleja. Por suerte, App Inventor proporciona bloques que ya implementan internamente esta lógica, y que podemos usar sin preocuparnos por los detalles de su funcionamiento.

EDGE REACHED.

Consideremos de nuevo la animación del segundo ejemplo de la sección 6.2, en la que una pelota se movía diagonalmente desde la parte superior izquierda del lienzo hacia la zona inferior derecha del lienzo. Tal y como está programada, la pelota se movería en diagonal y se detendría al llegar al borde derecho o inferior del lienzo (el sistema no mueve ningún sprite más allá de los límites del lienzo).

Si quisiéramos que la pelota reapareciera en la esquina superior izquierda cada vez que llegase al borde derecho o inferior del lienzo, deberíamos definir una respuesta al evento "Ball.EdgeReached" similar a la mostrada en la figura:



El evento "EdgeReached" se dispara cada vez que un sprite impacta contra cualquier borde de un lienzo. Este programa, junto con el programa que definía el movimiento diagonal, hacen que la pelota se mueva en diagonal desde arriba a la izquierda hacia abajo a la derecha, vuelva a la esquina superior izquierda, y repita este movimiento una y otra vez, para siempre (o hasta que la app le diga otra cosa).

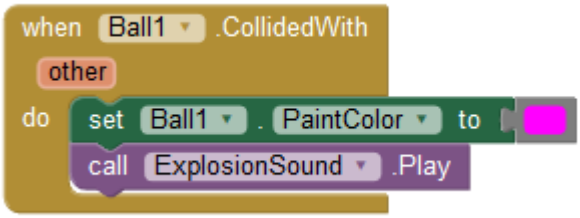
En este ejemplo no era necesario distinguir qué borde fue alcanzado. Pero el manejador "when (Ball).EdgeReached" incuye un argumento "edge" que reporta contra qué borde ha colisionado el sprite con el siguiente código numérico:

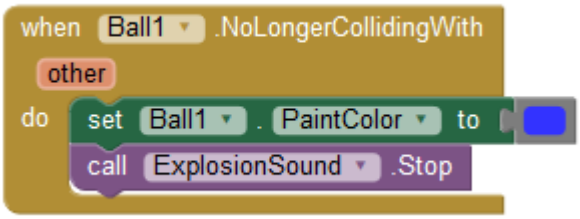
- Borde superior (o norte): 1.
- Borde inferior (o sur): -1.
- Borde derecho (o este): 3.
- Borde izquierdo (u oeste): -3.
- Borde noreste: 2.
- Borde suroeste: -2.
- Borde sureste: 4.
- Borde noroeste: -4.

Esto nos permite realizar diferentes acciones (mediante bloques "if") dependiendo del borde contra el que colisione el objeto en cuestión.

COLLINDING WITH Y NO LONGER COLLINDING WITH.

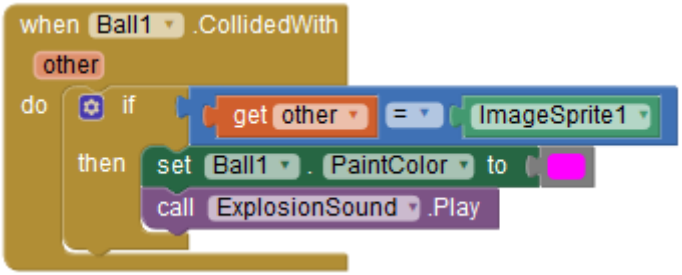
El desarrollo de un videojuego y de otras apps animadas suele depender de que dos o más sprites colisionen entre sí. A modo de ejemplo, consideremos un juego en el que un objeto cambia de color y reproduce un sonido de impacto cuando choca contra cualquier otro objeto. La figura de la derecha muestra el código necesario para implementar este comportamiento:





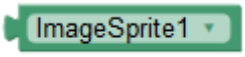
"NoLongerCollindingWith" es el evento opuesto a "CollidedWith". Este evento se dispara cuando dos objetos han colisionado y se han separado. Así, podríamos completar el código de este ejemplo como indica la figura de la izquierda.

Notar que los manejadores de los eventos "CollidedWith" y "NoLongerCollidingWith" tienen un argumento "other" que reporta contra qué objeto en particular hemos colisionado (o hemos dejado de colisionar). Esto nos permite realizar ciertas acciones solo cuando la pelota interaccione con otro objeto en particular, como muestra la figura:



El bloque "ImageSprite1" es uno que todavía no habíamos utilizado. Este bloque se refiere al componente como un todo, y no a una propiedad particular del componente. Cuando necesitemos comparar componentes (por ejemplo, para determinar qué objetos han colisionado), debemos usar este bloque. Todos los

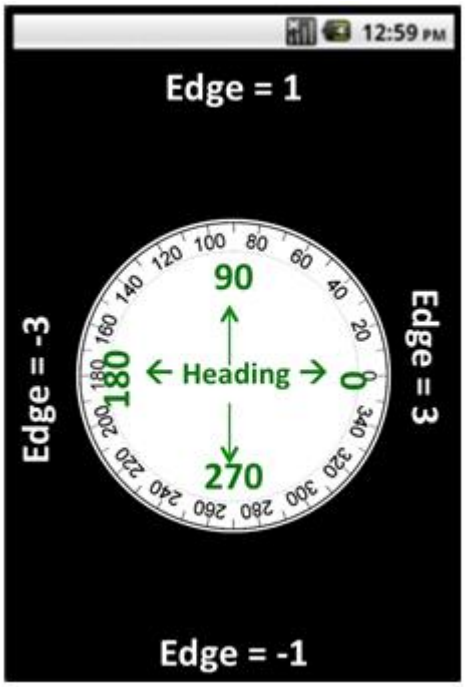
componentes tienen un bloque como éste en su propia bandeja, y el bloque tiene el mismo nombre que el componente.



6.8. MOVER SPRITES USANDO SUS PROPIEDADES.

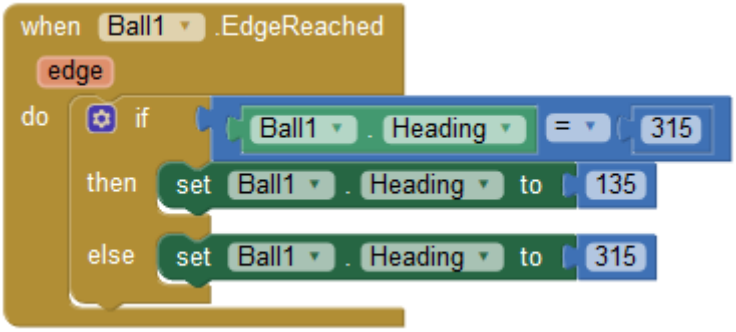
Las animaciones que hemos creado hasta ahora usaban un componente "Clock" para especificar que el objeto debía moverse cada vez que se disparaba el evento "Clock.Timer". Esta forma de mover un objeto es el método más general de implementar una animación. Además de simplemente mover un objeto, también podríamos haber usado esta técnica para modificar el color de un objeto con el tiempo, cambiar un texto (para que aparezca como si la app lo estuviese mecanografiando), para hacer que la app convierta a voz un cierto texto con un ritmo dado, etc.

Pero si lo único que queremos hacer es mover un objeto, App Inventor proporciona un enfoque alternativo que no requiere del uso de un componente "Clock". Tanto los componentes "Ball" como los componentes "ImageSprite" poseen unas propiedades llamadas "Heading", "Speed", e "Interval". En vez de usar un manejador de evento "when (Clock).Timer", podemos ajustar los valores de estas propiedades (en el Diseñador de Componente o en el Editor de Bloques) para controlar el movimiento del sprite.



La propiedad "Heading" sirve para especificar la dirección del movimiento del sprite, y puede ajustarse a un valor numérico cualquiera entre 0 y 360 grados. Si a "Heading" le fijamos un valor de 0, el sprite se moverá de izquierda a derecha. Si le fijamos un valor de 90, el sprite se moverá de abajo a arriba. Un valor de 180 moverá al sprite de derecha a izquierda. Y un valor de 270 moverá al sprite de arriba a abajo. Por supuesto, también podemos fijar un valor cualquiera entre 0 y 360: Por ejemplo, si fijamos "Heading" a 315, el sprite se moverá desde arriba a la izquierda hacia abajo a la derecha.

Para hacer que el sprite se mueva, también debemos fijar la propiedad "Speed" a un valor distinto de 0. En realidad, la rapidez del objeto viene determinada conjuntamente por las propiedades "Speed" e "Interval". La propiedad "Speed" es la distancia, en píxeles, que el objeto se moverá tras cada "Interval", en milisegundos. Por ejemplo, si la propiedad "Interval" está fijada a 100 milisegundos ($0{,}1$ segundos), y le damos a "Speed" un valor de 5 píxeles, el sprite se moverá 5 píxeles a cada $0{,}1$ segundos, esto es, $(5\ px/0{,}1\ s) = 50$ píxeles por segundo, lo que representa un movimiento suave y rápido.

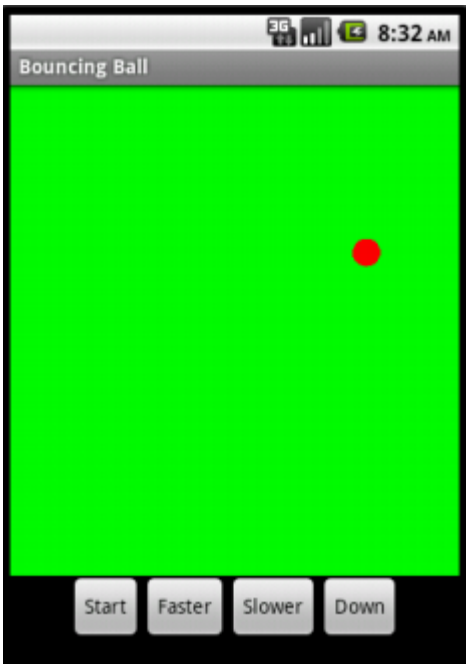


Por ejemplo, suponer que queremos mover una pelota ininterrumpidamente desde arriba a la izquierda hacia abajo a la derecha sobre un lienzo. En el Diseñador podríamos inicializar las propiedades "Speed" e "Interval" de la pelota a 5 y 100, respectivamente, y la propiedad "Heading" a 315. A continuación

añadiríamos al Editor de Bloques un manejador de eventos "when Ball1.EdgeReached" como el de la figura para cambiar la dirección de la pelota al llegar a un borde.

6.9. COMENZAR CON LA APP "BOUNCINGBALL".

A modo de ejemplo, vamos a construir una app en la que una pelota rebote constantemente al llegar al borde del lienzo en el que se mueve, y donde la app cambie el color del lienzo tras cada rebote. También permitiremos que el usuario controle la rapidez de la pelota, y que le permita reiniciar la animación. Por último, añadiremos un botón que obligará a la pelota a rebotar hacia abajo desde donde quiera que esté en ese momento.

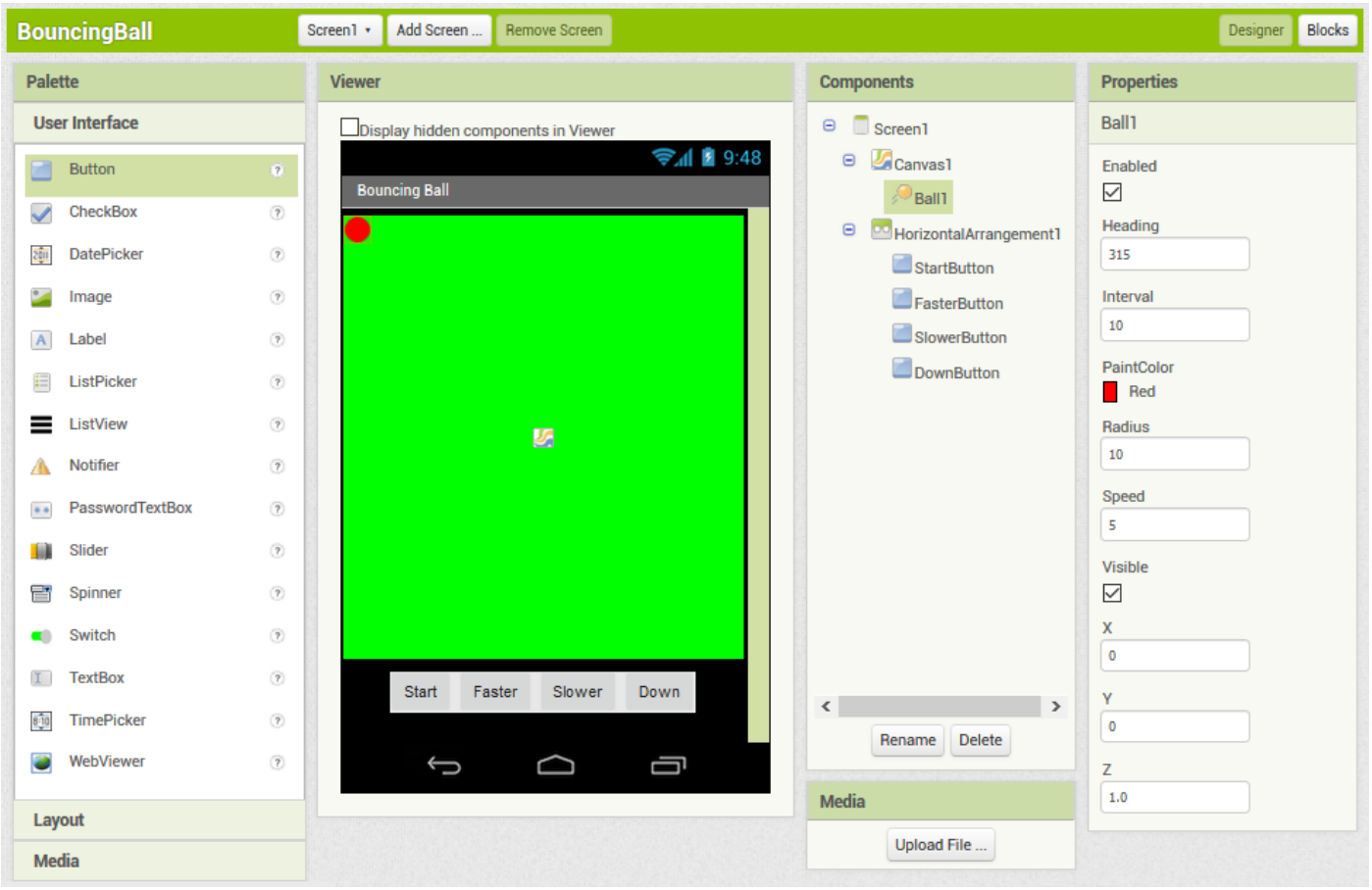


Nos conectamos a App Inventor y creamos un nuevo proyecto llamado "BouncingBall". A continuación, configuramos nuestro dispositivo o el emulador para hacer pruebas en vivo.

6.10. DISEÑAR LOS COMPONENTES DE "BOUNCINGBALL".

La figura muestra el Diseñador de Componentes para esta app.

Con esta captura de pantalla, y con los valores iniciales mostrados en la figura para las propiedades del componente "Ball1" (disponible en la bandeja "Drawing and Animation"), ya deberías ser capaz de crear una interfaz de usuario similar a la ilustrada.

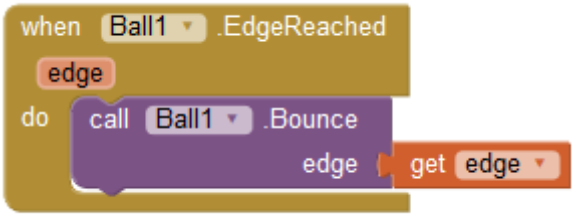


6.11. AÑADIR COMPORTAMIENTOS A LOS COMPONENTES DE "BOUNCINGBALL".

HACER REBOTAR A LA PELOTA.

Cuando los sprites chocan contra el borde del lienzo o colisionan contra otro sprite, lanzan el evento correspondiente. En nuestro caso, estamos interesados en el evento "EdgeReached", y el manejador que se encarga de capturar este evento y actuar en consecuencia es "when (Ball1).EdgeReached". Este manejador incluye un argumento llamado "edge" que almacena el borde contra el que se produjo la colisión.

Para esta app, queremos que la pelota rebote al llegar al borde del lienzo. Para hacer rebotar a la pelota, acudimos a la bandeja de bloques de "Ball1" y seleccionamos el bloque de función "call (Ball1).Bounce". Este bloque necesita que le especifiquemos contra qué borde debe rebotar (parámetro "edge"). Como queremos que rebote contra cualquier borde detectado por el manejador "when (Ball1).EdgeReached", obtenemos un bloque "get (edge)" del bloque manejador, y lo conectamos a la ranura abierta del parametro "edge" del bloque "call (Ball1).Bounce". El programa debería quedar como muestra la figura:

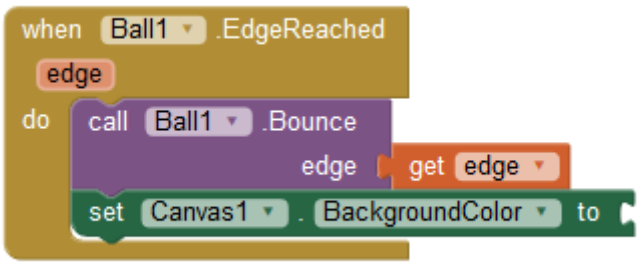


Vamos a probar la app. Nada más empezar, la pelota debería aparecer la posición $(x, y) = (0, 0)$ (esquina superior izquierda del lienzo), y empezar a moverse en la dirección $315°$ (hacia abajo y a la derecha) con la rapidez especificada. (Todos estos valores iniciales los hemos especificado en la lista de propiedades del componente "Ball1" en el Diseñador). Al llegar al borde del liezo, la pelota debería rebotar con un ángulo de $90°$ en la dirección opuesta.

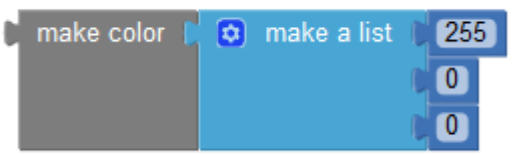
CAMBIAR EL COLOR DEL LIENZO TRAS CADA REBOTE.

Cuando la pelota llega al borde del lienzo, además de hacer rebotar a la pelota, queremos cambiar el color de fondo del lienzo a un color aleatorio. Para ello:

1) Añadimos el bloque "set (Canvas1).BackgroundColor" dentro del manejador "when (Ball1).EdgeReached".

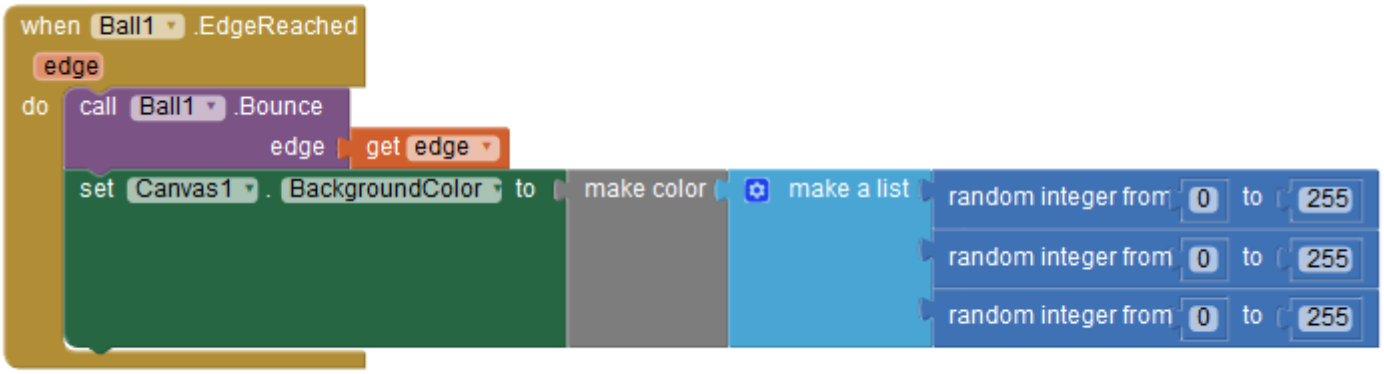


2) Ahora debemos espeficiar el color que le queremos fijar al fondo del lienzo. Para indicar que queremos un color aleatorio, acudimos a la bandeja "Colors", y extraemos el bloque "make color".



El bloque "make color" permite fabricar un color personalizado proporcionando los valores de sus componentes de rojo, verde, azul (y opcionalmente alfa). Los valores de estos componentes pueden especificarse mediante un número del 0 al 255 en la ranura correspondiente del bloque "make a list" adosado. Por consiguiente, para fabricar un color totalmente aleatorio, vamos a conectar a las tres ranuras del bloque "make a list" unos bloques "random integer from (0) to (255)".

3) Finalmente, conectamos el bloque "make color" al bloque "set (Canvas1).BackgroundColor" dentro del manejador "when (Ball1).EdgeReached". El programa debería quedar como muestra a figura:



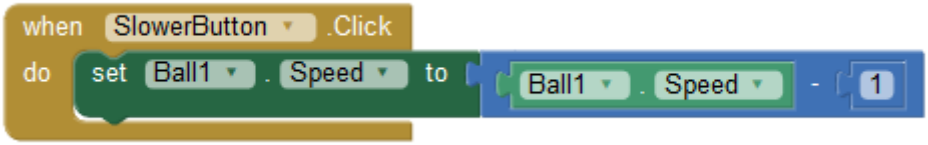
Si ahora probamos la app, comprobaremos que a cada rebote de la pelota, el fondo del lienzo cambia a un color aleatorio que, en general, no será uno de los colores básicos disponibles en App Inventor.

CAMBIAR LA RAPIDEZ DE LA PELOTA.

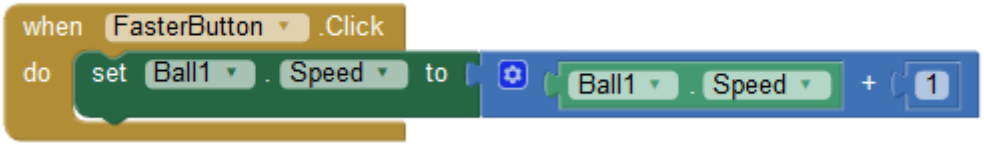
Vamos a comenzar programando el comportamiento del botón "SlowerButton", que le permitirá al usuario hacer que la pelota se mueva más despacio. Recordar que la ropiedad "Speed" permite determinar la cantidad de píxeles que recorre la pelota tras cada intervalo (propiedad "Interval"), por lo que tenemos dos formas de reducir la rapidez de la pelota:

1) Reduciendo el valor de su propiedad "Speed", o bien…
2) Aumentando el valor de su propiedad "Interval".

Aquí hemos optado por la primera opción: Cada vez que el usuario presione el botón "SlowerButton", vamos a restarle una unidad al valor actual de la porpiedad "Speed" de "Ball1". El programa para hacerlo es muy sencillo. Intenta escribirlo por ti mismo y cuando hayas acabado, comprueba que se ajusta al de la figura:



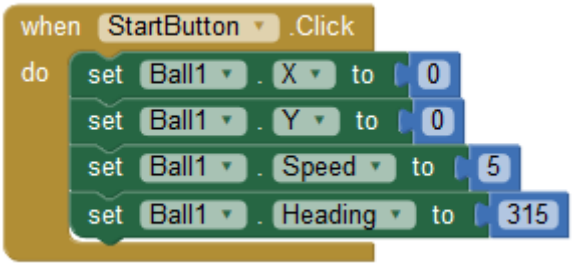
Para programar el comportamiento del botón "FasterButton" hacemos algo similar, solo que en este caso vamos a incrementar el valor de la propiedad "Speed" en una unidad cada vez que el usuario lo pulse.



Vamos a probar la aplicación. Cada vez que el usuario pulsa los botones "FasterButton" o "SlowerButton", la pelota debería pasar a moverse más rápido o más lento que antes, respectivamente.

REINICIAR EL MOVIMIENTO DE LA PELOTA.

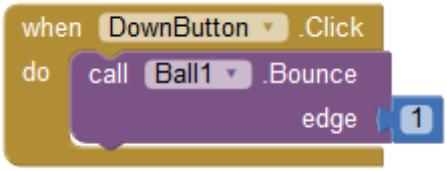
Cuando el usuario presione el botón "StartButton" queremos que la app reinicie la animación, llevando de vuelta a la pelota a su posición inicial $(x, y) = (0, 0)$, y volviendo a ajustar su movimiento a sus valores iniciales ("Speed" $= 5$ y "Heading"$= 315$). El programa es muy sencillo:



OBLIGAR A LA PELOTA A REBOTAR HACIA ABAJO.

Para terminar, vamos a programar el comportamiento del botón "DownButton". Cuando el usuario pulse este botón queremos que la pelota rebote hacia abajo esté donde esté, como si hubiese chocado contra un muro horizontal invisible que estuviese por encima de ella.

Para programar este comportamiento vamos a usar el bloque "call (Ball1).Bounce", pasándole al parámetro "edge" el valor 1. De esta forma, al ejecutar este bloque la pelota rebotará como si en ese momento hubiese colisionado contra un borde superior (esto es, hacia abajo). Como queremos que esto ocurra cuando el usuario presione el botón "DownButton", el programa será el siguiente:



Y con esto hemos terminado. Al ejecutar la app en nuestro dispositivo o en el emulador deberíamos comprobar que, al presionar el botón "DownButton", la pelota debería rebotar hacia abajo aún cuando no haya llegado al borde superior del lienzo. Además, al presionar el botón "StartButton", la pelota debería comenzar de nuevo su movimiento inicial, partiendo desde la esquina superior derecha, y moviéndose hacia abajo y hacia la derecha con una rapidez igual a la rapidez inicial.

6.12. MOVER SPRITES USANDO LA INTERACCIÓN DEL USUARIO.

En las animaciones que hemos creado hasta ahora el usuario apenas si estaba implicado. Pero los videojuegos son interactivos, y a menudo, el usuario es el encargado de controlar la rapidez o la dirección en la que se mueve un objeto.

En las siguientes secciones aprenderemos varias formas en las que el usuario puede controlar el movimiento de un objeto, ya sea deslizando su dedo sobre el sprite a lo largo de la pantalla táctil (aplicación "PoppingBalloons"), o inclinando el dispositivo móvil (aplicación "LadybugChase").

6.13. COMENZAR CON LA APP "POPPINGBALLOONS".

Vamos a construir una app en la que varios globos flotan en la parte superior de la pantalla, moviéndose de derecha a izquierda a distintas velocidades. El objetivo del juego es explotar todos los globos. Para ello, el usuario podrá arrastrar una flecha de derecha a izquierda en la parte infeior de la pantalla. Cuando el usuario deja de arrastrar la flecha, esta saldrá disparada hacia arriba para tratar de alcanzar a los globos.

Para esta app necesiaremos varios archivos multimedia: las imágenes *balloon1.png*, *balloon2.png*, *balloon3.png*, *balloonPopping.png*, y *arrow.png*, y el sonido *balloonPopping.mp3*.

6.14. DISEÑAR LOS COMPONENTES DE "POPPINGBALLOONS".

Para la pantalla de esta app necesitamos los siguientes componentes:

1) Un componente "Canvas". Para sus propiedades "Height" y "Width" fijaremos unos valores de 360 píxeles y "Fill parent", respectivamente.
2) Cuatro componentes "ImageSprite" (disponibles en la bandeja "Drawing and Animation"):
   - "BalloonSprite1": Para este sprite fijamos las siguientes propiedades:
     - "Picture": *balloon1.png*.
     - "Height" y "Width": fijamos unos valores de 50 y 25 píxeles, respectivamente.
     - "x" e "y": fijamos unos valores de 10 y 10, respectivamente, para que el globo comience arriba y a la izquierda en el lienzo.[8]
     - "Heading": fijamos un valor de 0, para que empiece a moverse hacia la derecha.
     - "Interval": 50.
     - "Speed": 5.
     - "Rotates": Desactivamos esta propiedad para que la imagen del globo no rote al cambiar su dirección (propiedad "Heading").
   - "BalloonSprite2": Para este sprite fijamos las siguientes propiedades:
     - "Picture": *balloon2.png*.
     - "Height" y "Width": fijamos unos valores de 50 y 25 píxeles, respectivamente.
     - "x" e "y": fijamos unos valores de 100 y 60, respectivamente, para que el globo comience centrado y un poco más abajo que "BalloonSprite1").
     - "Heading": fijamos un valor de 180, para que empiece a moverse hacia la izquierda.
     - "Interval": 50.
     - "Speed": 8.
     - "Rotates": Desactivada.
   - "BalloonSprite3": Para este sprite fijamos las siguientes propiedades:
     - "Picture": *balloon3.png*.
     - "Height" y "Width": fijamos unos valores de 50 y 25 píxeles, respectivamente.
     - "x" e "y": fijamos unos valores de 250 y 110, respectivamente, para que el globo comience un poco más abajo y a la derecha de "BalloonSprite2").
     - "Heading": fijamos un valor de 180, para que empiece a moverse hacia la izquierda.
     - "Interval": 50.
     - "Speed": 3.

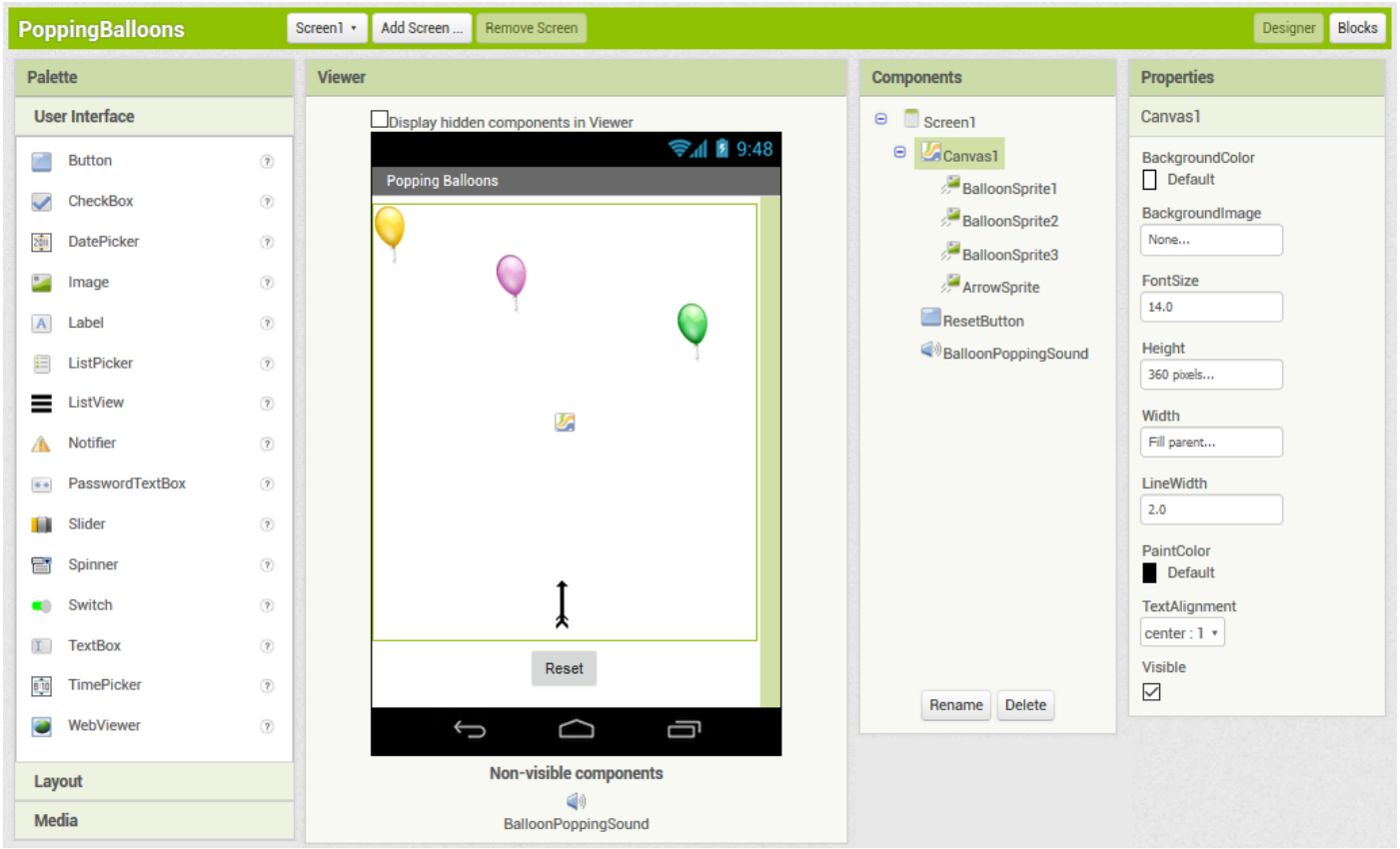
[8] Por alguna razón, si le fijamos una posición inicial de $(x, y) = (0, 0)$, este sprite se queda atascado en la esquina superior izquierda al iniciar su movimiento. Por ello hemos optado por ubicarlo inicialmente en $(x, y) = (10, 10)$.

  - "Rotates": Desactivada.
- "Arrow": Para este sprite fijamos las siguientes propiedades:
  - "Picture": *arrow.png*.
  - "Height" y "Width": fijamos unos valores de 40 y 10 píxeles, respectivamente.
  - "x" e "y": fijamos unos valores de 150 y 310, respectivamente, para que la flecha comience centrada en la parte inferior del lienzo).
  - "Heading": 0.
  - "Interval": 10.
  - "Speed": 10.
  - "Rotates": Desactivamos esta propiedad para que la imagen de la flecha no rote al cambiar su dirección (propiedad "Heading").

3) Un componente botón, al que llamaremos "ResetButton", y cuyo texto cambiaremos a "Reset".
4) Un componente de sonido, al que llamaremos "BalloonPoppingSound", y cuya propiedad "Source" fijaremos a *balloonPopping.mp3*.

NOTA: Los valores aquí indicados no son obligatorios. De hecho, probablemanete tengamos que ajustarlos para que la app funcione adecuandamente en nuestro dispositivo móvil.

Nuestro diseño de la interfaz de usuario debería ser similar al mostrado en la figura:

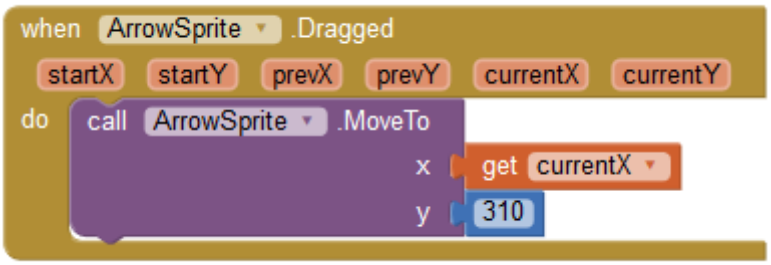


6.15. AÑADIR COMPORTAMIENTOS A LOS COMPONENTES DE "POPPINGBALLOONS".

ARRASTRAR LA FLECHA.

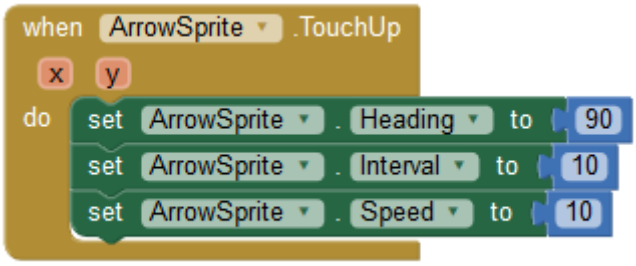
Recordemos que el usuario puede arrastrar la flecha horizontalmente a lo largo de la parte baja del lienzo. Para programar este comportamiento, sacamos el manejador de evento "when (ArrowSprite).Dragged" y el bloque "call (ArrowSprite).MoveTo" de la bandeja de "ArrowSprite". A continuación ubicamos el bloque "call (ArrowSprite).MoveTo" dentro de la sección "do" del manejador "when (ArrowSprite).Dragged".

Para el bloque "call (ArrowSprite).MoveTo" debemos especificar los valores de los parámetros "x" e "y", esto es, los valores de las coordenadas x e y de la nueva posición a la que queremos mover el sprite. Como queremos que el sprite se mueva horizontamente a la posición donde ha terminado el arrastre de nuestro dedo, fijamos el valor del parámetro "x" al valor del argumento "currentX" reportado por el manejador "when (ArrowSprite).Dragged". Y como no queremos que el sprite se mueva verticalmente, fijamos el valor del parámetro "y" a 310, esto es, el valor inicial de su coordenada vertical, que lo fuerza a permanecer en la parte baja del lienzo. El programa debería quedar como muestra la figura:



LANZAR LA FLECHA.

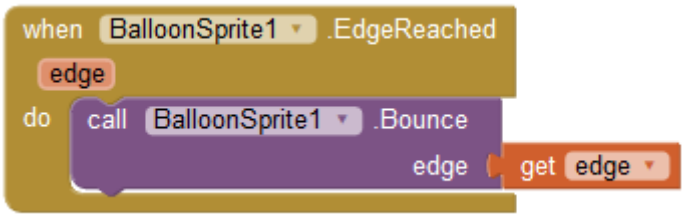
Cuando el usuario deja de arrastrar la flecha levantando el dedo de la pantalla, la flecha debe salir disparada hacia arriba desde la posición en la que está. El evento que se activa cuando el usuario deja de tocar un sprite es "TouchUp". El bloque manejador con el que gestionaremos este evento es "when (ArrowSprite).TouchUp". Este manejador reporta dos argumentos "x" e "y", que especifican las coordenadas x e y del punto donde el usuario levantó el dedo de la pantalla. Al activarse este evento, queremos fijar las propiedades "Heading", "Interval", y "Speed" de la flecha para que vuele velozmente hacia arriba desde la posición en la que el usuario deja de arrastrarla. Para ello, programamos el manejador como muestra la figura:



Vamos a probar la aplicación: Si delizamos nuestro dedo sobre la flecha a lo largo de la pantalla, la flecha debería deslizarse horizontalmente en la parte baja del lienzo. Al levantar nuestro dedo de la pantalla, la flecha debería volar velozmente hacia arriba desde la posición en la que levantamos el dedo.

MOVER LOS GLOBOS.

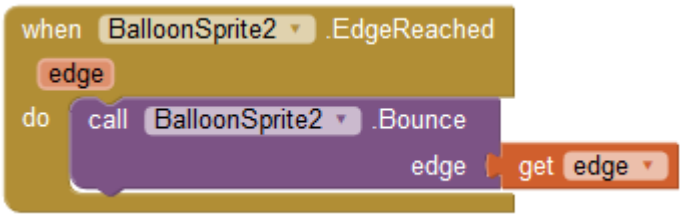
Con las configuraciones que realizamos en el Diseñador para sus propiedades "Heading", "Interval", y "Speed", los globos empiezan a moverse según los valores fijados, pero al llegar al borde del lienzo, su movimiento se detiene y quedan atascados. Vamos a programar a los globos para que se muevan horizontalmente de izquierdas a derechas (y viceversa) hasta que los alcance una flecha.

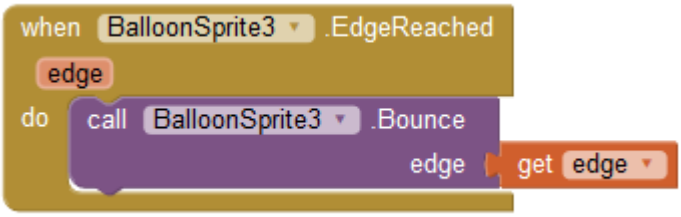


Comenzamos con el globo "BalloonSprite1": Seleccionamos el manejador de evento "when (BalloonSprite1).EdgeReached" y el bloque de función "call (Balloon1).Bounce" de la bandeja "BalloonSprite1". Ubicamos el bloque "call (Balloon1).Bounce" dentro del manejador "when

(BalloonSprite1).EdgeReached". Como queremos que el globo rebote hacia atrás al llegar al borde derecho o izquierdo del lienzo, conectamos al parámetro "edge" del bloque "call (Balloon1).Bounce" una referencia "get (edge)" al valor del argumento "edge" reportado por el manejador "when (BalloonSprite1).EdgeReached".

A continuación programamos de forma similar a los otros dos globos, añadiendo los siguientes programas:

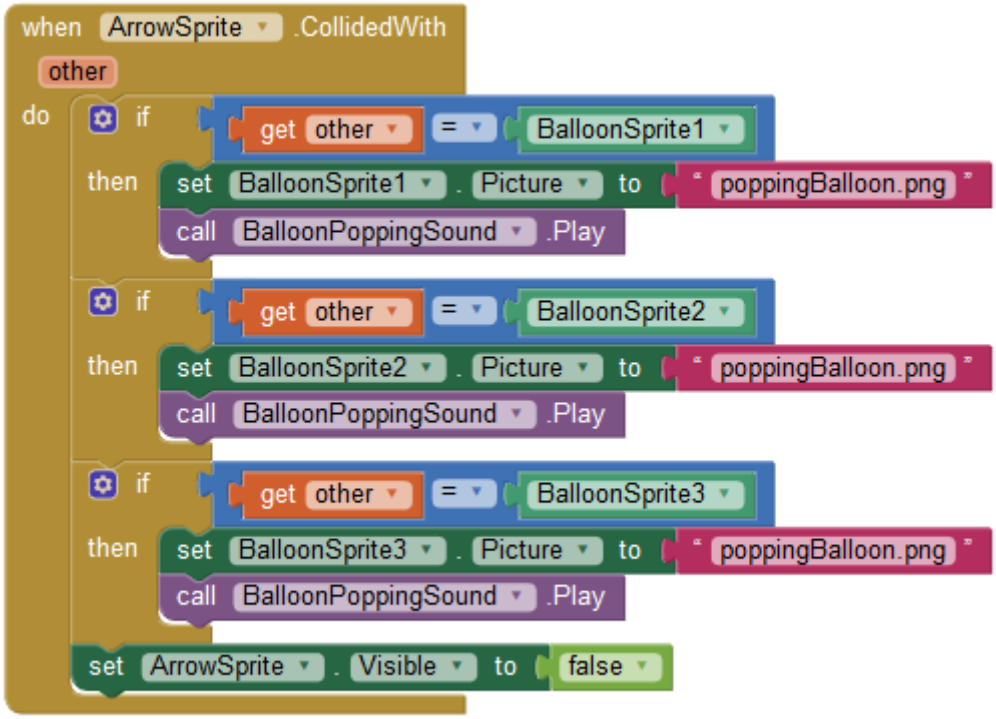




Si probamos la aplicación, veremos que los tres globos se mueven horizontalmente de derechas a izquierdas de forma ininterrumpida. Pero si movemos la flecha y la soltamos, los globos no explotan si son alcanzados, y la flecha se queda clavada en el borde superior de la pantalla. Vamos a corregir estos defectos.

EXPLOTAR LOS GLOBOS.

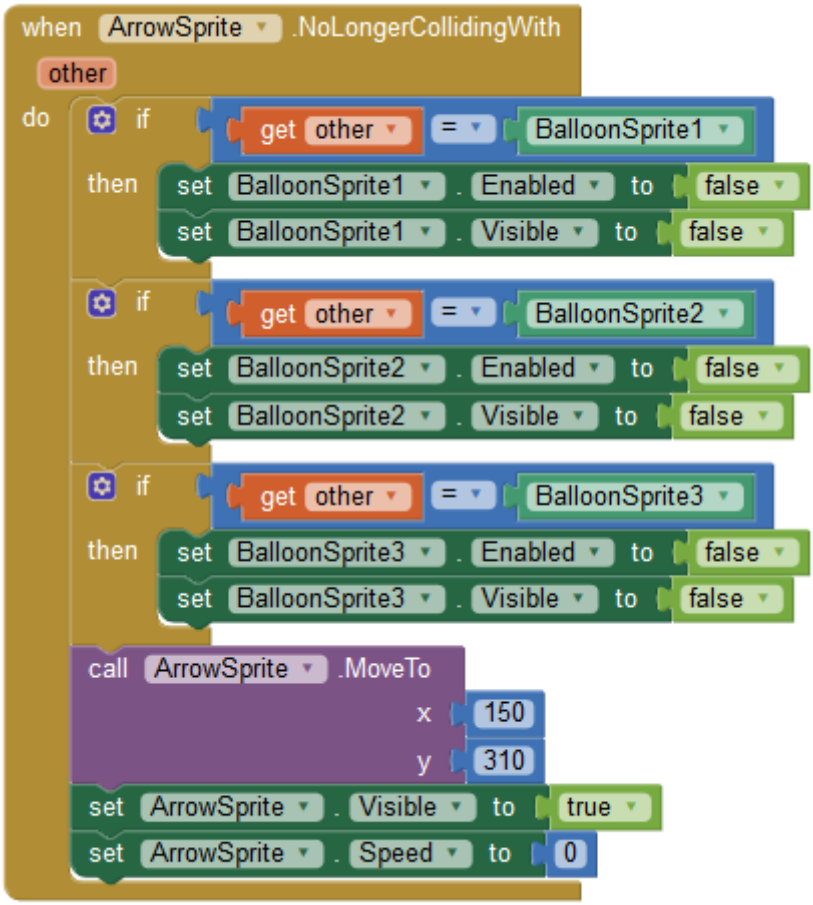
Cuando la flecha impacte con alguno de los globos, queremos que el globo alcanzado cambie su imagen por la de una explosión (archivo *balloonPopping.png*), y que la flecha desaparezca. Para ello, escribimos el programa de la figura:



¿Cómo funciona este código? Cuando la flecha colisiona con algún sprite, se activa el manejador de evento "when (ArrowSprite).CollideWidth". Este manejador incluye un argumento "other" que almacena el sprite contra el que ha colisionado la flecha. A continuación, y dentro del manejador, comprobamos con tres bloques "if" contra qué sprite a colisionado la flecha ("BalloonSprite1", "BalloonSprite2", o "BalloonSprite3"). Si la flecha impacta contra cualquiera de los tres globos, cambiamos la imagen del globo alcanzado por la imagen de la explosión, y reproducimos el sonido de la explosión. Además, y en cualquiera de los tres casos, hacemos invisible a la flecha para hacerla desaparecer.

Ahora, cuando el globo ha explotado y la flecha ha desaparecido, queremos que el globo alcanzado también desaparezca y quede inhabilitado (porque ya no participa en el juego), y que la flecha vuelva a quedar inmóvil en su posición inicial para poder volver a arrastrarla y dispararla. Todas estas acciones deben

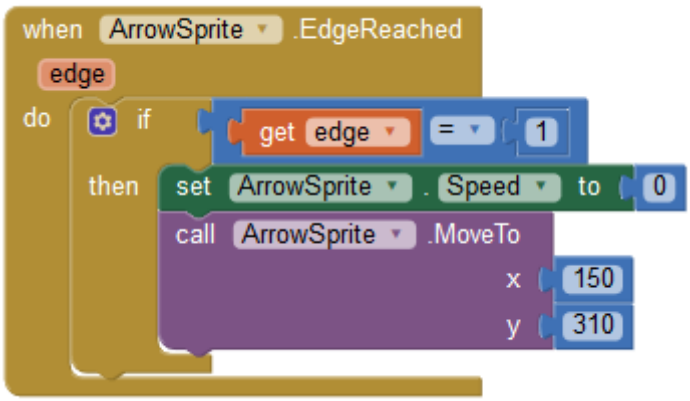
desencadenarse en el momento en el que la flecha deje de colisionar contra el globo alcanzado, evento "NoLongerCollidingWith". Para programar este comportamiento, escribimos el programa de la figura:



¿Cómo funciona este código? Cuando en el programa previo la flecha colisiona contra el globo, hacemos desaparecer a la flecha, y en ese momento deja de colisionar contra el globo. En ese instante se activa el manejador "when (ArrowSprite).NoLongerCollidingWith". Este manejador incluye un argumento "other" que almacena el sprite contra el que la flecha ha dejado de colisionar. Dentro del manejador comprobamos contra qué sprite ha dejado de colisionar la flecha ("BalloonSprite1", "BalloonSprite2", o "BalloonSprite3") con tres bloques "if". Dependiendo del sprite contra el que la flecha haya dejado de colisionar, deshabilitamos y hacemos invisible el sprite involucrado. Además, y en cualquiera de los tres casos, llevamos a la flecha a su posición inicial, $(x, y) = (150, 310)$, la hacemos visible nuevamente, y fijamos su propiedad "Speed" a cero para dejarla inmóvil.

HACER DESAPARECER A LA FLECHA SI FALLA EL DISPARO.

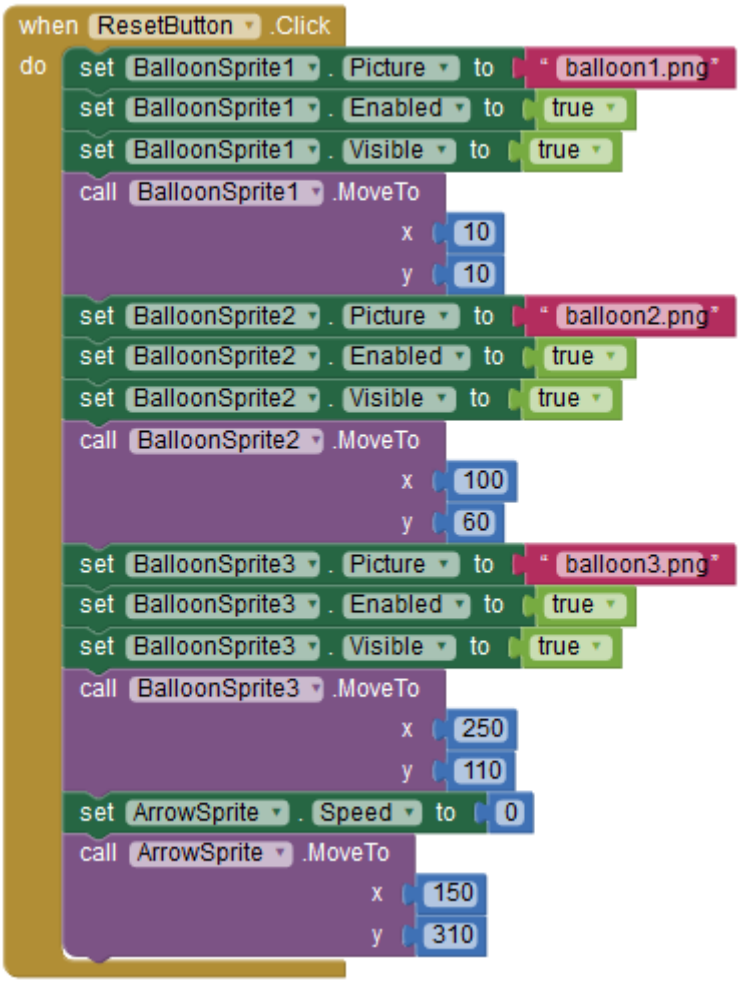
Al lanzar la flecha podríamos errar el disparo, y en ese caso, queremos llevar a la flecha a su posición inicial para poder volver a lanzarla. El siguiente código define este comportamiento:



El funcionamiento de este código es sencillo: Cuando la flecha alcanza un borde del lienzo, se activa el manejador "when (ArrowSprite).EdgeReached". El argumento "edge" de este manejador almacena el borde al que ha llegado la flecha. Dentro del manejador, comprobamos si el borde alcanzado es el borde superior (identifcado por un valor numérico de 1), y en tal caso, dejamos a la flecha inmóvil, y la llevamos de vuelta a su posición inicial en $(x, y) = (150, 310)$.

PROGRAMAR EL BOTÓN DE RESET.

Para terminar queremos que, al presionar el botón de Reset, la app vuelva a su situación original, con los tres globos habilitados y visibles mostrando sus imágenes correspondientes y moviéndose desde sus respectivas posiciones iniciales, y con la flecha en reposo y visible en su posición inicial. Como vemos en la figura, este comportamiento es fácil de programar, pero debemos tener cuidado con no olvidar ninguna de las acciones recién mencionadas.



Y con esto ya hemos terminado. Vamos a probar la aplicación: Al lanzar la flecha y alcanzar un globo, deberíamos oir un sonido de explosión y el globo debería desaparecer. Si la flecha falla el disparo, debería volver a su posición inicial en el centro de la parte baja del lienzo. Si presionamos el botón de Reset deberían volver a aparecer todos los globos explotados para empezar a moverse de izquierdas a derechas en la parte alta del lienzo.

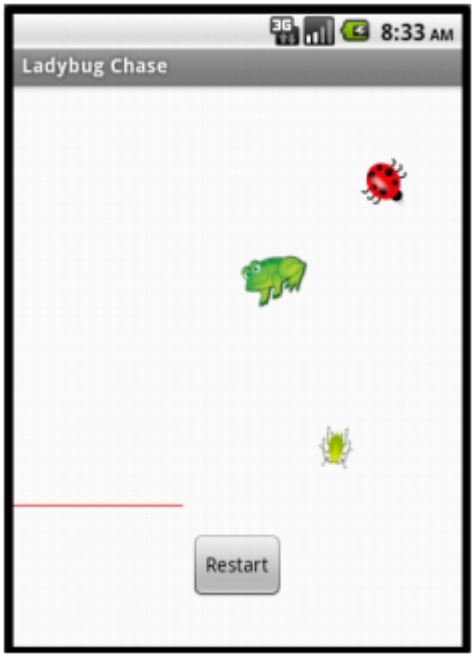
6.16. MODIFICACIONES A "POPPINGBALLOONS".

Algunas variaciones que podemos aplicar a "PoppingBalloons" son las siguientes:

- Podemos añadir más globos que se muevan de formas distintas (más o menos rápido, en diagonal, etc.).
- Podemos añadir un marcador que muestre el número de aciertos y de errores.

- Para hacer el juego más complicado, podemos limitar el número de flechas disponibles para explotar los globos.
- Y para mejorar la apariencia del juego, podemos agregar algunos efectos de sonidos más (al lanzar la flecha, al fallar el disparo, etc.), y añadir una imagen de fondo al lienzo.

6.17. COMENZAR CON LA APP "LADYBUGCHASE"



Vamos a construir un videojuego en el que una mariquita intentará comerse tantas pulgas como pueda, mientras evita que una rana se la coma a ella. El usuario controlará el movimiento de la mariquita inclinando el dispositivo. La app también incluirá una barra de energía que disminuye progresivamente con el tiempo para incitar a la mariquita a comer más pulgas: La mariquita muere si su nivel de energía cae a cero, pero su energía aumenta cada vez que se come una pulga.

Comenzamos creando un nuevo proyecto llamado "LadybugChase", y cambiando el título de la pantalla a "Ladybug Chase". A continuación, conectamos el dispositivo o el emulador para hacer pruebas en vivo.

Para esta app necesitaremos unos cuantos archivos multimedia: Las imágenes *ladybug.png*, *aphid.png*, *dead_ladybug.png*, y *frog.png*, y el archivo de sonido *frog.wav*.

6.18. DISEÑAR LOS COMPONENTES DE "LADYBUGCHASE".

Esta aplicación contendrá un componente "Canvas" que proporcionará el terreno de juego donde se moverán tres objetos "ImageSprite": uno para la mariquita ("LadybugSprite"), otro para la pulga ("AphidSprite"), y otro para la rana ("FrogSprite"). Usaremos un componente "OrientationSensor" para medir la inclinación del dispositivo y mover a la mariquita de forma correspondiente. También usaremos un componente "Clock" para cambiar la dirección en la que se mueven los diferentes sprites. Un segundo "Canvas" mostrará el nivel de energía de la mariquita. La app incluirá un botón de reseteo para reiniciar el juego cuando la mariquita se quede sin energía o cuando se la coma la rana. Vamos a añadir todos estos componentes y a ajustar sus propiedades:

- Si vamos a usar un dispositivo distinto al emulador, debemos deshabilitar la rotación automática de la pantalla (que se encarga de cambiar la orientación de la pantalla al girar el dispositivo). Para ello, seleccionamos el componente "Screen1" y fijamos su propiedad "ScreenOrientation" a "Portrait".
- En el Diseñador creamos un componente "Canvas", lo llamamos "FieldCanvas", y fijamos su propiedad "Width" a "Fill parent" y su propiedad "Height" a 300 píxeles.
- Vamos a añadir a la mariquita: Ubicamos un objeto "ImageSprite" sobre el lienzo "FieldCanvas", lo renombramos "LadybugSprite", y ajustamos su propiedad "Picture" a la imagen *ladybug.png*. Fijamos su propiedad "Interval" a 10 (milisegundos). Por el momento, no ajustamos ningún valor para sus propiedades "Heading" ni "Speed", porque las cambiaremos más adelante en el Editor de Bloques.
- En el Diseñador acudimos a la bandeja "Sensors" y añadimos un componente (no visible) "OrientationSensor", que detectará el grado de inclinación del dispositivo. También de la bandeja "Sensors" añadimos un componente "Clock" para comprobar la orientación del dispositivo cada 10 milisegundos (esto es, 100 veces por segundo) y cambiar la dirección de la mariquita (propiedad "Heading") en consecuencia. Para ello, ajustamos su "TimerInterval" a 10 milisegundos.
- Vamos a mostrar la energía de la mariquita mediante una barra roja ubicada en un segundo lienzo. La línea tendrá un píxel de altura, y su anchura tendrá el mismo número de píxeles que energía tenga la mariquita. La energía de la mariquita variará de 200 (completamente alimentada) a 0 (muerta). Para ello,

creamos un nuevo componente "Canvas", lo ubicamos bajo "FieldCanvas", y lo llamamos "EnergyCanvas". Fijamos su propiedad "Width" a "Fill parent" y su propiedad "Height" a 1 píxel.

- El siguiente paso es añadir la pulga: Creamos otro componente "ImageSprite" (asegurándonos de no ubicarlo justo sobre la mariquita) y lo llamamos "AphidSprite". Fijamos su propiedad "Picture" a la imagen de la pulga, *aphid.png*. Fijamos su propiedad "Interval" a 10 para que se mueva cada 10 milisegundos, igual que la mariquita. Fijamos su propiedad "Speed" a 2, de forma que no se mueva demasiado rápido para que la mariquita pueda atraparla. No debemos preocuparnos por su propiedad "Heading", que ajustaremos en el Editor de Bloques.
- Ahora añadimos a la rana: agregamos un tercer "ImageSprite" (que renombraremos como "FrogSprite") al lienzo "FieldCanvas". Fijamos su propiedad "Picture" al archivo *frog.png*, su propiedad "Interval" a 10, y su propiedad "Speed" a 1, porque la rana debe moverse más despacio que el resto de criaturas de la app.
- Ahora añadimos un componente "Button" bajo el "EnergyCanvas", lo renombramos como RestartButton, y fijamos su propiedad "Text" a "Restart".
- Finalmente añadimos un componente "Image" vacío para separar la barra de energía del botón de reinicio.

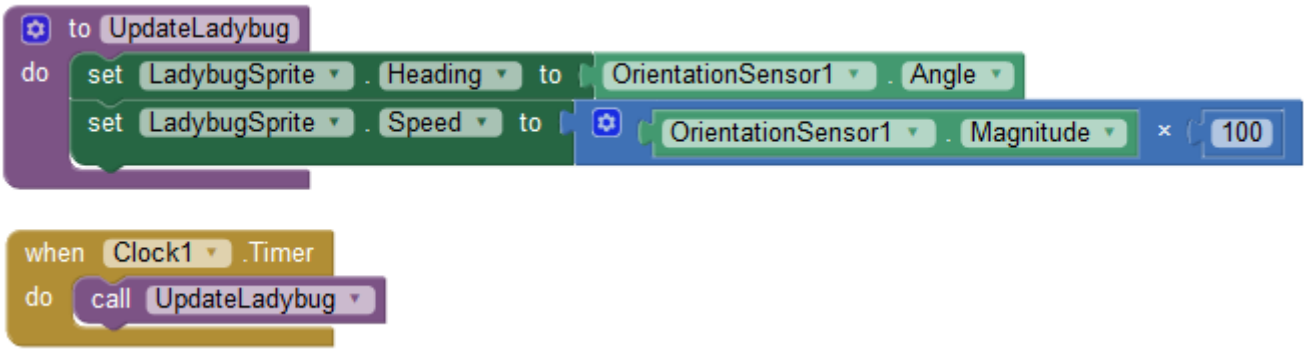
La pantalla de nuestra app debería parecerse a la mostrada en la figura:

6.19. AÑADIR COMPORTAMIENTOS A LOS COMPONENTES DE "LADYBUGCHASE".

MOVER LA MARIQUITA.

Acudimos al Editor de Bloques, creamos el procedimiento "UpdateLadybug", y completamos el manejador de eventos "when (Clock1).Timer" como muestran las figuras. Para acelerar la creación de bloques, intenta escribir el nombre de un bloque (como "Clock1.Timer") en vez de buscarlo y arrastrarlo desde su bandeja correspondiente. No es obligatorio añadir comentarios a los programas, pero si así lo deseamos, podemos

hacerlo pinchando con el botón derecho del ratón sobre un bloque, y seleccionando la opción "Add Comment".



El procedimiento "UpdateLadybug" usa dos de las propiedades más útiles del componente "OrientationSensor":

- La propiedad "Angle" indica la dirección en la que se ha inclinado el dispositivo (en grados).
- La propiedad "Magnitude" indica la cantidad de inclinación en la dirección especificada por "Angle", y varía entre 0 (sin inclinación) y 1 (máxima inclinación).

Al multiplicar "Magnitude" por 100 y asignar este valor a "Speed", le estamos diciendo a la mariquita que se mueva entre 0 y 100 píxeles en la dirección especificada por "Angle" en su propiedad "Heading" cada vez que se dispare el temporizador de "Clock1". El tiempo del temporizador lo fijamos en la propiedad "TimerInterval" a 10 milisegundos.

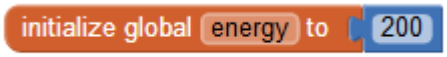
Aunque podemos probar el funcionamiento de este programa en el dispositivo que tengamos conectado, el movimiento de la mariquita podría ser más lento o más brusco que si empaquetamos la app y la instalamos en el dispositivo. Si después de instalar la app vemos que el movimiento de la mariquita es demasiado lento, debemos incrementar el multiplicador para la rapidez, y si la mariquita se mueve demasiado rápido, debemos disminuirlo.

MOSTRAR EL NIVEL DE ENERGÍA.

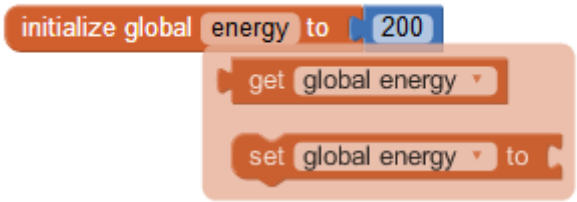
Crear una variable para la energía.

En el Editor de Bloques creamos una variable llamada "energy" con un valor inicial de 200, para controlar el nivel de energía de la mariquita. Aunque ya aprendimos a crear variables en el capítulo 4, vamos a recordar el proceso:

1) En el Editor de Bloques seleccionamos un bloque "initialize global (name) to" de la bandeja "Variables", y le cambiamos el texto "name" por "energy".
2) Creamos un bloque numérico con unvalor de 200, y se lo conectamos a la derecha del bloque "initialize global (energy) to".



Recordar que, al definir una variable, se crean unos nuevos bloques "get" y "set" para acceder y fijar el valor de esa variable, respectivamente. Esos bloques pueden seleccionarse pasando el ratón sobre el nombre de la variable en el bloque "initialize global (name) to", ver figura:



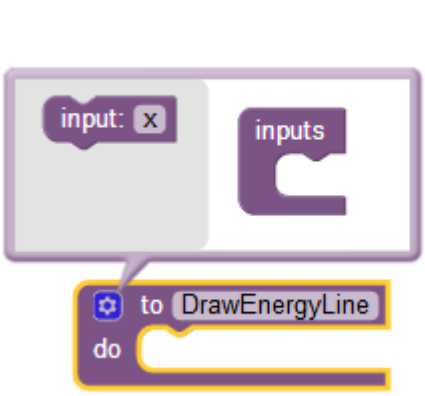
Crear un procedimiento con parámetros para dibujar la barra de energía.

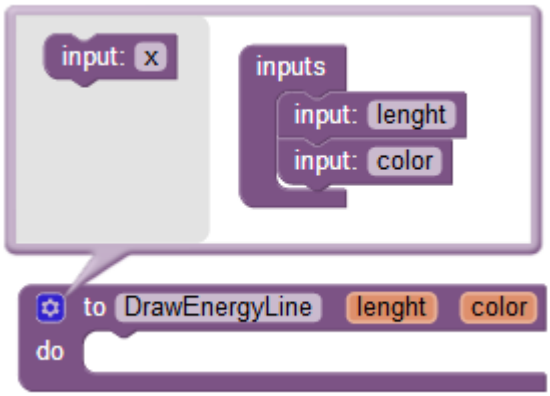
Queremos indicar el nivel de energía con una barra roja que tenga una longitud en píxeles igual al valor actual de energía. Para ello podríamos hacer lo siguiente:

1) Dibujar una línea roja desde el punto $(0, 0)$ hasta el punto $(\text{energy}, 0)$ en "EnergyCanvas" para mostrar el nivel de energía actual.
2) Dibujar una línea blanca desde el punto $(0, 0)$ hasta el punto $(\text{EnergyCanvas.Width}, 0)$ en "EnergyCanvas" para borrar el nivel de energía actual antes de dibujar el nuevo nivel de energía

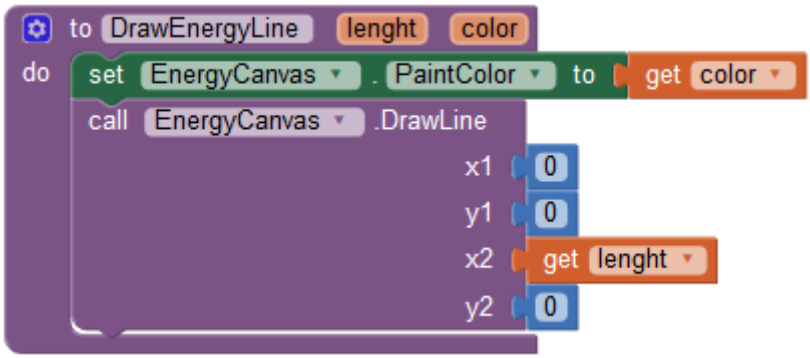
Sin embargo, es mucho mejor crear un procedimiento que dibuje en "EnergyCanvas" una línea de una longitud y un color cualesquiera. Pero para ello tendremos que pasarle al procedimeinto el valor de unos parámetros "lenght" y "color" cada vez que lo llamemos, de la misma forma que tenemos que especificar los valores de ciertos parámetros al llamar al procedimiento integrado "random integer". A continuación detallamos los pasos para crear el procedimiento "DrawEnergyLine":

1) Vamos a la bandeja "Procedures" y seleccionamos un bloque "to (procedure) do". (Nos aseguramos de elegir la versión que contiene un "do" y no un "return", porque este procedimiento no va a devolver ningún valor).
2) Clicamos en el nombre del procedimiento (que ahora es "procedure"), y lo cambiamos a "DrawEnergyLine".
3) En la esquina superior izquierda de este nuevo bloque pinchamos en el cuadrado azul con el engranaje. Al hacerlo se abrirá la ventana mostrada en la primera figura.
4) En esta nueva ventana arrastramos un bloque "input" desde la izquierda a la derecha de la ventana, cambiando su nombre de "x" a "length". Con esto indicamos que el procedimiento recibirá un parámetro llamado "length".
5) Repetimos el proceso para crear un segundo parámetro llamado "color", que debe ir debajo del parámetro "length". El procedimiento debería quedar como muestra la figura.

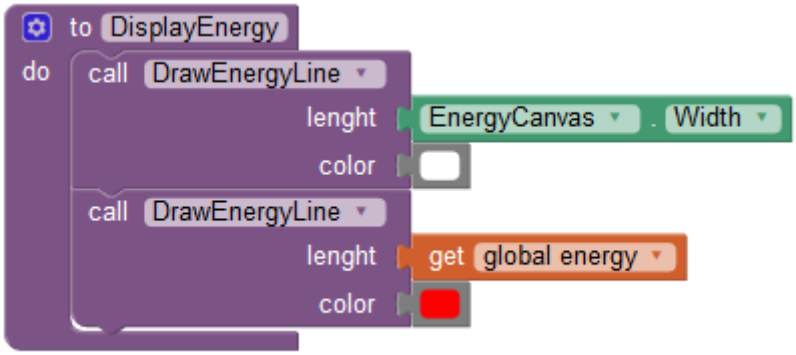




6) Pinchamos en el icono del engranaje para cerrar la ventana de creación de parámetros.
7) Rellenamos el resto del procedimiento como muestra la figura a continuación. Podemos acceder a los bloques "get (color)" y "get (length)" pasando el ratón por encima de sus nombres en la definición del procedimiento.



Ahora que ya sabemos cómo crear nuestros propios procedimientos con parámetros, vamos a escribir un procedimiento "DisplayEnergy" que llame al procedimiento "DrawEnergyLine" dos veces: Una para borrar la línea previa (dibujando una línea blanca a lo largo de toda la anchura del lienzo), y otra para mostrar la nueva línea:



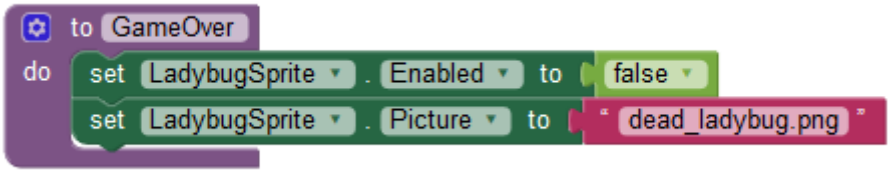
El procedimiento "DisplayEnergy" hace lo siguiente:

1) Fija el color con el que se va a dibujar a blanco.
2) Dibuja una línea a lo largo de toda la anchura del lienzo "EnergyCanvas" (el cual solo tiene 1 pixel de altura).
3) Fija el color con el que se va a dibujar a rojo.
4) Dibuja una línea cuya longitud en píxeles es igual al valor de energía actual.

PROGRAMAR LA MUERTE POR INANICIÓN.

Al contrario que algunas apps previas, este juego tiene una forma de finalizar: El juego termina si la mariquita no come lo suficiente y su nivel de energía cae a cero, o si se la come la rana. En cualquiera de estos casos queremos que la mariquita deje de moverse (lo que haremos fijando la propiedad "LadybugSprite.Enabled" a "false"), y que la imagen que la representa cambie de la mariquita viva a la mariquita muerta (lo que podemos hacer cambiando el valor de la propiedad "LadybugSprite.Picture" al archivo *dead_ladybug.png*).

Para ello, creamos el procedimiento "GameOver" como muestra la figura:

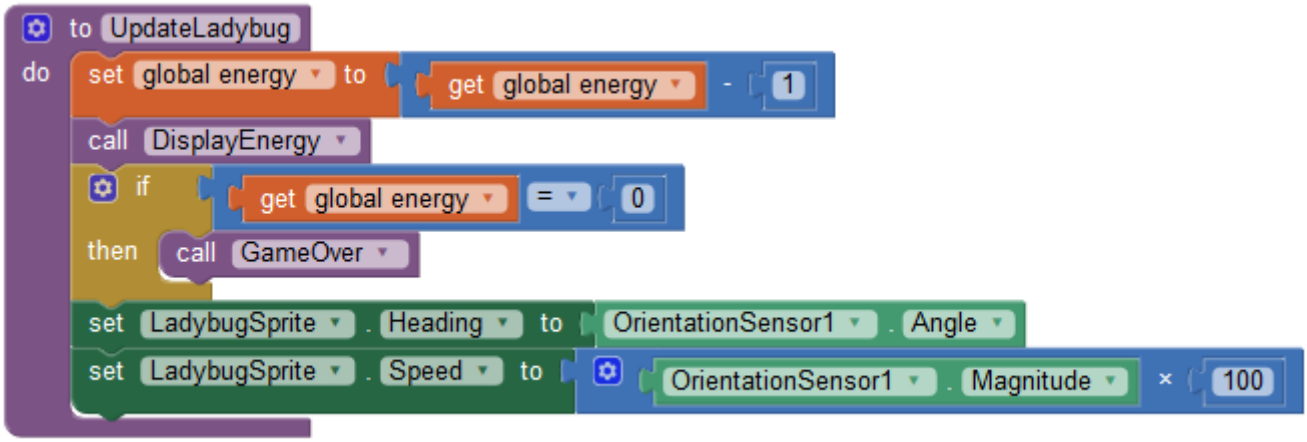


El bloque "false" está disponible en la bandeja "Logic", y el bloque con el nombre de la imagen *dead_ladybug.png* es un bloque de texto de la bandeja "Text".

A continuación completamos el procedimiento "UpdateLadybug" (al cual llama el componente "Clock" cada 10 milisegundos) como muestra la figura para hacer lo siguiente:

- Disminuir el nivel de energía de la mariquita.
- Mostrar el nuevo nivel.
- Terminar el juego si la energía cae a cero.

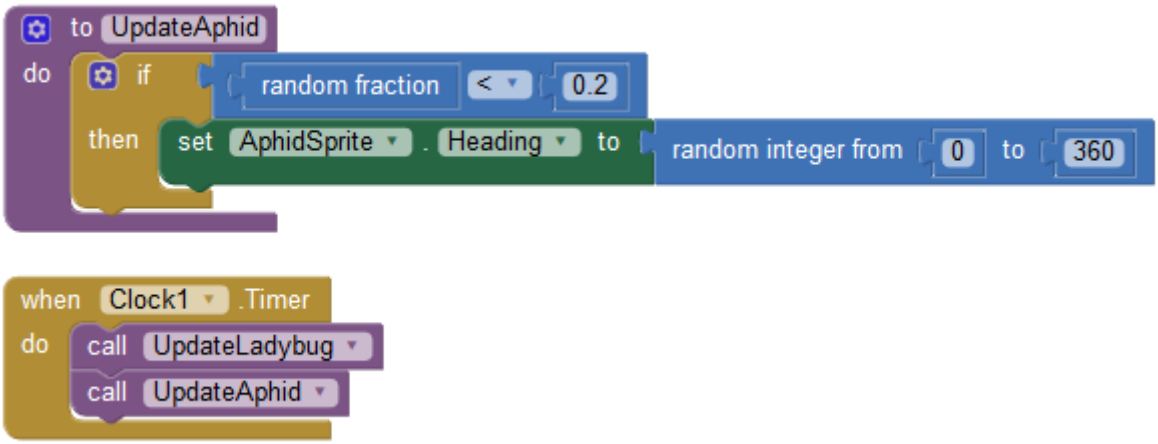
Comprobamos el funcionamiento de la app en nuestro dispositivo: Verificamos que el nivel de energía disminuye con el tiempo, y que al llegar a cero, la mariquita fallece.



PROGRAMAR A LA PULGA.

El siguiente paso es controlar a la pulga. La pulga debería revolotear por el "FieldCanvas". Si la mariquita se encuentra con la pulga (lo que significará que se la come), el nivel de energía de la mariquita debería aumentar, y la pulga debería desaparecer para ser reemplazada por otra un tiempo después. (Desde el punto de vista del usuario la nueva pulga que aparezca será otra diferente, pero en realidad se tratará del mismo objeto "ImageSprite").

Por simple prueba y error hemos determinado que lo mejor es que la pulga cambie su dirección cada 50 milisegundos (esto es, tras 5 intervalos del reloj "Clock1"). Una forma de programar este comportamiento podría ser crear un segundo reloj con un "TimerInterval" de 50 milisegundos. Sin embargo, aquí vamos a emplear una técnica alternativa para aprender a manejar el bloque "random fraction", que devuelve un número aleatorio mayor o igual que 0 y menor que 1 cada vez que se le llama. Para hacer todo esto, creamos el procedimiento "UpdateAphid" como muestra la figura, y añadimos una llamada al mismo en el manejador "when (Clock1).Timer".



¿Cómo funciona este código? Siempre que el temporizador termine su cuenta atrás (esto es, cada 10 milisegundos, o lo que es igual, 100 veces por segundo), se llama a los procedimientos "UpdateLadybug" y "UpdateAphid". Lo primero que hace el procedimiento "UpdateAphid" es generar un número fraccionario aleatorio entre 0 y 1 (por ejemplo, 0.36). Si el número es menor que 0.20 (lo que ocurrirá el 20% del tiempo), la pulga cambiará de dirección a un número de grados aleatorio entre 0 y 360. Si el número no es menor que 0.20 (lo que ocurrirá el 80% del tiempo), la pulga mantendrá su dirección actual.

PROGRAMAR QUE LA MARIQUITA SE COMA A LA PULGA.

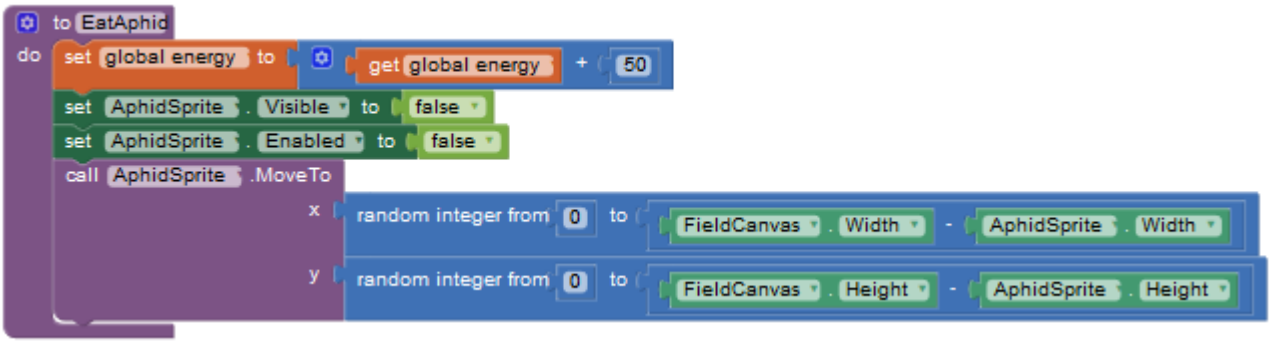
El siguiente paso es hacer que la mariquita se "coma" a la pulga cuando ambos objetos entran en contacto, o como se dice en programación, cuando *colisionan*. Como ya sabemos, App Inventor proporciona una serie de bloques para detectar colisiones entre componentes "ImageSprite". Pero esto suscita una pregunta: ¿Qué

debería ocurrir cuando colisionan la mariquita y la pulga? Vale la pena tomarse un momento a reflexionar sobre ello…

Para gestionar lo que ocurre cuando la mariquita y la pulga colisionan vamos a crear un procedimiento, llamado "EatAphid", que haga lo siguiente:

- Incrementar el nivel de energía en 50.
- Hacer que la pulga desaparezca (ajustando su propiedad "Visible" a "false").
- Forzar que la pulga deje de moverse (ajustando su propiedad "Enabled" a "false").
- Hacer que la pulga se mueva a una localización aleatoria en la pantalla. (Esto lo haremos de forma similar a como lo hicimos en el programa "MoleMash").

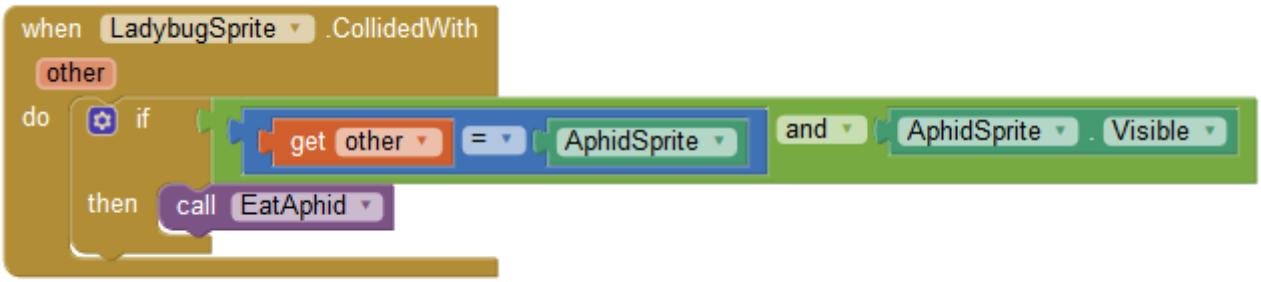
Escribimos nuestro propio programa y comprobamos que se ajusta al mostrado en la figura. Si nos apetece añadir algunos elementos extras (como efectos de sonido), podemos hacerlo.



¿Cómo funciona este código? Siempre que llamamos al procedimiento "EatAphid", éste suma 50 unidades a la variable energía, lo que evitará que la mariquita muera por inanición. A continuación se fijan las propiedades "Visible" y "Enabled" de la pulga a "false", para simular que desaparece y para hacer que se detenga, respectivamente. Finalmente, genera unas coordenadas x e y aleatorias para el procedimiento "(AphidSprite).MoveTo", para que la próxima vez que aparezca, lo haga en una localización desconocida (de otra forma, la mariquita se comería a la pulga nada más reaparecer).

DETECTAR LA COLISIÓN ENTRE LA MARIQUITA Y LA PULGA.

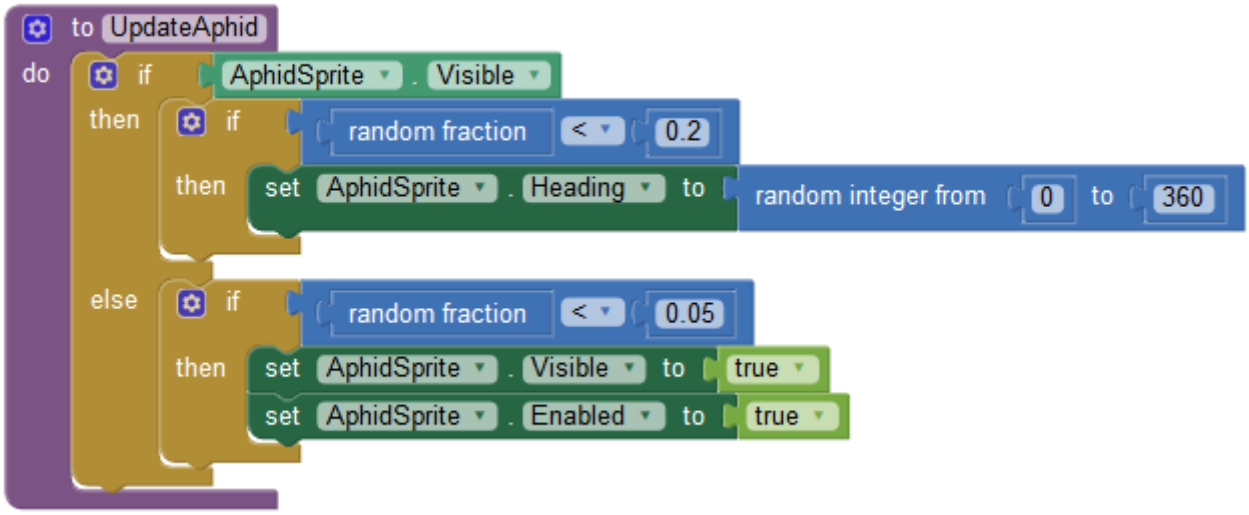
La figura muestra el código para detectar las colisiones entre la mariquita y la pulga:



¿Cómo funciona el código? Cuando la mariquita colisiona con otro "ImageSprite" se activa el manejador "when (LadybugSprite).CollideWith", donde el parámetro "other" representa al objeto contra el que ha colisionado la mariquita. Ahora mismo, la única cosa contra la que puede colisionar la mariquita es la pulga, pero más adelante también podrá colisionar contra la rana. Aquí usamos *programación defensiva* y comprobamos explícitamente si la colisión fue contra la pulga antes de llamar al procedimiento "EatAphid". También hacemos una comprobación que confirma si la pulga está visible. (De no hacer esta última comprobación, después de que la mariquita se coma a la pulga y antes de reaparecer, la pulga podría volver a colisionar con la mariquita. En tal caso la mariquita se comería de nuevo a la pulga invisible, provocando un nuevo e indeseado incremento en su energía).

HACER RETORNAR A LA PULGA.

Para hacer que la pulga reaparezca tras ser comida debemos modificar el procedimiento "UpdateAphid" como muestra la figura, de forma que cambie la dirección de la pulga solo si está visible. (Cambiar su dirección cuando es invisible es una pérdida de tiempo). Si la pulga no está visible (como cuando la mariquita se la acaba de comer), hay una probabilidad de 1 entre 20 (esto es, un 5%) de que vuelva a quedar habilitada ("Enabled"), es decir, de que pueda ser comida nuevamente.



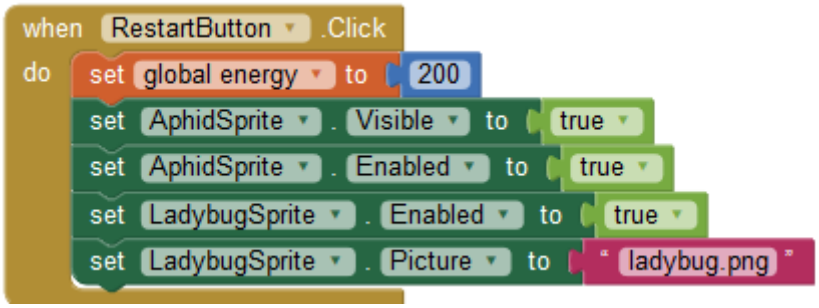
¿Cómo funciona el código? El método "UpdateAphid" se está volviendo un poco enrevesado, así que vamos a analizar cuidadosamente su funcionamiento:

- Si la pulga está visible (lo que siempre será el caso, menos cuando la mariquita se la acabe de comer), "UpdateAphid" se comporta tal y como lo escribimos antes, a saber, otorgándole una probabilidad del 20% para cambiar dirección.
- Si la pulga no está visible (cuando la mariquita se la acaba de comer), se ejecutará la parte "else" del bloque "if-else". Entonces se genera un número fraccionario aleatorio. Si ese número es menor que 0.05 (lo que ocurrirá el 5% del tiempo), la pulga vuelve a hacerse visible y a activarse ("Enabled"), lo que posibilita que la mariquita se la pueda comer nuevamente. Como "when (Clock1).Timer" llama al método "UpdateAphid" cada 10 milisegundos y hay un 5% de probabilidades de que la pulga se haga visible nuevamente, la pulga tardará en promedio unos 200 milisegundos en volver a aparecer.

PROGRAMAR EL BOTÓN DE REINICIO.

Si hemos probado la app puede que nos hayamos dado cuenta de que el juego ciertamente necesita un botón para reiniciar la partida. En el Editor de Bloques creamos el código mostrado en la figura, para que al pulsar en el botón "RestartButton" ocurra lo siguiente:

- Fijamos el nivel de energía de vuelta a 200.
- Reactivamos a la pulga y la hacemos visible.
- Reactivamos a la mariquita y cambiamos su imagen a la de la mariquita viva (de no hacerlo tendríamos una mariquita zombi).



PROGRAMAR A LA RANA.

Tal y como tenemos la app hasta ahora no es demasiado difícil mantener viva a la mariquita. Para complicar el juego vamos a añadir un depredador. En concreto añadiremos una rana que se mueva directamente hacia la mariquita. Si ambos objetos colisionan, la rana se come a la mariquita y el juego termina.

Conseguir que la rana persiga a la mariquita.

La figura muestra el procedimiento "UpdateFrog", al que llamará el manejador "when (Clock1).Timer" cada 10 milisegundos.

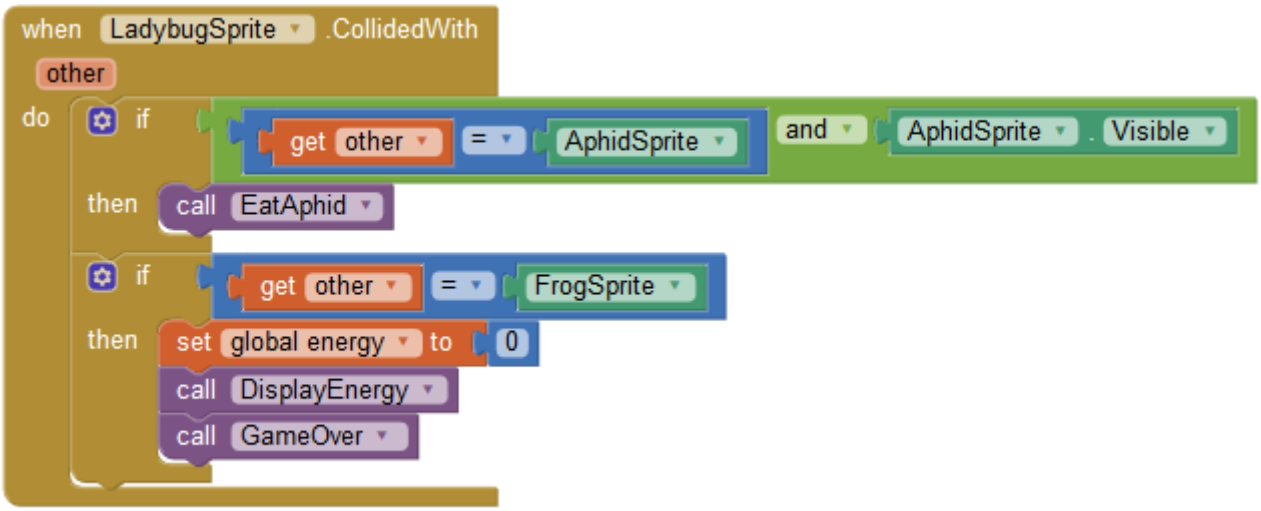
¿Cómo funciona este código? A estas alturas ya deberíamos estar familiarizados con el uso del bloque "random fraction" para hacer que un cierto evento ocurra con una probabilidad determinada. En este caso, hay un 10% de posibilidades de que la dirección de la rana cambie para dirigirse directa hacia la posición de la mariquita. Para hacer que la rana acuda a donde se sitúa la mariquita necesitamos algo de trigonometría. Por suerte, App Inventor proporciona un montón de funciones matemáticas, incluso trigonométricas. En nuestro caso, necesitamos el bloque "atan2" (la función arcotangente), el cual devuelve el ángulo correspondiente a unos ciertos valores de las coordenadas x e y. (Para aquellos que están familiarizados con la trigonometría, la razón por la que el parámetro "y" de "atan2" tiene signo opuesto (porque la resta se efectúa en orden inverso) es que en los lienzos de App Inventor la coordenada y se incrementa hacia abajo, mientras que en los sistemas coordenados más habituales esta coordenada se incrementa hacia arriba).

Hacer que la rana se coma a la mariquita.

Ahora hemos de modificar el código que detecta las colisiones de la mariquita para que incluya las colisiones con la rana. Si la mariquita colisiona con la rana su nivel de energía cae a cero y el juego termina:

¿Cómo funciona este código? Además del primer "if" (el cual comprueba si la mariquita ha colisionado con la pulga), ahora tenemos un segundo "if" que mira si la mariquita ha colisionado con la rana. Si la mariquita y la rana colisionan, ocurren tres cosas:

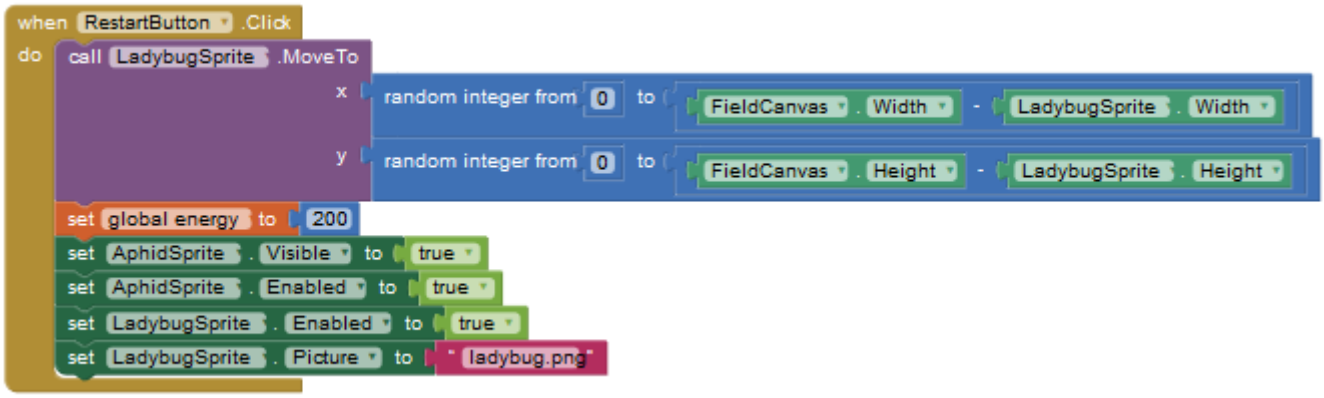
1) La variable "energy" se va a cero, para que la mariquita muera.
2) Se llama al procedimiento "DisplayEnergy" para borrar la barra de energía previa.
3) Se llama al procedimiento "GameOver" para hacer que la mariquita deje de moverse y cambiar su imagen por la de la mariquita muerta.



HACER RETORNAR A LA MARIQUITA.

El manejador "when (RestartButton).Click" ya incluye el código para cambiar la imagen de la mariquita muerta a la imagen de la mariquita viva. Ahora debemos añadirle el código para mover a la nueva mariquita viva a una localización aleatoria. (Esto es muy importante. Piensa lo que ocurriría si no moviésemos a la mariquita a una nueva posición al comenzar una nueva partida. ¿Dónde estaría en relación a la rana?). La figura muestra el código necesario para mover a la mariquita cuando empieza una nueva partida:

¿Cómo funciona este código? La única diferencia entre esta nueva versión del manejador "when (RestartButton).Click" con la versión previa es el bloque "call (LadybugSprite).MoveTo" y sus argumentos. Llamamos a la función "random integer" dos veces: Una vez para generar una coordenada x válida y otra más para producir una coordenada y válida. Aunque no hacemos ninguna comprobación que evite que la mariquita aparezca justo sobre la pulga o la rana, hay pocas probabilidades de que esto ocurra.



Vamos a probar nuestra app: Reiniciamos el juego y nos aseguramos de que la mariquita aparece en una nueva localización totalmente aleatoria.

AÑADIR EFECTOS DE SONIDO.

Al probar el videojuego notaremos que la respuesta del mismo no es del todo buena cuando algo se come a algo. Para mejorar este comportamiento vamos a añadir algunos efectos de sonido y respuestas táctiles:

1) En el Diseñador añadimos un componente "Sound". Fijamos su propiedad "Source" al archivo de sonido *frog.wav*.
2) Vamos al Editor de Bloques, donde hacemos lo siguiente:
   - Hacemos que el dispositivo vibre cada vez que la mariquita se coma una pulga, añadiendo al procedimiento "EatAphid" un bloque "call (Sound1).Vibrate" con un argumento de 100 (milisegundos).

- Hacemos que la rana croe cuando se come a la mariquita, añadiendo una llamada a "(Sound1).Play" en el manejador "when (LadybugSprite).CollideWith" justo antes de la llamada al método "GameOver".

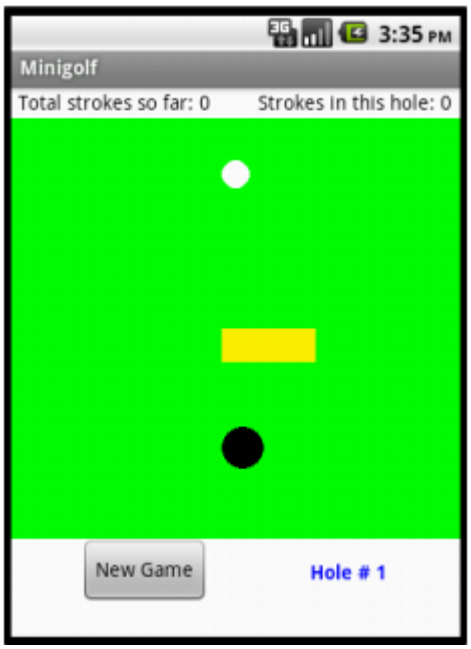
Con esto, hemos terminado nuestra app. Pruébala en tu dispositivo Android para fijar los valores más adecuados para los distintos parámetros de control de la dinámica del videojuego. (Por ejemplo, puede que para nuestro gusto la barra roja de energía disminuya demasiado rápido. En ese caso deberíamos reducir la cantidad que le restamos a la variable "energy" cada vez que el reloj cuenta 10 milisegundos, o empezar con un nivel de energía mayor de 200, o tal vez otorgar más puntos de energía cada vez que la mariquita se come una pulga, etc.).

6.20. MODIFICACIONES A "LADYBUGCHASE".

He aquí unas cuantas ideas para mejorar o personalizar el videojuego de la mariquita:

- Ahora mismo tanto la rana como la pulga siguen moviéndose después de que la partida haya terminado. Evita este comportamiento fijando sus propiedades "Enabled" a "false" en el procedimiento "GameOver". Pero entonces debemos acordarnos de fijarlas a "true" cuando el usuario empiece una nueva partida, esto es, en el manejador "when (RestartButton).Click".
- Otra buena idea sería mostrar un marcador que indique el tiempo que la mariquita sigue con vida. Esto podemos hacerlo creando una etiqueta que vayamos incrementando en el manejador "when (Clock1).Timer".
- Podemos hacer que la barra de energía sea más visible incrementando la propiedad "Height" de "EnergyCanvas" a 2, y dibujando dos líneas (una sobre otra) en el procedimiento "DrawEnergyLine". (Aquí vemos una gran ventaja de haber usado un procedimiento para borrar y redibujar la barra de energía: Ahora solo necesitamos cambiar el código en este procedimiento para modificar la forma en la que se dibuja la barra).
- Para añadir algo de ambientación podemos agregar una imagen de fondo y algunos efectos de sonido más, como por ejemplo, sonidos de la naturaleza, o un aviso acústico cuando el nivel de energía de la mariquita esté muy bajo.
- Podemos hacer que la dificultad del videojuego aumente con el tiempo, por ejemplo, aumentando la propiedad "Speed" de la rana o disminuyendo su propiedad "Interval".
- Técnicamente, la mariquita debería desaparecer cuando se la come la rana. Cambia el juego para que la mariquita se vuelva invisible cuando se la coma la rana, pero que cambie a su imagen de mariquita muerta cuando su nivel de energía llegue a cero.
- Reemplaza las imágenes de la mariquita, la pulga, y la rana por otras más a tu gusto, como un hobbit, un orco, un mago; superhéroes y supervillanos; jedis, siths, y soldados imperiales, etc.

6.21. EJERCICIOS DEL CAPÍTULO 6.



Ejercicio 6.1. Minigolf.
En este ejercicio vamos a programar una sencilla app de minigolf, donde deberemos usar el evento "Flung" que estudiamos en el capítulo previo, y los nuevos eventos "TouchUp", y "TouchDown".

La mecánica del juego es la siguiente: El jugador primero posiciona la pelota en un sporte, y a continuación, la lanza hacia el agujero. La pelota podrá rebotar contra los obstáculos y paredes que se encuentre. Por cada lanzamiento. El contador de golpes se incrementa en una unidad cada vez que el jugador lanza la pelota. Por supuesto, el juego consiste en meter la pelota en el agujer con el mínimo número de golpes posible.

Vamo a programar esta app en etapas, añadiendo componentes y funcionalidades al juego en cada uno de los pasos. Creamos una nueva app llamada "Minigolf". Al abrirse la ventana del diseñador, acudimos a las propiedades del componente "Screen1" y nos aseguramos de desactivar la opción "Scrollable".

Paso 1: Crear y programar el lanzamiento de la pelota.

a) En el Diseñador añadimos un componente "Canvas" de altura 300 píxeles y anchura igual a la de su componente padre ("Fill Parent"). Como color de fondo elegimos el verde.
b) Para implementar la pelota añadimos al lienzo un componente "Ball", al que llamaremos "GolfBall". A continuación, fijamos los siguientes valores para sus propiedades:
   - "Radius": 10.
   - "PaintColor": blanco.
   - "Speed": 0 (para que la pelota aparezca estática).
   - "Interval": $1\ ms$.
   - "X": 150, "Y": 30, "Z": 1.
c) Acudimos al Editor de Bloques, para programar el lanzamiento de la pelota. Para lanzar la pelota, el usuario deslizará su dedo sobre ella, y la pelota pasará a moverse en la dirección de ese movimiento de deslizamiento, y con una rapidez proporcinal a la rapidez de dicho movimiento. Como sabemos de capítulos previos, este comportamiento puede programarse usando el manejador de eventos "when (GolfBall).Flung". Este manejador simplemente ajustará las propiedades "Heading" y "Speed" de la pelota después de que el usuario deslice su dedo sobre ella para lanzarla. Como en la app "FlingFlung", al ajustar el valor de la propiedad "Speed" vamos a multiplicar por 4 el argumento "speed" del manejador "when (GolfBall).Flung", para que la pelota se mueva más rápido que el gesto de deslizamiento.

Paso 2: Reducir progresivamente la rapidez de la pelota.

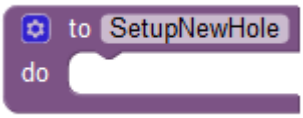
a) Tras ser golpeada, la pelota debería ir perdiendo velocidad progresivamente conforme se mueve por el campo. Si no programamos este comportamiento, la pelota se estará moviendo con una rapidez constante y no se frenará nunca. Para implementar esta funcionalidad, añadimos un componente "Clock" en el Diseñador, y fijamos los siguientes valores para sus propiedades:
   - TimerAlwaysFires: Activado.
   - TimerEnabled: Activado.
   - TimerInterval: $100\ ms$.
b) Programamos el comportamiento del reloj: Queremos que el reloj reduzca un poco la rapidez de la pelota cada vez que se dispare su temporizador (esto es, cada $100\ ms$). Si la pelota no se está moviendo, el reloj no debería hacer nada. Este comportamiento lo implementaremos con el manejador "when (Clock1). Timer", y funcionará de la siguiente manera:
   - Si la rapidez de la pelota es mayor o igual que 0.5, reajustamos el nuevo valor de la propiedad "Speed" de la pelota retándole 0.5 unidades a su valor actual.
   - En otro caso, o bien la pelota ya estaba parada, o estará senecialmente estática, así que fijamos el valor de la propiedad "Speed" a 0.

Paso 3: Creamos y programamos el agujero.

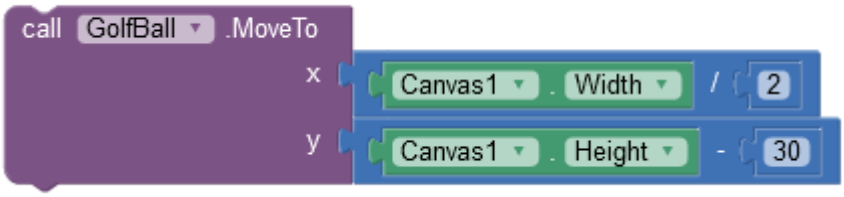
a) Acudimos el Diseñador, y añadimos al lienzo un segundo componente "Ball", que representará el agujero donde hay que colar la pelota. Renombramos este componente como "Hole", y fijamos los siguientes valores para sus propiedades:
   - "Radius": 15.
   - "PaintColor": negro.
   - "Speed": 0.
   - "X": 150, "Y": 220, "Z": 1.
b) Escribimos un procedimiento llamado "SetupNewHole" que entrará en acción cuando la pelota entre en el agujero. Cuando eso ocurra, el procedimiento devolverña la pelota a su posición de inicio y moverá el agujero a un lugar aleatorio en la pantalla, para jugar la siguiente partida.

Para ello:

- Acudimos a la bandeja "Procedures" y arrastramos un bloque "to (procedure)" al área de trabajo. Cambiamos el nombre del procedimiento de "procedure" a "SetupNewHole".

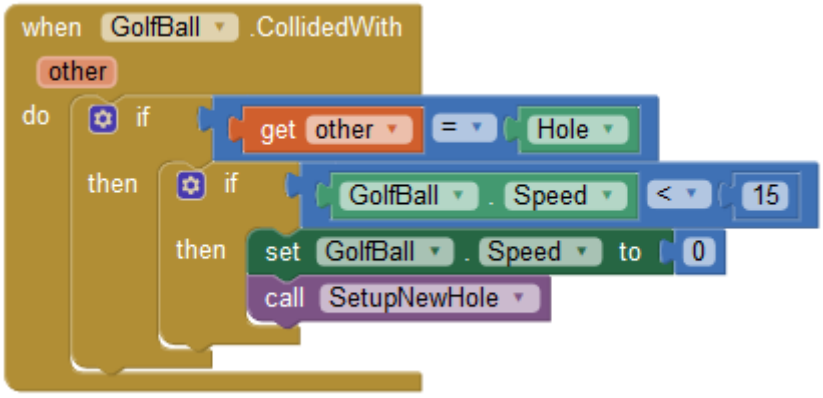


- Comenzamos a programar el procedimiento añadiendo bloques a la sección "do". Lo primero que hacemos cuando la pelota se cuela en el agujero es llevarla a su posición de inicio. Usamos la función "(GolfBall).MoveTo" para llevar a la pelota a una ubicación centrada horizontalmente en el lienzo y a 30 píxeles del borde superior del lienzo:



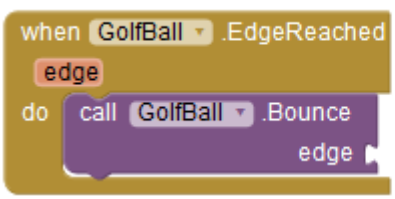
- A continuación, movemos el hoyo a una posición aleatoria en el lienzo, usando nuevamente la función "(Hole).MoveTo":

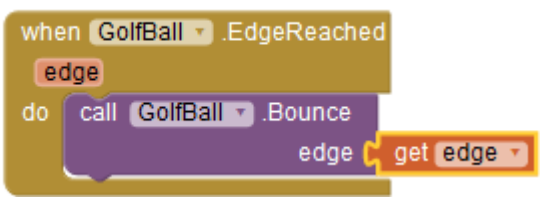
c) Ahora programamos el comportamiento del agujero: Cuando la pelota golpee colisione con otro sprite, primero comprobamos *si* ese sprite es efectivamente el agujero. En caso afirmativo queremos que, *si* la velocidad de la pelota es la adecuada (digamos, menor que 15), la pelota entre en el agujero. (En otro caso, la pelota va demasiado rápido y se saldrá del agujero). Si la velocidad es la adecuada, la la app reduce la velocidad de la pelotaa a 0, y llama al procedimiento "SetupNewHole".



Paso 4: Hacemos que la pelota rebote al llegar a un borde.

Si probamos la app tal y como está ahora mismo, veremos que la pelota queda atascada al llegar al borde de la pantalla. Este problema podemos solucionarlo indicando que, cuando la pelota llega al borde de la pantalla (manejador "when (GolfBall).EdgeReached"), éstala debe rebotar (bloque de función "call (GolfBall).Bounce").





Paso 5: Contar el número de golpes.
Vamos a añadir un contador que se incrementará en una unidad con cada lanzamiento de la pelota. Por supuesto, el objetivo es colar la pelota en el hoyo con el minimo número de lanzamientos. En esta app mostraremos el número de golpes del jugador en el hoyo en el que está jugando actualmente, y también durante toda la partida.

a) En el Diseñador agregamos los siguientes componentes y fijamos sus propiedades como sigue:
   - Horizontal Arrangement: organizador para contener a las etiquetas que muestran el numero total de golpes en la partida y el número de golpes en el hoyo actual. Lo ubicamos en la parte superir de la pantalla.
   - Label1: La renombramos como "TotalScoreLabel", y se encargará de mostrar el número total de golpes en la partida. Inicializamos el valor de su propiedad "Text" a "Total strokes so far: 0"
   - Label2: La renombramos como "CurrentHoleLabel", y se encargará de mostrar el número de golpes en el hoyo actual. Inicializamos el valor de su propiedad "Text" a "Strokes in this hole: 0"
b) En el Editor de Bloques definimos dos variables globales llamadas "StrokesCounter" (contador de golpes) y "TotalScore", y las inicializamos a cero.
c) Volvemos al manejador de evento "when (GolfBall).Flung", y añadimos los bloques necesarios para incrementar el contador de golpes "StrokesCounter" en una unidad cada vez que lanzamos la pelota, y para que la etiqueta "CurrentHoleLabel" muestre el valor actual de ese contador (la figura muestra cómo actualizar el texto mostrado por la etiqueta).

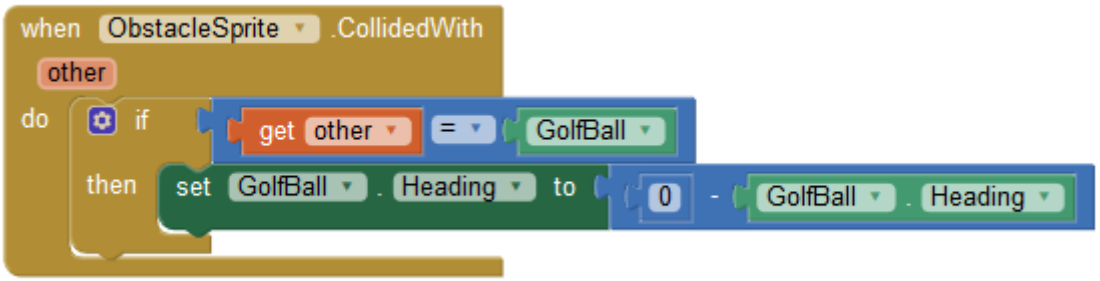
d) Ahora acudimos al manejador de evento "when (GolfBall).CollideWith" que se lanza cuando la pelota llega al agujero, y añadimos algunas funcionalidades dentro del bloque condicional: Cuando la pelota llega al agujero con la velocidad adecuada, además de fijar su velocidad a cero (esto ya está hecho), ajustamos el valor de la variable "TotalScore" a su valor previo más el número de golpes en el hoyo actual, mostramos ese valor en la etiqueta "TotalScoreLabel", reseteamos a cero el valor de la variable "StrokesCounter", y finalmente, llamamos al procedimeitno "SetupNewHole" (esto también está hecho ya).

Paso 6: Permitimos el reseteo del juego.
Ahora mismo tenemos un juego que funciona, pero lo podemos mejorar si permitimos que el usuario pueda reiniciar el juego. Mientras el usuario esté jugando le informaremos cuántolos hoyos ha completado. Para ello:
a) Añadimos los siguientes componentes en el Diseñador, y fijamos los siguientes valores para sus propiedades:
   - HorizontalArragement2: Es un organizador que contiene el botón para iniciar una nueva partida, y la etiqueta que muestra el hoyo actual.
   - Button1: Es el botón que el usuario pulsará para resetear el juego, llevándonos de vuelta al hoyo 1 y fijando la puntuación total a 0. Renombramos el componente como "NewGameButton" y fijamos su propiedad "Text" a "New Game".
   - Label3: Es una etiqueta que muestra el número de hoyo actual. Renombramos el componente a "HoleNumLabel" y fijamos su propiedad "Text" a "Hole # 1", activamos su propiedad "FontBold", y fijamos su propiedad "RextColor" a azul.
b) Definimos una nueva variable global llamada "HoleNum" para realizar el seguimiento del númer de hoyo actual, y la inicializamos a 0.
c) En el procedimiento "SetupNewHole" añadimos los bloques necesarios para actualizar el valor de la variable "HoleNum" y para mostrar en la etiqueta "HoleNumLabel" el número del hoyo actual, en formato "Hole # 1", "Hole # 2", "Hole # 3", etc.

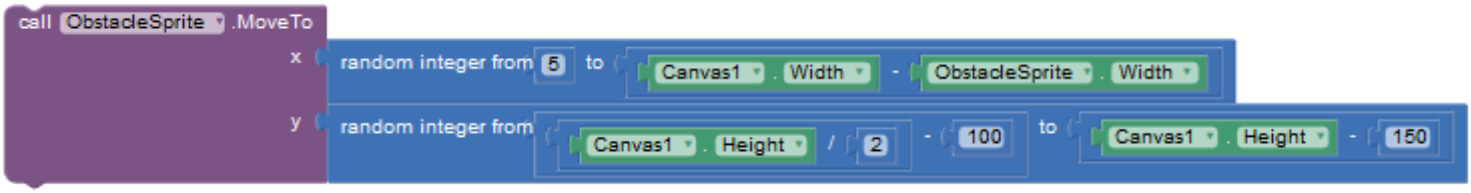
d) Ahora programamos el botón "NewGameButton". Al pulsarlo, fijamos la velocidad de la pelota a 0, fijamos el valor de la variable "totalScore" a 0, hacemos que la etiqueta "TotalScoreLabel" muestre el texto "Total strokes so far: 0", fijamos el valor de la variable "StrokesCounter" a 0, fijamos el valor de la variable "HoleNum" a 0, y finalmente, llamamos al procedimiento "SetupNewHole".

Paso 6: Añadimos un obstáculo.
La mayoría de minigolfs tienen obstáculos que hacen que la partida sea más divertida. Vamos a añadir un obstáculo rectangular simple que se colocará aleatoriamente en algún lugar entre la pelota y el hoyo. Cada nuevo hoyo el obstáculo se moverá (al igual que lo hace el propio hoyo).

a) Añade un componente "ImageSprite" al lienzo, cambia su nombre a "ObstacleSprite", sube al proyecto la imagen *obstacle.png*, y fija la propiedad "Picture" del sprite a dicha imagen.
b) Programamos el comportamiento de este obstáculo. Primero vamos a crear el código que hace rebotar la pelota cuando golpea al obstáculo. Este código simplemente modificará la dirección de la pelota para que rebote de forma natural contra el obstáculo:



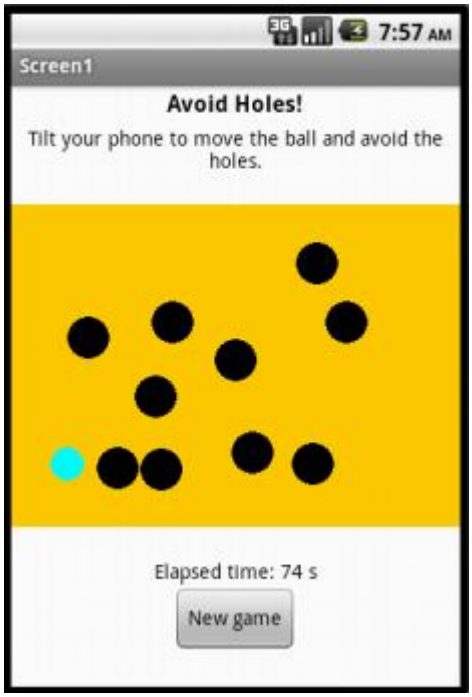
c) Además, cada vez que cambiemos de hoyo el obsáculo se colocará aleatoriamente. Para ello, acudimos al procedimiento "SetupNewHole" y agregamos el siguiente código para posicionar el obstáculo aleatoriamente:

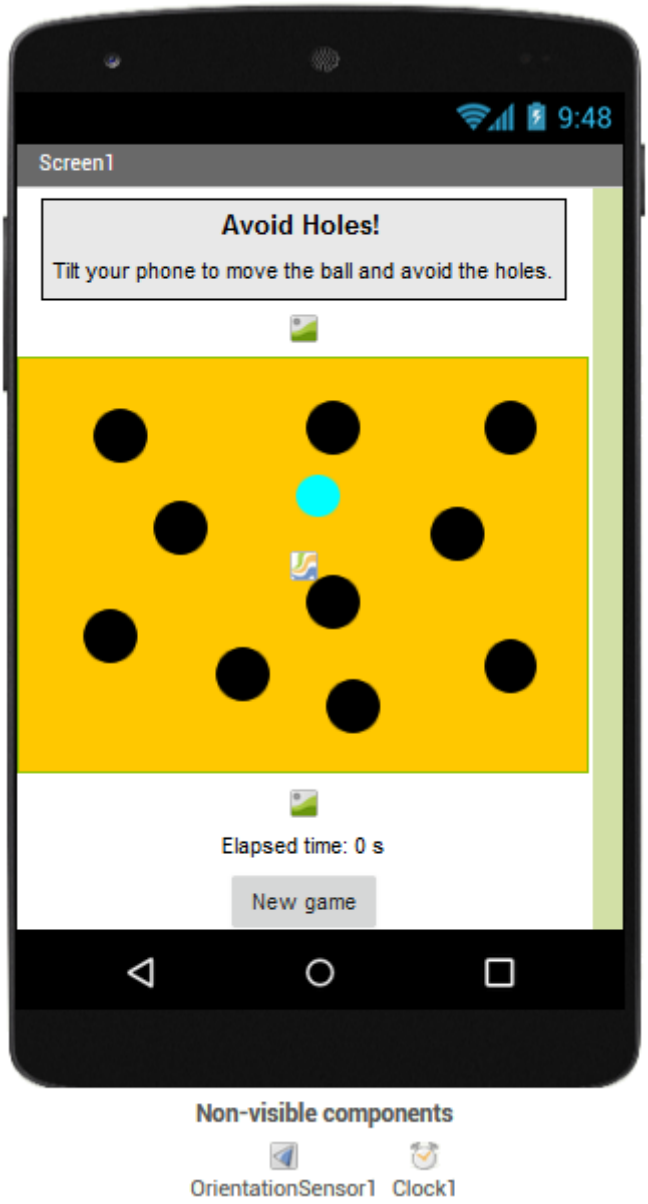


Y con esto, ya hemos terminado. Probamos la app para comprobar que todo funciona correctamente.
http://tecnoparador.blogspot.com/2018/04/ejercicos-con-app-inventor-golf.html

Ejercicio 6.2. Evita los agujeros.
En este ejercicio vas a construir una app llamada "AvoidHoles" en la que el usuario inclina el teléfono para mover una pelota a través de la pantalla. En terreno sobre el que se mueve la pelota hay un conjunto de agujeros, que el usuario debe evitar para que la pelota no caiga dentro. La puntuación del jugador vendrá dada por la cantidad de tiempo que consiga mantener la pelota en pantalla sin que ésta se cuele en uno de los agujeros.

La primera figura muestra la pantalla del Emulador para esta app, y la segunda la vista de Diseñador con todos los componentes necesarios para implementarla.





Para construir la interfaz de usuario de esta app, te recomendamos seguir los siguientes pasos:

1) Añade un componente "HorizontalArrangement" para alojar dos etiquetas llamadas "TitleLabel" (que contendrá el título "Avoid Holes!") y "IntructionsLabel" (que mostrará las instrucciones del juego, a saber, "Tilt your phone to move the ball and avoid the holes.").
2) Añade un componente "Canvas" con un fondo de color naranja. Dentro del lienzo agrega un componente "Ball" de radio 12 píxeles y color cian para implementar la pelota móvil, y otros 10 ó 12 componentes "Ball" para implementar los agujeros estáticos. Renombra a estos últimos como "Hole1", "Hole2", etc., fija su radio a 15 píxeles, y cambia su color a negro. Distribuye estos agujeros a lo largo del lienzo.
3) Bajo el lienzo, añade una etiqueta llamada "TimeLabel" que se encargará de mostrar el tiempo transcurrido (en segundos) desde el inicio de la partida, y un botón llamado "ResetButton" para comenzar una nueva partida cuando la pelota se cuele en un agujero.
4) Añade todas las imágenes vacías que creas conveniente para separar adecuadamente los distintos elementos de la interfaz de usuario.
5) Finalmente, grega los componentes no visibles "OrientationSensor" y "Clock" que necesita esta app para funcionar correctamente.

A continuación, programa el comportamiento de la app como se te indica:

a) Para esta app necesitaremos definir tres variables, que inicializaremos a cero:
   - "initialInstant" almacenará el instante en el que comienza la partida.

- "finalInstant" almacenará el iinstante actual (para llevar la cuenta del tiempo transcurrido), así como el instante en el que la partida termina, cuando la pelota se cuela en un agujero.
- "elapsedTime" almacenará el número de segundos transcurridos entre "initialInstant" y "finalInstant".

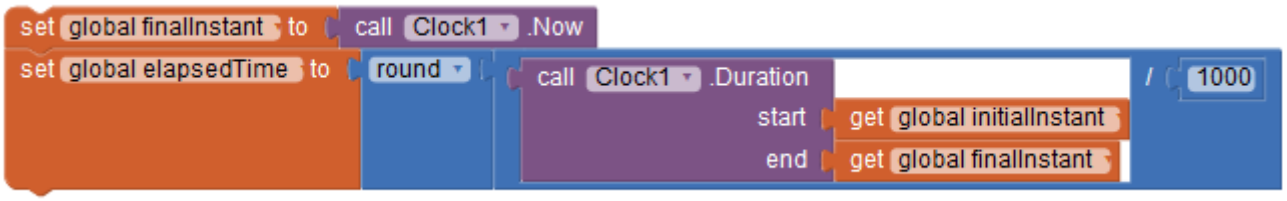
b) Creamos un procedimiento llamado "reset" que se encargue de inicializar la app. Este procedimiento debería hacer visible la pelota, ubicar a la pelota y a los agujeros en unas posiciones aleatoria sobre el lienzo, fijar a cero la rapidez y a 10 el intervalo de la pelota (para que empiece en reposo con el valor de intervalo adecuado), fijar a cero la rapidez de los agujeros (para que se mantengan inmóviles), inicializar correctamente los textos de las distintas etiquetas, habilitar los componentes "Clock" y "OrientationSensor", y guardar en "InitialInstant" el instante en el que empieza la partida (ver figura).

c) La app debe llamar al procedimiento "reset" nada más arrancar.

d) La app también debe llamar al procedimiento "reset" cuando el usuario pulse el botón "ResetButton".

e) Creamos un procedimiento llamado "MoveBall" que se encargue de mover a la pelota a cada disparo del temporizador del reloj "Clock1". El usuario controlará el movimiento de la pelota inclinando el dispositivo móvil. En la app "LadybugChase" ya implementamos un procedimiento para mover un sprite inclinnado el dispositivo móvil; este procedimiento es muy parecido.

f) La app ha de llamar al procedimiento "MoveBall" cada vez que se dispare el temporizador del reloj, cuyo intervalo habremos fijado en la pantalla de Diseño a $10\,ms$. Además de llamar al procedimiento "MoveBall" cada $10\,ms$, la app también debe registrar el instante actual en la variable "finalInstant", y guardar el número de segundos transcurridos entre el instante inicial de la partida y el instante actual en la variable "elapsedTime" (ver figura). Por último, y también a cada golpe de reloj, la app ha de mostrar en la etiqueta "TimeLabel" el número de segundos transcurridos, por ejemplo, mediante un mensaje de la forma "Elapsed time: 16 s" (donde 16 es el valor actual de la variable "elapsedTime").

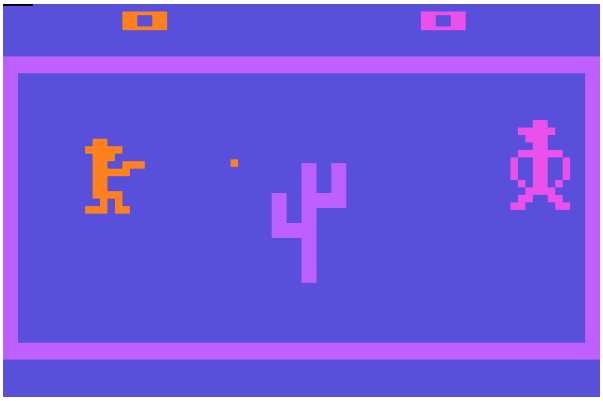


g) Creamos un procedimiento llamado "gameOver", al que la app llamará cada vez que la pelota se cuele por (colisione contra) uno de los agujeros. Este procedimiento debe efectuar múltiples tareas:

- Como en el paso anterior, el procedimiento debe registrar el instante actual en la variable "finalInstant", y guardar el número de segundos transcurridos entre el instante inicial de la partida y el instante actual en la variable "elapsedTime".
- A continuación, fijamos el color del texto de la etiqueta "TitleLabel" a rojo, y cambiamos el texto en color negro "Avoid Holes!" por el texto en color rojo "GAME OVER!". (Esto nos obligará a inicializar el color del texto del título a negro en el procedimiento "Reset", cuando empezamos mostrando el mensaje "Avoid Holes!").
- Además, el procedimiento cambia el texto de la etiqueta "InstructionsLabel" de "Tilt your phone to move the ball and avoid the holes." a un texto del tipo "You have avoided the holes during 54 s.", donde 54 es el valor actual de la variable "elapsedTime" en el momento en el que la pelota colisionó contra un agujero.
- También debemos actualizar el texto de la etiqueta "TimeLabel" al valor actual de la variable "elapsedTime", con un texto de la forma "Elapsed time: 54 s".
- Por último, la app debe deshabilitar el reloj (para que deje de contar segundos), deshabilitar el sensor de orientación (para que el usuario deje de mover la pelota, porque se ha colado en el agujero), cambiar la rapidez de la pelota a 0 (para detenerla), y hacerla invisible.

Y con esto, ya hemos terminado. Intenta mejorar la app añadiendo algún efecto de sonido al empezar la partida, al colar la bola, etc. También puedes añadir una vibración del móvil cuando la pelota se cuela por un agujero, o cuando choca contra el borde del lienzo, etc. Tienes total libertad para mejorar la app como creas conveniente.

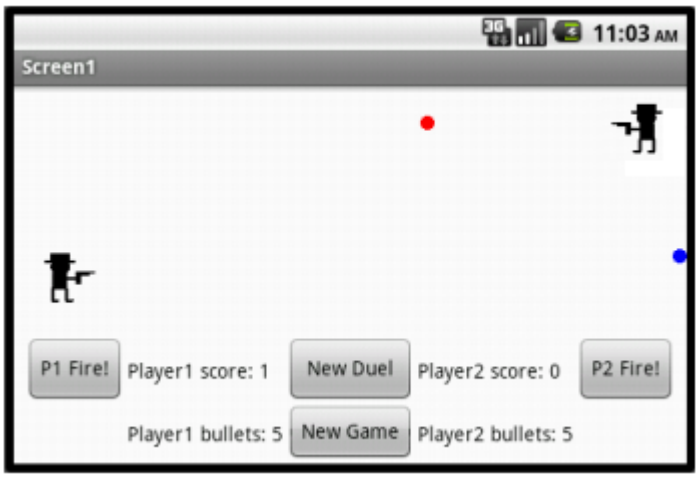
Ejercicio 6.3. Duelo de pistoleros.
Vamos a crear una versión del famoso juego de los pistoleros "Outlaw", de Atari (ver figura). En este videojuego, dos pistoleros intercambian disparos hasta que uno consigue alcanzar al otro, y gana el duelo.

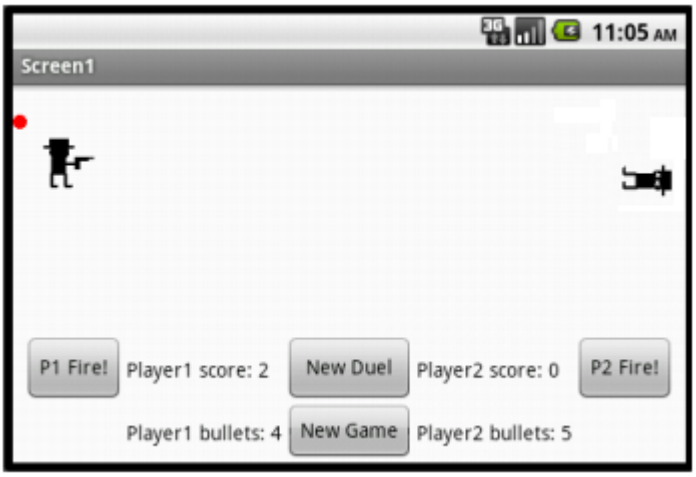


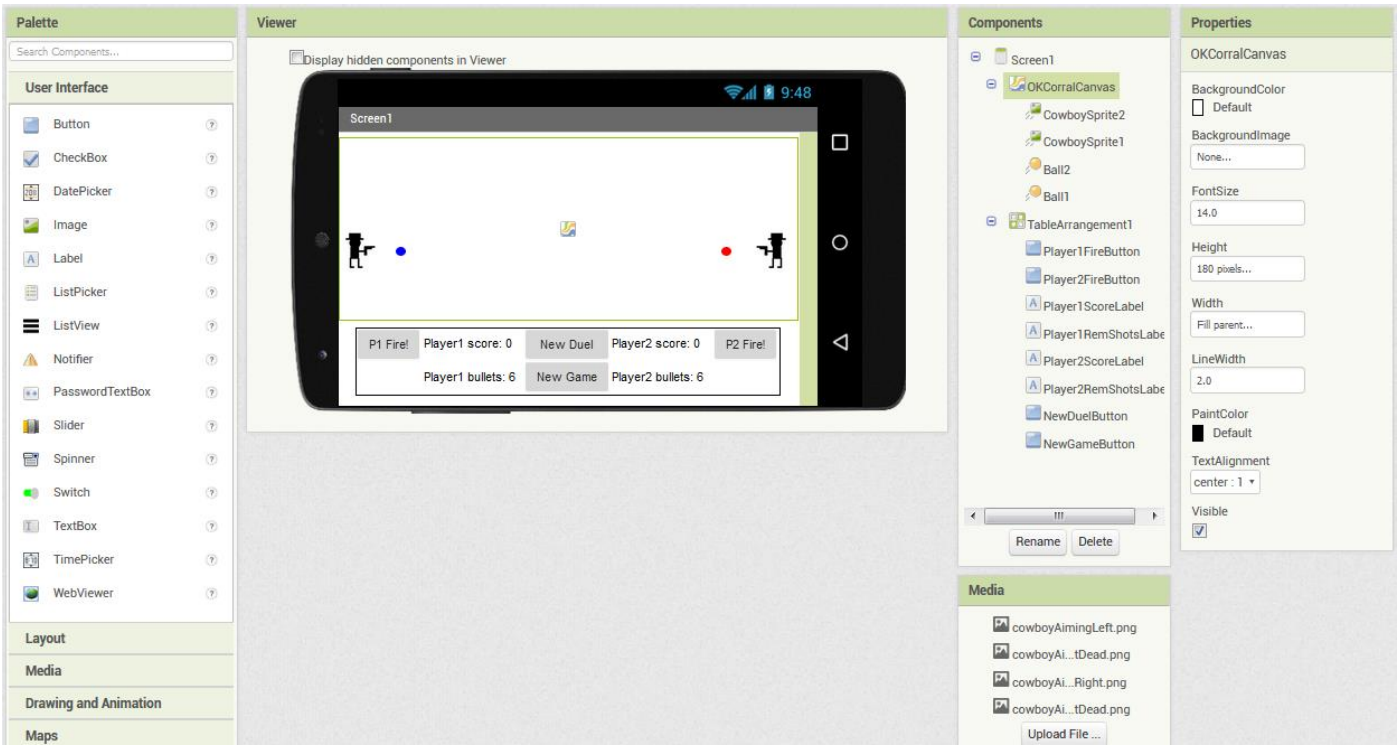
En nuestra versión del juego, que llamaremos "OKCorral", la mecánica será muy similar:

- Programaremos un videojuego para dos jugadores (Player1 y Player2) que jugarán en el mismo teléfono móvil. Cada jugador controlará uno de los pistoleros, arrastrándolo en vertical con su dedo a lo largo del extremo derecho e izquierdo de la pantalla, respectivamente.
- Cada jugador podrá disparar una de sus 6 balas al presionar el botón destinado a tal efecto. Cuando un jugador se queda sin balas, debe esperar a que el otro jugador dispare sus 6 cartuchos.
- Si el disparo alcanza al adversario, este muere y la imagen de su sprite cambia a la de un pistolero muerto. Además, el vencedor del duelo suma un tanto en su puntuación.
- Cuando un jugador muere podemos comenzar un nuevo duelo presionando el botón correspondiente. Este botón debe mantener la puntuación de los jugadores, y recargar las 6 balas en sus pistolas.
- Un botón de Nueva Partida reinicia las puntuaciones de los jugadores.

La primera figura muestra la pantalla del Emulador para esta app, y la segunda la vista de Diseñador con todos los componentes necesarios para implementarla.







Para construir esta app te recomendamos seguir estos pasos:

Paso 1: Para este proyecto necesitas cargar las imágenes de los dos pistoleros, tanto vivos como muertos. Estas imágenes se llaman *cowboyAimingLeft.png*, *cowboyAimingRight.png*, *cowboyAimingLeftDead.png*, y *cowboyAimingRightDead.png*, y todas ellas están disponibles en la carpeta de archivos de alumnos.
Paso 2: Fija la orientación de la pantalla "Screen1" a apaisada (propiedad "ScreenOrientation" a "Landscape").

Paso 3: Crea un lienzo de altura 180 píxeles y anchura igual a la de su componente padre. Sobre este lienzo ubicaremos los sprites de los pistoleros y de las balas disparadas.

Paso 4: Para representara los pistoleros en la pantalla usarás dos componentes "ImageSprite", con sus propiedades "Picture" fijadas a los archivos *cowboyAimingRight.png* y *cowboyAimingLeft.png*, respectivamente. Fija el tamaño de estos sprites a 40 píxeles de altura y de anchura.

Paso 5: Para representar las balas por pantalla usaremos dos componenets "Ball", ambos de radio 5 píxeles, pero de distinto color (rojo y azul, por ejemplo). Usamos dos balas distintas para distinguir los proyectiles disparados por el jugador 1 de los disparados por el jugador2.

Paso 6: Programa el movimiento de los pistoleros. Para ello, usa el manejador "when (ImageSprite).Dragged", de forma muy similar a como lo hicimos en la app "PoppingBalloons". Por cierto, en otro programa deberás codificar la forma en la que los sprites responden al llegar al borde del lienzo (deben rebotar sin voltearse). Recuerda: Debes programar el movimiento de los dos pistoleros.

Paso 7. Crea e inicializa las variables enecsarias para el desarrollo del juego: Dos variables para llevar la cuenta de la puntuación total de cada jugador, y otras dos variables para llevar la cuenta del número de bañas que quedan en la pistola de cada jugador. Inicialízalas al valor adecuado.

Paso 7. Programar el comportamiento de los botones de disparo: Cuando uno de los jugadores pulse su botón de disparo, habilitamos y hacemos visible la bala que haya disparado, reducimos el número de balas en su pistola en una unidad, y mostramos esta información por pantalla en la etiqueta correspondiente. Para que la bala salga disparada, debemos llevarla a la posición (x, y) del pistolero que ha apretado el gatillo, y

moverla hacia el otro extremo fijando los valores adecuados para sus propiedades "Heading", "Interval", y "Speed". Recuerda: Debes programar el comportamiento del botón de disparo de ambos jugadores.

Paso 8: Programar el impacto de la bala contra el otro jugador: Si la bala impacta contra el jugador contrario, debemos cambiar su imagen por la de un pistolero muerto (*cowboyAimingRightDead.png* y *cowboyAimingLeftDead.png*, respectivamente), aumentar en una unidad la puntuación del jugador que ha vencido, y mostrar la nueva puntuación por pantalla en la etiqueta correspondiente. También debemos deshabilitar las dos balas, y opcionalmente, hacerlas invisibles. Recuerda: Debes programar el posible impacto de ambas balas contra el pistolero adecuado.

Paso 9: Programar el comportamiento del botón de Nuevo Duelo: Cuando cualquiera de los jugadores pulsa el botón de Nuevo Duelo, ambos jugadores vuelven a sus posiciones iniciales ($x_1 = 0,\ y_1 = 90,\ z_1 = 1$ y $x_2 = 410,\ y_2 = 90,\ z_2 = 1$)[9], se recargan las 6 balas en sus pistolas, se vuelven a poner las imágenes de los pistoleros vivos para ambos sprites, las balas vuelven a habilitarse, y se hacen invisibles. Las puntuaciones de los jugadores deben mantenerse inalteradas. En este caso es recomendable construir un procedimiento de "Reset" que se encargue de realizar todas estas tareas, y al que llamemos dentro del manejador de eventos del botón.

Paso 10: Programar el comportamiento del botón de Nueva Partida: Al presionar el botón de Nueva Partida ocurre exactamente lo mismo que al pulsar el botón de Nuevo Duelo, solo que en este caso, las puntuaciones de los jugadores se restablecen a cero.

> NOTA: El control del movimiento de los pistoleros no tiene por qué hacerse arrastrando el dedo por la pantalla. Otras alternativas es hacerlo mediante botones (un botón para subir y otro para bajar), mediante deslizadores, etc. Así mismo, el control del disparo tampoco tiene por qué ser obligatoriamente con un botón. Otros enfoques podrían ser tocar el sprite del pistolero (si su movimiento se controla con botones), deslizar el dedo rápidamente sobre el pistolero (con lo cual incluso podríamos dirigir la bala en una dirección distinta a la horizontal), etc.
>
> A modo de ejercicio, te proponemos que cambies el control del movimiento vertical de los pistoleros para efectuarlo con dos deslizadores, y que modifiques el control del disparo para hacerlo tocando el sprite correspondiente.

Para terminar, añade algunas mejoras estéticas al videojuego, como un fondo para el lienzo, una serie de sonidos para los disparos, las muertes de los pistoleros, o los impactos de los proyectiles contra el borde del lienzo, una breve espera temporizada antes de iniciar cada nuevo duelo (para que el duelo no empiece de forma súbita), etc.

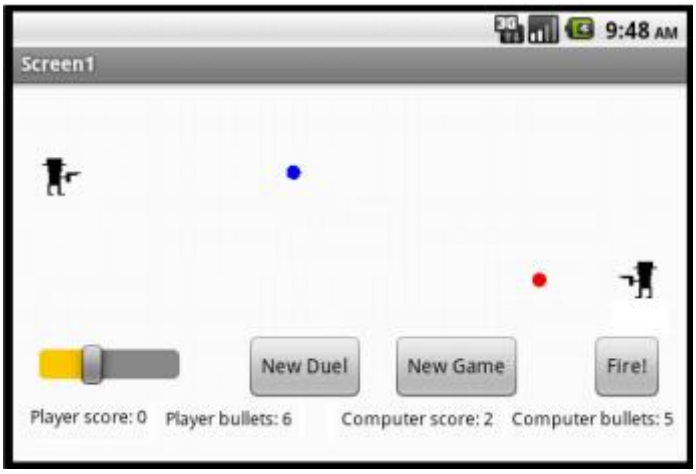
Ejercicio 6.4. Duelo de pistoleros para un jugador.
Si has terminado el programa previo, verás que presenta un (grave) problema: Los dos jugadores no pueden arrastrar sus dedos simultáneamente por la pantalla del teléfono para mover sus pistoleros al mismo tiempo. Las pantallas de los teléfonos actuales no soportan un doble arrastre de dedos, y solo se mueve uno de los pistoleros.

[9] Una mejor forma de fijar las coordenadas iniciales, en vez de fijarlas directamente con números, es referirnos a la altura y a la anchura del lienzo. Por ejemplo, las coordenadas y de ambos jugadores pueden obtenerse dividiendo la altura total de lienzo (propiedad "(Canvas1).Height"), que la hemos fijado a 180 píxeles, entre 2. También podríamos obtener la coordenada x_2 restándole, digamos 20 píxeles, a la anchura del lienzo (propiedad "(Canvas1).Width"). Opcionalemnet, también podríamos optar por fijar una coordenada y aleatoria para ambos jugadores, dentro del rango de 0 a 180 (ya que 180 es la altura del lienzo)

Hay dos formas de solventar este problema:

a) Podemos hacer que cada jugador juegue con su teléfono móvil, para lo cual habría que poner en comunicación ambos teléfonos (por WiFi, Bluetooth, etc.) para actualizar en la pantalla de un jugador los movimientos del jugador contrario. Para esto no estamos preparados *todavía*.
b) Podemos cambiar la app para convertirla en un juego en la que un solo jugador compita contra la máquina (el teléfono). Este es el objetivo de este ejercicio.

Guarda una copia del programa previo, y modifica la app para permitir que sea el dispositivo quien controle los movimientos y los disparos del pistolero de la derecha. En este caso, el jugadar controlará al pistolero de la izquierda usando un deslizador y un botón de disparo (ver figura).

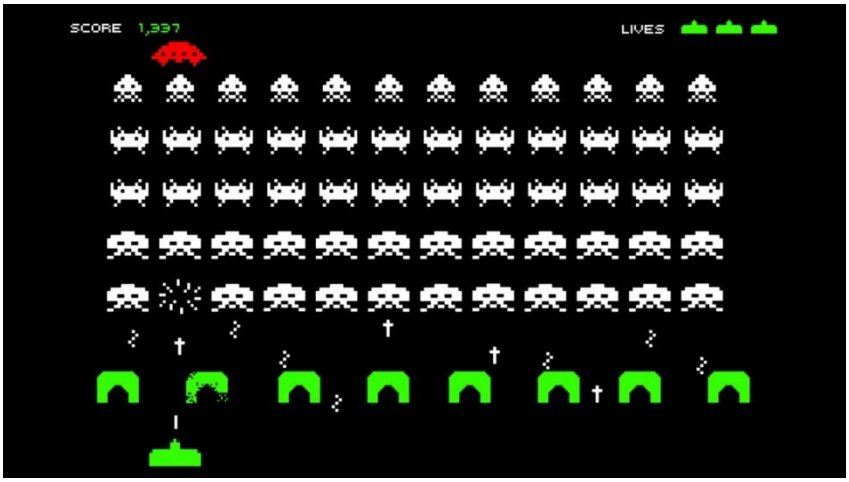


La mecánica del juego es exactamente igual que antes, solo que ahora deberás programar la "inteligencia artificial" que le permita a la app controlar automáticamente al pistolero de la derecha. Algunos consejos:

- Para mover y hacer disparar periódicamente al pistolero automático, necesitarás uno o dos relojes (y sus temporizadores correspondientes).
- Aleatoriza el movimiento vertical del pistolero automático de forma que cambie de dirección (arriba o abajo) con una cierta probabilidad, tal y como lo hicimos con la rana y la pulga en la app "LadybugChase".
- Controla el disparo del pistolero automático para que dispare siempre que su coordenada y sea similar a la coordenada y del pistolero izquierdo. Para que no dispare siempre que ambos pistoleros se crucen, haz que el pistolero solo dispare con cierta probabilidad.
- Para que el pistolero automático no vacíe su cargador nada más cruzarse con el pistolero del jugador, inhabilita la acción de volver a disparar hasta que el proyectil no haya impactado contra el borde izquierdo del escenario o alcanzado al pistolero del jugador. Para ello puedes desactivar el temporizador que controla los disparos (pero acuérdate de volver a habilitarlo después).

Ejercicio 6.5. Invasores del espacio.

Como ejercicio opcional, te proponemos que programes tu versión del videojuego clásico Space Invaders (ver figura).



En este videojuego, el jugador controla una nave en la parte inferior de la pantalla, mientras desde la parte superior se le van aproximando oleadas de naves alienígenas enemigas. Las naves alienígenas van moviéndose dehorizontalmente y descendiendo progresivamente hacia el jugador, y opcionalmente, también le disparan proyectiles. El objetivo del jugador es acabar con todas las naves enemigas, moviéndose horizontalmente en la parte baja de la pantalla, y disparando contra ellas. (ver vídeo: https://www.youtube.com/watch?v=MU4psw3ccUI).

Por supuesto, no se te exige que hagas una versión idéntica (aunque podrías), sino que hagas tu propio tributo en App Inventor para este juego. Por ejemplo, haz una versión con 5 o 10 enemigos, sin parapetos de cobertura, sin vidas, etc. Dispones de total libertad para hacerlo, pero cuanto más completo, mejor.

7. TEXTO A VOZ, MENSAJERÍA, Y ALMACENAMIENTO PERSISTENTE.

Los objetivos de este capítulo son múltiples: (1) Entender cómo almacenar información en el dispositivo de forma persistente usando el componente "TinyDB", (2) aprender a gestionar la recepción y el envío de mensajes de texto SMS con el componente "Texting", y (3) hacer que nuestras apps sean capaces de leer un texto en voz alta con el componente "TextToSpeech", y conviertir a texto un mensaje hablado con el componente "SpeechRecognizer".

7.1. BASES DE DATOS.

¿QUÉ ES UNA BASE DE DATOS?

Facebook tiene una base de datos con la información de la cuenta de cada usuario, su lista de amigos, y sus publicaciones. Amazon usa una base de datos que incluye todo lo que se puede comprar. Google mantiene una base de datos con información acerca de cada página presente en la Web. Y aunque no a semejante escala, casi cualquier aplicación que podamos crear necesitará interactuar con una base de datos. Una **base de datos** no es más que una colección de datos relacionados que pueden guardarse y recuperarse de forma ágil y rápida.

En la mayoría de entornos de prograación, la construcción de una app que pueda comunicarse con una base de datos requiere de técnicas de programación avanzadas, como levantar un servidor con un software como Oracle o MySQL, y escribir el código que interactúa con esa base de datos. Afortunadamente, App Inventor proporciona una serie de componentes que convierten la programación de bases de datos en simples operaciones de guardar y recuperar datos. Con estos componentes podemos crear aplicaciones que guarden los datos directamente en el dispositivo Android, y con algunas configuraciones adicionales, también podemos desarrollar aplicaciones que compartan datos con otros dispositivos y usuarios, almacenando esos datos en una base de datos centralizada en la Web.

Los datos que guardamos en las variables y en las propiedades de los componentes son *temporales*: Si el usuario almacena información y a continuación cierra la aplicación antes de haberla guardado en una base de datos, la información se habrá perdido cuando se vuelva a abrir la app. Para guardar información de forma *persistente*, debemos almacenarla en una base de datos. Se dice que la información guardada en bases de datos es **persistente** porque, cuando cerramos una app y volvemos a abrirla, los datos siguen estando disponibles.

A modo de ejemplo, consideremos la app "NoTextingWhileDriving" que construiremos a continuación. Esta app manda una respuesta automática cuando el terminal recibe un mensaje de texto SMS. La app comienza con un mensaje de respuesta por defecto, pero también le permite al usuario escribir una respuesta personalizada. Si el usuario cambia el mensaje de respuesta y a continuación cierra la app, la próxima vez que la abra probablemente querrá que la respuesta automática siga siendo la que escribió, y no la respuesta por defecto. Por consiguiente, la respuesta personalizada debe almacenarse en una base de datos, para que la próxima vez que el usuario abra la app, la respuesta pueda recuperarse de la base de datos.

ALMACENAR DATOS PERSISTENTES EN UNA BASE DE DATOS "TINYDB".

App inventor proporciona dos componentes para facilitar el trabajo con bases de datos: Los componentes "TinyDB" y "TinyWebDB".

El componente "TinyDB" sirve para almacenar datos persistentes directamente en el dispositivo Android. Este componente es útil en las apps personales en las que el usuario no necesita compartir datos con otros dispositivos o personas, como es el caso de la app "NoTextingWhileDriving". Por otro lado, el componente "TinyWebDB" permite almacenar datos persistentes en una base de datos en la Web para poder compartirlos entre múltiples dispositivos. El hecho de poder acceder a los datos de una base de datos en la Web es esencial para los juegos multijugador y para las apps en las que los usuarios pueden insertar y compartir información, como en el caso de la app "MakeQuiz" que desarrollaremos en el capítulo 9.

Ambos componentes son similares, pero el componente "TinyDB" es un poco más sencillo, y por esa razón es el primero que presentaremos aquí. El componente "TinyDB" no necesita levantar una base de datos; los datos se almacenan en una base de datos que se construye directamente en el dispositivo, y que queda asociada a nuestra app.

Para transferir datos a la memoria a largo plazo del dispositivo usamos el bloque de función "(TinyDB).StoreValue". La figura muestra la forma en la que la app "NoTextingWhileDriving" guarda de forma persistente la respuesta personalizada escrita por el usuario:

Las bases de datos "TinyDB" utilizan un mecanismo de asociación etiqueta a valor para el almacenamiento de los datos. En el programa de la figura, el valor almacenado en la base de datos es la respuesta automática personalizada que el usuario escribe en una caja de texto, como por ejemplo, "Estoy en una reunión, llámame en un par de horas". Dicho valor se almacena con la etiqueta "responseMessage".

La etiqueta (parámetro "tag") sirve para identificar con un nombre el dato almacenado en la base de datos, a modo de mecanismo para poder referenciar esa información. El valor (parámetro "value") es el dato que se almacena. Así, podemos imaginar que una base de datos "TinyDB" es como una tabla de parejas etiqueta – valor. La etiqueta es el nombre identificativo del objeto a almacenar, y el valor es el objeto almacenado. Después de que se ejecute el bloque "(TinyDB1).StoreValue" de la figura, la base de datos del dispositivo contendrá los datos listados en la tabla a continuación:

Etiqueta	Valor
"responseMessage"	"Estoy en una reunión, llámame en un par de horas."

Una app podría almacenar muchas parejas etiqueta – valor para los distintos datos que deban guardarse de forma persistente. La etiqueta siempre es un texto, mientras que el valor puede ser un objeto individual (como un texto, un número, o una imagen), o tal vez una lista de objetos. Cada etiqueta identifica un único valor, y cada vez que asociamos un nuevo valor a una etiqueta ya existente, el valor reemplazado es sobreescrito y desaparece

RECUPERAR DATOS DE UNA BASE DE DATOS TINYDB.

Para recuperar datos persistentes de una base de datos "TinyDB" usamos el bloque de función "(TinyDB).GetValue". Cuando llamamos a esta función, solicitamos la recuperación de unos datos en particular proporcinando una etiqueta. Para la app "NoTextingWhileDriving" podemos solicitar la respuesta personalizada del usuario usando la misma etiqueta que usamos en el bloque "(TinyDB).StoreValue", esto es, la etiqueta "responseMessage". La llamada a la función "GetValue" devolverá los datos guardados bajo esa etiqueta, por lo que deberemos almacenarlos en una variable.

Es muy habitual tener que recuperar los datos almacenados en una base de datos nada mas abrir la app. Como sabemos, App Inventor proporciona un manejador especial, "(Screen1).Initialize", que se dispara justo cuando arranca la app. En tales casos, debemos tener la precaución de considerar la situación en la que todavía no existan datos en la base de datos (como ocurre, por ejemplo, la primera vez que se usa la app). Cuando llamamos a la función "GetValue" debemos especificar un parámetro "valueIfTagNotThere", que es el valor que se devuelve si no hay datos almacenados bajo esa etiqueta.

La figura muestra los bloques utilizados en el manejador "(Screen1).Initialize" de la app "NoTextingWhileDriving", e indican la forma en la que muchas bases de datos cargan los datos durante la fase de inicialización de la aplicación:

Estos bloques ponen los datos devueltos por la función "GetValue" en la etiqueta "ResponseLabel". Si ya había datos en la base de datos, estos datos se cargan en la etiqueta "ResponseLabel". Pero si no hay datos asociados a la etiqueta "responseMessage", o si esa etiqueta no existe, la función carga el valor "valueIfTagNotThere" (el texto "I'm drivng right now, I'll text you later") en la etiqueta "ResponseLabel".

7.2. CONVERSIÓN DE TEXTO A VOZ (Y VICEVERSA).

En App Inventor distinguimos dos tipos de conversiones de texto a voz:

SERVICIO	USUARIO/PROGRAMADOR	APP INVENTOR	¿NECESITA CONEXIÓN?
Texto a voz (componente "TextToSpeech")	Escribe un texto	Lee el texto en voz alta	No
Voz a texto (componente "SpeechRecognizer")	Habla en voz alta	Usa Google Voice para convertir a texto	Sí

El servicio de reconocimiento de voz (conversión de voz a texto) ha experimentado un gran desarrollo en los últimos años, aunque está lejos de ser perfecto. El servicio de conversión de texto a voz es más fiable y efectivo, pero adolece de algunas pequeñas anomalías. Sin embargo, ambos servicios pueden resultarnos útiles para construir apps muy interesantes.

El ejemplo típico de conversión de texto a voz sería una app en la que asociásemos un botón a una etiqueta de texto, y cuando el usuario presionase el botón, la app leyese el texto en voz alta. Pero también podríamos permitir que el usuario escribiese el texto a leer en un cuadro de texto, o incluso podríamos hacer que la app leyese el texto de un SMS recibido en el dispositivo. Éste es precisamente el objetivo de la primera app que construiremos en este capítulo.

7.3. COMENZAR CON LA APP "NOTEXTINGWHILEDRIVING1".

Vamos a desarrollar una app de "manos libres" que lea el contenido de un mensaje de texto que hayamos recibido en nuestro dispositivo móvil mientras conducimos, y que lo responda automáticamente con otro mensaje de texto de tipo "Estoy conduciendo, responderé en breve". Esta app nos enseñará a usar una de las características más interesantes de los dispositivos Android: El servicio de mensajería de textos cortos (SMS).

Para empezar, abrimos el navegador para conectarnos a la web de App Inventor, y empezamos un proyecto nuevo al que llamaremos "NoTextingWhileDriving1". (En capítulos posteriores construiremos una segunda versión que incluya información de localización). Establecemos el título de la pantalla a "No Texting While Driving". Nos conectamos al dispositivo Android o al emulador para iniciar el proceso de pruebas en vivo.

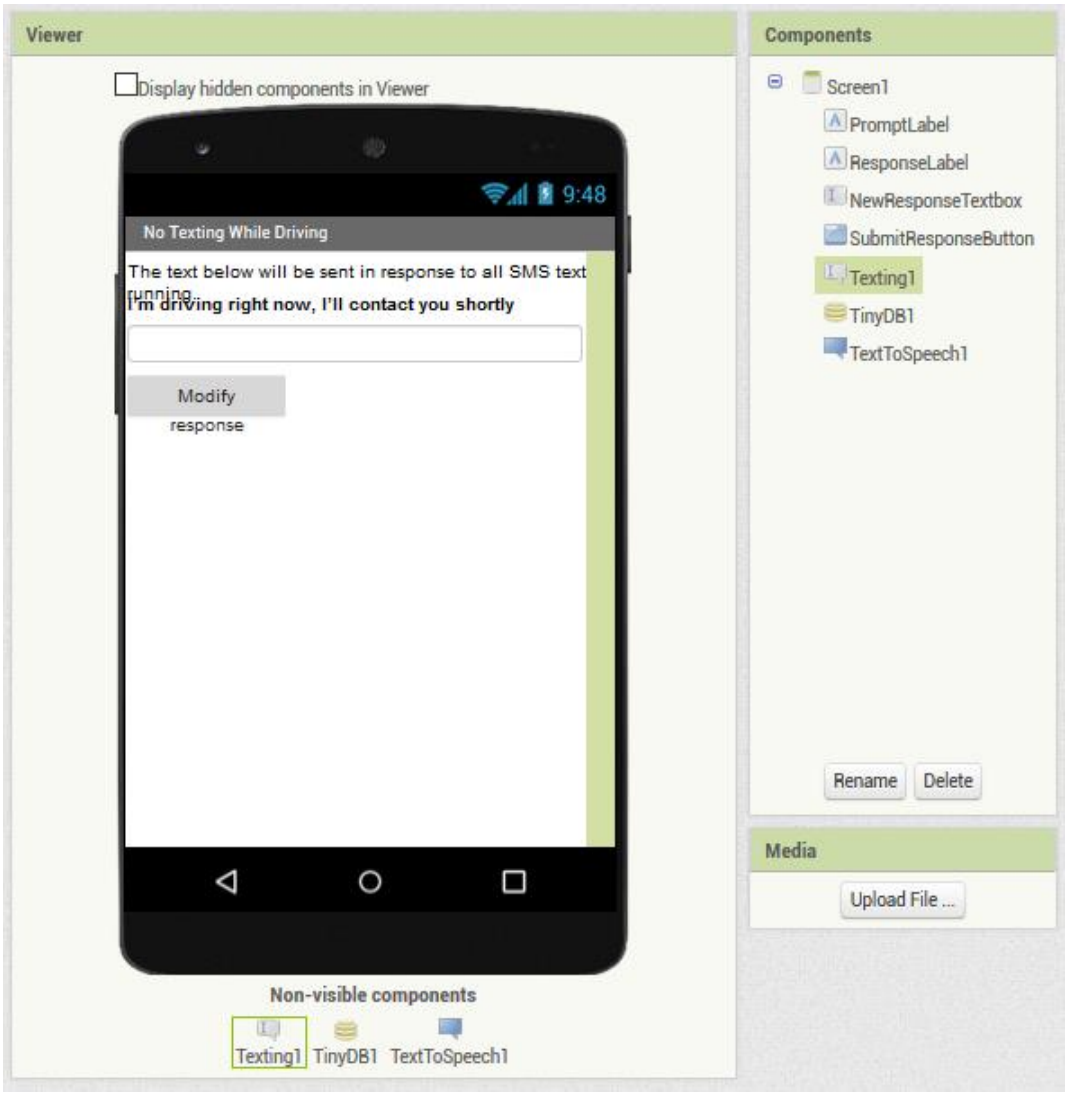
7.4. DISEÑAR LOS COMPONENTES DE "NOTEXTINGWHILEDRIVING1".

La interfaz de usuario de la app es bastante simple, como vemos en la figura que muestra la app en el Emulador. Tiene una etiqueta que le indica al usuario cómo funciona la app, una etiqueta que muestra el texto que se enviará automáticamente como respuesta a los mensajes de texto entrantes, una caja de texto para cambiar la respuesta, y un botón para registrar ese cambio. También necesitaremos añadir un componente "Texting", un componente "TinyDB", y un componente "TextToSpeech", todos los cuales aparecerán en la zona de componentes no visibles. El único componente nuevo en esta app es el componente "Texting", que sirve para enviar mensajes de texto y procesar los mensajes de texto recibidos.

La tabla lista todos los componentes necesarios para la app, la bandeja donde localizarlos, el nombre que les adjudicaremos, y la misión que cumplen en nuestro proyecto.

Component type	Palette group	What you'll name it	Purpose
Label	User Interface	PromptLabel	Let the user know how the app works.
Label	User Interface	ResponseLabel	The response that will be sent back to the sender.
TextBox	User Interface	newResponseTextBox	The user will enter the custom response here.
Button	User Interface	SubmitResponseButton	The user clicks this to submit response.
Texting	Social	Texting1	Process the texts.
TinyDB	Storage	TinyDB1	Store the response in the database.
TextToSpeech	Media	TextToSpeech1	Speak the text aloud.

La figura muestra una captura de pantalla de la vista de Diseño con todos los componentes agregados.



Podemos construir esta interfaz de usuario arrastrando todos los componentes indicados en la tabla, y ajustando los siguientes valores para sus propiedades:

- Fijamos la propiedad "Text" del componente "PromptLabel" a "The text below will be sent in response to all SMS texts received while this app is running".
- Fijamos la propiedad "Text" de "ResponseLabel" a "I'm driving right now, I'll contact you shortly". Activa también su propiedad "FontBold".
- Fijamos la propiedad "Text" de "NewResponseTextbox" a " ". (Con esto dejaremos la caja de texto vacía hasta que el usuario escriba algo).
- Fijamos la propiedad "Hint" de "NewResponseTextbox" a "Enter new responde text".
- Fijamos la propiedad "Text" de "NewResponseButton" a "Modify response".

7.5. AÑADIR COMPORTAMIENTOS A LOS COMPONENTES DE "NOTEXTINGWHILEDRIVING1".

Comenzaremos programando el comportamiento de la respuesta automática, para enviar un texto de respuesta a cualquier mensaje de texto recibido. A continuación añadiremos los bloques que le permitan al usuario cambiar el texto de la respuesta automática a un texto personalizado, y que éste quede guardado de forma indefinida en una base de datos. Finalmente escribiremos el código para leer en voz alta los mensajes de texto entrantes.

USAR EL COMPONENTE "TEXTING" PARA ENVIAR UNA RESPUESTA AUTOMÁTICA A UN MENSAJE ENTRANTE.

Para programar la respuesta automática usaremos el componente "Texting". Podemos imaginar que este componente es como un autómata dentro de nuestro teléfono que es capaz de leer y de escribir textos. Para gestionar los mensajes recibidos, este componente proporciona el manejador de evento "when (Texting1).MessageReceived". Tomando este bloque de la bandeja "Texting1" y añadiéndole otros bloques dentro de su sección "do" podemos especificar qué ocurrirá cuando el dispositivo reciba un mensaje de texto. En nuestro caso, queremos enviar automáticamente un texto de respuesta.

Para enviar un texto necesitamos tres bloques: En primer lugar, debemos indicar el número de teléfono al que queremos enviar el texto, lo que se especifica en una propiedad del componente "Texting1". Después indicamos el mensaje a enviar, lo que también se indica en una propiedad de "Texting1". Finamente, enviamos el mensaje con el bloque "call (Texting1).SendMessage". La tabla lista los bloques necesarios para esta respuesta automática, y la figura muestra cómo deben quedar en el Editor de Bloques.

Block type	Drawer	Purpose
`Texting1.MessageReceived`	Texting	The event handler that is triggered when the phone receives a text.
`set Texting1.PhoneNum ber to`	Texting	Set the `PhoneNumber` property before sending.
`value number`	Drag from when block	The phone number of the person who sent the text.
`set Texting1.Message to`	Texting	Set the `Message` property before sending.
`ResponseLabel.Text`	ResponseLabel	The message the user has entered.
`Texting1.SendMessage`	Texting	Send the message.

¿Cómo funciona este programa? Cuando nuestro teléfono recibe un mensaje de texto, se lanza el evento "(Texting1).MessageReceived". El número de teléfono del emisor del mensaje se guarda en el argumento "number", y el contenido del mensaje de texto se localiza en el argumento "messageText".

Como lo que queremos es enviar el texto de la respuesta automática al emisor del mensaje recibido, fijamos el valor de "Texting1.PhoneNumber" al valor contenido por el argumento "number". Y como el texto que queremos enviar como respuesta automática está en la propiedad "Text" del componente

"ResponseLabel" (a saber, "I'm driving right now, I'll contact you shortly."), fijamos el valor de "Texting1.Message" a "ResponseLabel.Text". Una vez fijamos estos dos valores, la app llama a la función "Texting.SendMessage" para enviar la respuesta.

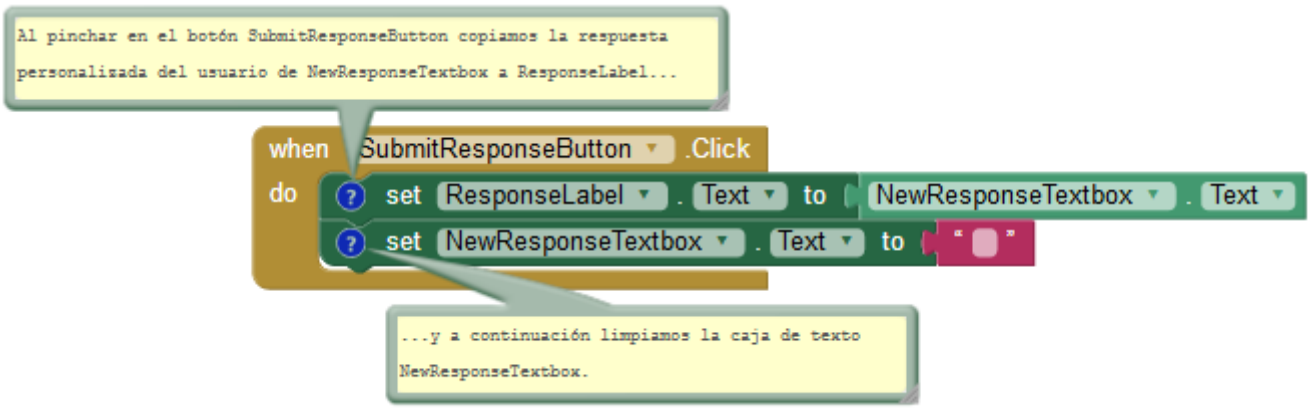
Vamos a probar nuestra app. Para ello necesitaremos dos teléfonos móviles, uno que esté ejecutando la app, y otro que envíe el mensaje de texto inicial. Si no disponemos de un segundo teléfono móvil, podemos usar en el ordenador Google Voice (o un servicio similar) para enviar un mensaje de texto al teléfono que está ejecutando la app. Comprobamos que el primer teléfono (el que envía el mensaje en primer lugar) recibe el texto de la respuesta automática.

ESCRIBIR UNA RESPUESTA PERSONALIZADA.

A continuación vamos a escribir el programa que le permita al usuario escribir su propia respuesta personalizada. En el Diseñador ya añadimos un componente "TextBox" llamado "NewResponseTextbox". Aquí es donde el usuario escribirá su respuesta personalizada. Cuando el usuario pinche en el botón "SubmitResponseButton" copiaremos el texto escrito en "NewResponseTextbox" a "ResponseLabel", etiqueta cuyo texto se utiliza como contenido del mensaje de respuesta automática. La tabla lista todos los bloques que necesitaremos para transferir el texto escrito en "NewResponseTextbox" a "ResponseLabel".

Block type	Drawer	Purpose
SubmitResponseButton.Click	SubmitResponseButton	The user clicks this button to submit a new response message.
set ResponseLabel.Text to	ResponseLabel	Move (set) the newly input value to this label.
NewResponseTextbox.Text	NewResponseTextbox	The user has entered the new response here.
set NewResponseTextbox.Text to	NewResponseTextbox	Blank out the text box after transferring information
text ("")	Text	The empty text.

La figura muestra cómo debería quedar el programa:



¿Cómo funciona este programa? Piensa en cómo interactuamos con un típico formulario para la entrada de datos: Primero, escribimos algo en la caja de texto del formulario, y a continuación, presionamos un botón de "Enviar" para indicarle al sistema que procese el texto que hemos escrito. El formulario de entrada de esta app funciona exactamente igual. La figura previa muestra el programa que permite que, cuando el usuario pinche en el botón "SubmitResponseButton", se active el evento "SubmitResponseButton.Click". En ese caso, el manejador del evento copia el texto que el usuario ha escrito en "NewResponseTextbox" a

"ResponseLabel". Recordemos que "ResponseLabel" guarda el mensaje que se enviará automáticamente a modo de respuesta, por lo que debemos asegurarnos de que esta etiqueta guarde ahora el mensaje personalizado escrito por el usuario.

Comprobamos el funcionamiento de la app. Escribimos un mensaje personalizado y pinchamos en el botón para procesarlo. A continuación, usamos un segundo teléfono móvil para enviar un mensaje de texto al móvil donde se ejecuta la app. Debemos comprobar que el móvil que envía el mensaje recibe un mensaje automático con la nueva respuesta personalizada escrita por el usuario.

GUARDAR LA RESPUESTA PERSONALIZADA DE FORMA INDEFINIDA.

Ahora mismo el usuario puede personalizar el texto de la respuesta automática, pero la app tiene un problema: Si el usuario escribe una respuesta personalizada, cierra la app, y vuelve a abrirla, esa respuesta personalizada ya no estará disponible (en su lugar, aparecerá la respuesta por defecto). Este comportamiento no es el que los usuarios esperarán de la app; en vez de eso, preferirán que la respuesta personalizada que ya escribieron siga disponible al reiniciar la app. Para que esto ocurra, debemos guardar la respuesta personalizada de forma **persistente** (esto es, de forma indefinida).

Al poner datos en la propiedad "Text" de "ResponseLabel", técnicamente estamos guardando esos datos. Pero el hecho es que los datos que guardamos en las propiedades de los componentes o en variables son **datos transitorios**. Los datos transitorios son como nuestra memoria a corto plazo: el teléfono los "olvida" en cuanto se cierra la app. Si queremos que nuestra app recuerde algo de forma persistente, debemos transferir esos datos desde la memoria a corto plazo (una propiedad o una variable) a una memoria a largo plazo (una base de datos o un archivo).

Para almacenar datos de forma persistente en App Inventor, podemos usar un componente "TinyDB", que permite guardar datos en un archivo en el dispositivo Android. El componente "TinyDB" proporciona dos funciones: "StoreValue" y "GetValue". Con la primera la app puede guardar información en la base de datos del dispositivo, y con la segunda la app puede recuperar información que haya sido guardada previamente.

Para la mayoría de aplicaciones, el proceso será el siguiente:

1) Guardamos datos en la base de datos siempre que el usuario envíe un nuevo valor.
2) Al iniciar la app, cargamos los datos desde la base de datos a la variable o propiedad encargada de albergarlos.

Vamos a empezar modificando el manejador de evento "when (SubmitResponseButton).Click" para que guarde la respuesta personalizada de forma persistente, usando los bloques listados en la tabla a continuación:

Block type	Drawer	Purpose
`TinyDB1.StoreValue`	TinyDB1	Store the custom message in the phone's database.
`text ("responseMessage")`	Text	Use this as the tag for the data.
`ResponseLabel.Text`	ResponseLabel	The response message is now here.

La figura a continuación muestra cómo debería quedar el programa:

RECUPERAR LA RESPUESTA PERSONALIZADA AL ARRANCAR LA APP.

La razón por la que hemos guardado la repuesta personalizada del usuario en una base de datos es para poder cargarla la siguiente vez que el usuario arranque la app. App Inventor proporciona un manejador de evento especial que se activa siempre que la app se abre, a saber, el bloque "when (Screen1).Initialize". Si ubicamos bloques dentro de la sección "do" de este manejador de evento, esos bloques se ejecutarán inmediatamente después de que la app se haya lanzado.

En esta app, el manejador de evento "when (Screen1).Initialize" cargará la respuesta personaliza del usuario recuperándola de la base de datos con la llamada a la función "TinyDB.GetValue". Los bloques que necesitaremos están listados en la tabla, y el aspecto que debería tener el programa se muestra en la figura.

Block type	Drawer	Purpose
`Screen1.Initialize`	Screen1	This is triggered when the app begins.
`TinyDB1.GetValue`	TinyDB1	Get the stored response text from the database.
`text` ("responseMessage")	Text	Plug this into the `tag` socket of `TinyDB.GetValue`, making sure the text is the same as that used in `TinyDB.StoreValue` earlier.
`text` ("I'm driving right now, I'll contact you shortly")	Text	Plug this into the `valueIfTagNotThere` slot of `TinyDB.GetValue`. This is the default message that should be used if the user has not yet stored a custom response.
`set ResponseLabel.Text to`	ResponseLabel	Place the retrieved value in `ResponseLabel`.

¿Cómo funciona el programa? Imaginar que el usuario abre la app por primera vez, escribe una respuesta personalizada, y a continuación cierra y abre la app varias veces. La primera vez que el usuario abre la app no hay ninguna respuesta personalizada en la base de datos, por lo que queremos es dejar en "ResponseLabel" la respuesta por defecto. Las siguientes veces que el usuario lance la app queremos cargar la respuesta personalizada que el usuario escribió y guardó previamente en la base de datos, y dejarla en "ResponseLabel".

Cuando la app comienza se activa el evento "Screen1.Initialize". Entonces, la app llama a la función "TinyDB1.GetValue" con la etiqueta "ResponseMessage", esto es, la misma etiqueta que usamos antes cuando guardamos la respuesta personalizada escrita por el usuario. Si hay datos guardados en "TinyDB1" bajo la etiqueta "ResponseMessage", éstos se devuelven y se cargan en "ResponseLabel".

Por supuesto, en "TinyDB1" no habrá datos la primera vez que se lanza la app, y este seguirá siendo el caso hasta que el usuario proporcione una respuesta personalizada. Para manejar estas situaciones, la función "TinyDB1.GetValue" tiene un segundo parámetro, llamado "ValueIfTagNotThere", que se usa siempre que no se encuentren datos en la base de datos bajo la etiqueta especificada en el primer parámetro. En esos casos, en "ResponseLabel" se carga la respuesta por defecto, a saber, "I'm driving right now, I'll contact you shortly."

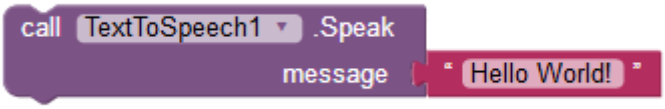
Ahora probamos la app. Para ello, debemos volver a arrancar la app para ver si los datos se guardan de forma persistente y si se recuperan correctamente. Para volver a lanzar la app con el modo de pruebas en vivo activo, podemos cambiar una propiedad poco relevante de algún componente (por ejemplo, el tamaño de fuente de una etiqueta). Esto hará que la app se vuelva a cargar, y que se active el evento "Screen1.Initialize". Por supuesto, también podríamos probar la app empaquetándola e instalando el archivo .apk en nuestro dispositivo. Cuando la app ya esté instalada en nuestro móvil podemos arrancarla, escribir una respuesta personalizada, cerrar la app, y volver a abrirla. Si el mensaje que escribimos sigue estando ahí, es que la aplicación funciona correctamente.

LEER EN VOZ ALTA LOS MENSAJES ENTRANTES.

En esta sección vamos a modificar la app para que, cuando recibamos un mensaje de texto, el móvil lea en voz alta tanto el número de teléfono del emisor como el contenido del mensaje. La idea aquí es que, si vamos conduciendo y recibimos un mensaje, sería muy imprudente descuidar la conducción para leer el mensaje a pesar de que la app envíe una respuesta automática. Para evitar esta peligrosa tentación, la app puede leernos en voz alta el contenido del mensaje para no tener que apartar las manos del volante y la vista de la carretera.

Los dispositivos Android permiten convertir texto a voz, y App Inventor incluye un componente "TextToSpeech" que es capaz de leer en voz alta cualquier texto que le proporcionamos. Sin embargo, es importante tener en cuenta que el texto que le pasemos a T"extToSpeech" debe ser una secuencia de letras, dígitos, y símbolos de puntuación, y no un mensaje SMS.

El componente "TextToSpeech" es muy fácil de usar: Basta con llamar a su función "(TextToSpeech).Speak" y conectar un bloque con el texto a leer a su parámetro "message". Por ejemplo, los bloques mostrados en la figura leerían en voz alta el texto "Hello World!".

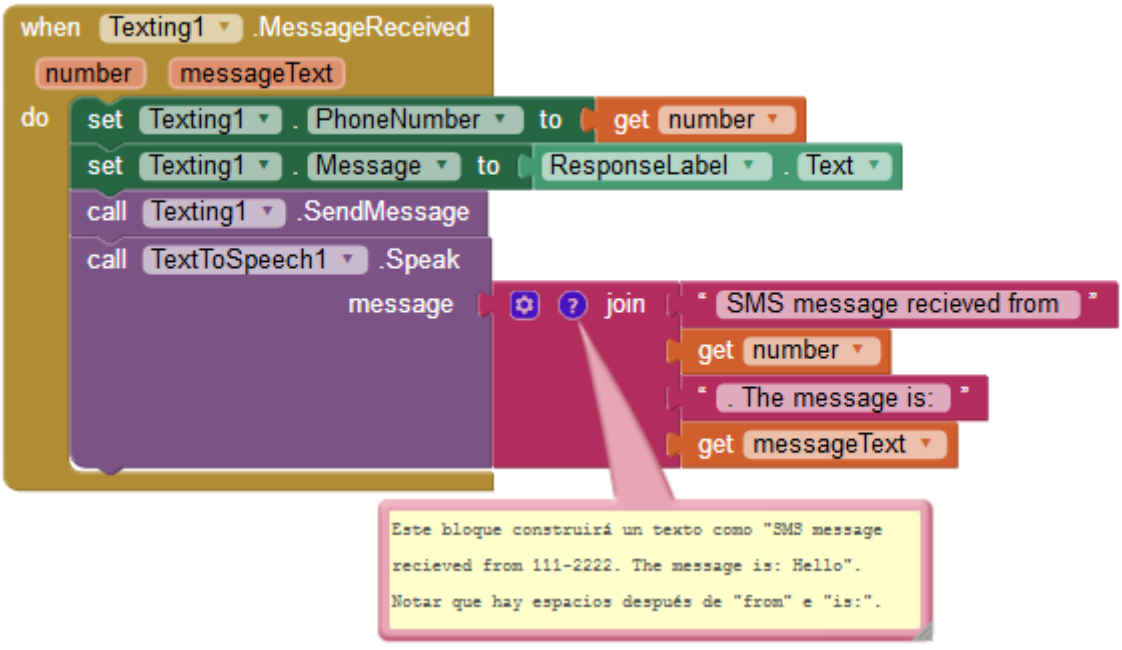


Para nuestra app debemos proporcionarle un texto más complejo, a saber, uno que incluya tanto el número de teléfono de la persona que se lo envió como el contenido del mensaje recibido. En vez de conectarle un texto estático como el bloque "Hello World!" proporcionado antes, le conectaremos un bloque "join" de la bandeja "Text". El bloque "join" permite combinar dos trozos de texto independientes para formar un único objeto de tipo texto.

Deberemos llamar a la función "(TextToSpeech).Speak" dentro del manejador de evento "when (Texting1).MessageReceived" que programamos previamente. El código que escribimos entonces respondía a la ocurrencia del evento "(Texting1).MessageReceived" fijando apropiadamente las propiedades "PhoneNumber" y "Message" de componente "Texting" y enviando un mensaje de respuesta. Aquí ampliaremos la funcionalidad de este manejador de evento añadiendo los bloques de la tabla:

Block type	Drawer	Purpose
`TextToSpeech1.Speak`	TextToSpeech1	Speak the message received aloud.
`join`	Text	Concatenate (join together) the words that will be spoken.
`text` ("SMS text received from")	Text	The first words spoken.
`get number`	Drag in from when block	The number from which the original text was received.
`text` (".The message is")	Text	Put a period in after the phone number and then say, "The message is."
`get messageText`	Drag in from when block	The original message received.

La figura muestra cómo queda el programa:



Veamos cómo funciona este código: Después de enviar la respuesta, llamamos a la función "(TextToSpeech1).Speak". A esta función le podemos conectar cualquier texto en la ranura "message" a su derecha. En este caso usamos el bloque "join" de la bandeja "Text" para construir el texto que la app leerá en voz alta. Esta función concatena el texto "SMS message received from ", el número de teléfono desde el que se envió el mensaje ("get (number)"), el texto ". The message is: ", y el contenido del mensaje recibido ("get (messageText)"). Así, si el abonado con número de teléfono $111-2222$ envió el mensaje "Hello", el teléfono leería en voz alta el texto "SMS message received from 111-2222. The message is: Hello".

Probamos el funcionamiento de la app. Para ello, necesitaremos un segundo teléfono desde el que enviar un mensaje SMS al teléfono que está ejecutando la app. Si todo está correcto, el teléfono leerá en voz alta el texto del mensaje recibido, y además, enviará automáticamente una respuesta.

7.6. MODIFICACIONES A "NOTEXTINGWHILEDRIVING1".

Cuando hayamos conseguido que la app funcione correctamente podemos explorar algunas modificaciones, como por ejemplo, las que se proponen a continuación:

- Podemos escribir una versión que le permita al usuario definir respuestas personalizadas para números de teléfono particulares. Para ello será necesario añadir bloques condicionales "if" que comprueben esos números.
- Escribe una versión que haga sonar una alarma cuando el mensaje recibido se haya enviado desde un número de teléfono perteneciente a una "lista de notificaciones".

7.7. COMENZAR CON LA APP "INTERACTIVESCRAPBOOK".

En esta sección volvemos a usar el componente "TextToSpeech" para construir una app que implemente un álbum de recortes interactivo (interactive scrapbook) con hasta tres fotografías de familiares y amigos. Cuando el usuario pulse en una de las fotografías, la voz de Android leerá una de las frases típicas que más caracterizan a esa persona. Además, la app permitirá reemplazar una de las tres fotografías del álbum por otra fotografía tomada con la cámara del dispositivo, y escribir una frase típica para esa nueva persona.

Para esta app necesitaremos algunos archivos: La app comienza con dos fotografías de amigos o familiares ya insertadas en el álbum. Asegúrate de traer estas dos fotografías en una memoria USB para descargarlas en la carpeta de archivos de los alumnos. Además, la fotografía que nosotros podemos tomar comienza mostrando la imagen *Yourpicturehere.png*, disponible en la carpeta de archivos de los alumnos.

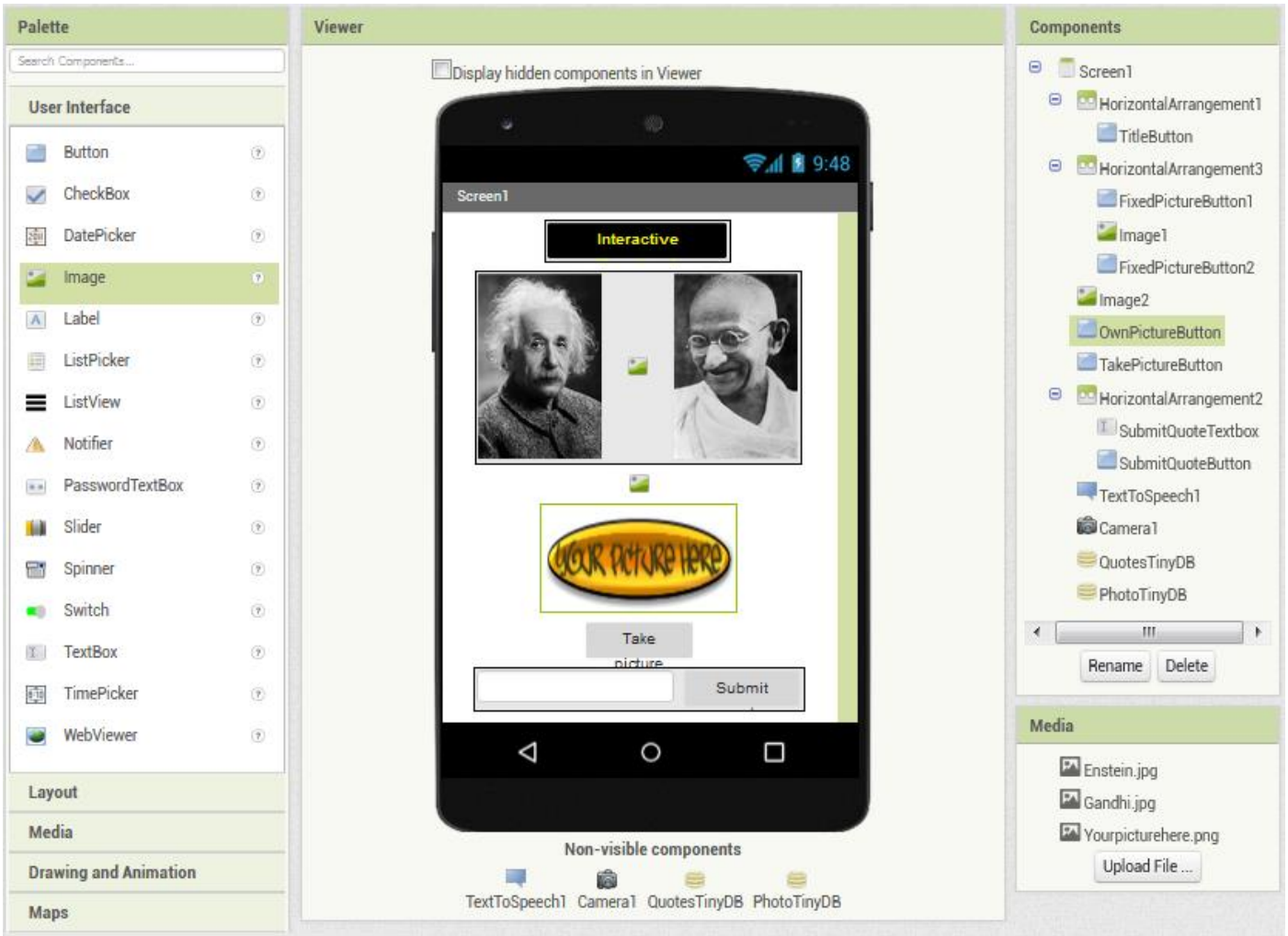
7.8. DISEÑAR LOS COMPONENTES DE "INTERACTIVESCRAPBOOK".

La figura a continuación muestra la vista de Diseñador para esta app, con todos los componentes necesarios:

- Un organizador "HorizontalArrangement" para alojar el botón de título, de forma que podamos posicionarlo en el centro.

- Un botón al que llamaremos "TitleButton" que muestra el título de la app y que al pulsarlo lee en voz alta las instrucciones para usar esta app.
- Otro organizador "HorizontalArragement" para alojar los dos botones que mostrarán las fotos fijas.
- Dos botones llamados "FixedPictureButton1" y "FixedPictureButton2" que muestran las fotografías fijas de dos de nuestro familiares y amigos, y que, al ser pulsados, acceden a una lista de frases típicas de la persona en cuestión. En nuestro caso, los botones muestran dos imágenes de Albert Einstein y Mahatma Gandhi, respectivamente. Al pulsar en uno de ellos, la app accede a la lista de citas célebres del personaje seleccionado. Aquí hemos usado las siguientes citas:

Einstein
"El verdadero signo de la inteligencia no es el conocimiento, sino la imaginación".
"El aprendizaje es experiencia. Todo lo demás es información".
"Hay una fuerza motriz más poderosa que el vapor, la electricidad, y la energía atómica: La voluntad".
Ghandi
"La fuerza no viene de la capacidad física. Viene de una voluntad indomable".
"Vive de forma más sencilla para que otros puedan sencillamente vivir".
"Vive como su fueras a morir mañana, aprende como si fueras a vivir para siempre".



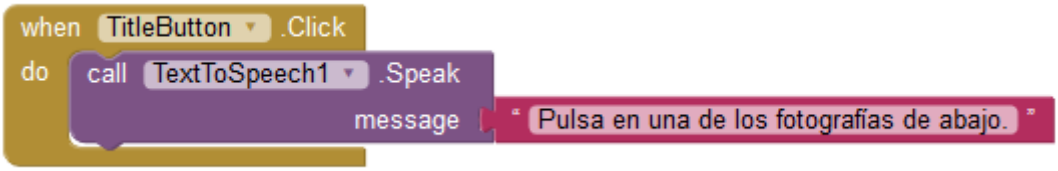
- Un componente "Image" vacío para separar las dos fotografías fijas.
- Un componente "TextToSpeech" que lea en voz alta el texto de las frases típicas.
- Un botón llamado "OwnPictureButton" que mostrará la fotografía que tomemos con la cámara, y que al pulsarlo, leerá la frase típica que hayamos escrito en la caja de texto "SubmitQuoteTextbox". La primera vez que usamos la app, este botón muestra la imagen *Yourpicturehere.png*.
- Un componente "Image" vacío para separar el botón "OwnPictureButton" del organizador de las fotos fijas.

- Un botón llamado "TakePictureButton" que abre la cámara del teléfono para permitirle al usuario tomar una fotografía.
- Un tercer organizador "HorizontalArragement" para contener la caja de texto "SubmitQuoteTextbox" y el botón "SubmitQuoteButton".
- Una caja de texto llamada "SubmitQuoteTextbox" que le permite al usuario escribir una frase típica de la persona que ha fotografiado.
- Un botón llamado "SubmitQuoteButton" que envía el texto desde la caja de texto a una base de datos "TinyDB" para almacenarla de forma persistente.
- Un componente "Camera" que le permita al usuario tomar una fotografía.
- Dos componentes "TinyDB", llamados "QuotesTinyDB" y "PhotoTinyDB", que almacenan de forma persistente la frase escrita por el usuario y la foto tomada por el usuario, respectivamente.

7.9. AÑADIR COMPORTAMIENTOS A LOS COMPONENTES DE "INTERACTIVESCRAPBOOK".

PROGRAMAR EL BOTÓN DE TÍTULO.

Cuando el usuario abre la app por primera vez, puede que no tenga claro para qué sirve. Hemos creado un título en forma de botón para que, cuando el usuario lo pulse, la app lea un mensaje que le indique qué debe hacer a continuación. El programa que implementa esta funcionalidad es el siguiente:



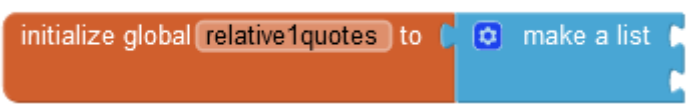
¿Cómo funciona este código? Queremos que la app lea en voz alta un texto informativo al pulsar el botón "TitleButton". Para ello, dentro del manejador "when (TitleButton).Click" añadimos el bloque de función "call (TextToSpeech1).Speak", que sirve para que el componente "TextToSpeech" lea en voz alta el mensaje que le conectemos a su ranura de parámetros "message".

DEFINIR UNA VARIABLE DE TIPO LISTA.

Las **listas** son un tipo especial de variable en el que podemos contener múltiples datos relacionados. Algunos ejemplos son una lista de alumnos de clase (donde cada elemento de la lista es un texto con el nombre y apellidos del alumno), las listas telefónicas (en las que cada elemento es un número), etc. Hablaremos mucho más de las listas en capítulos posteriores, pero en esta app solo necesitamos saber que, para almacenar en una sola variable todas las frases típicas de una cierta persona, necesitamos que esa variable sea de tipo lista (una lista de textos, en este caso).

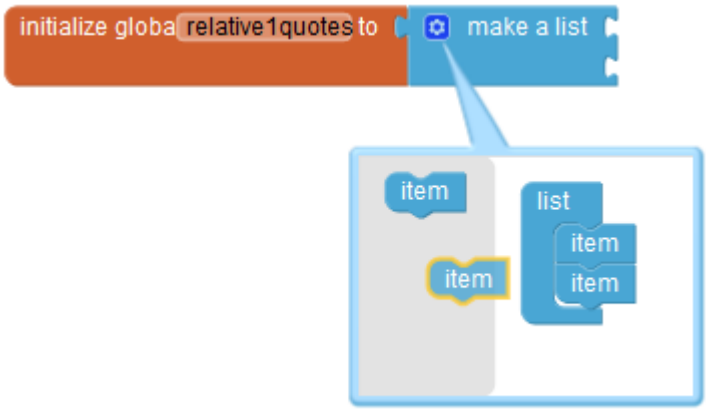
Para crear una lista, primero debemos definir la variable que la alberga. Para ello, vamos a la bandeja "Variables", y sacamos un bloque "initialize global (name) to". Cambiamos el nombre de la variable de "name" a "relative1quotes" (frases del pariente 1).

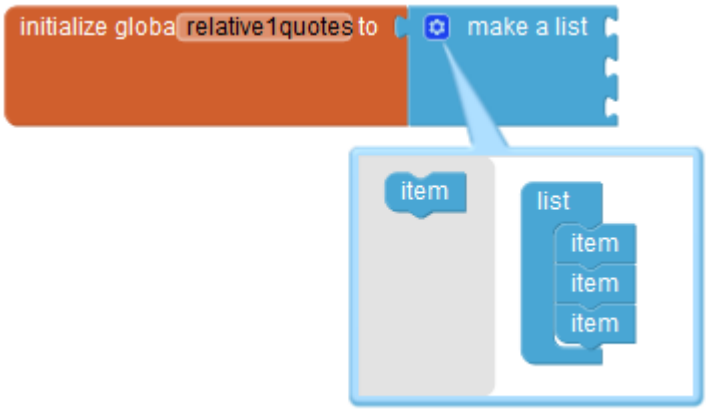
A continuación vamos a crear la lista que almacenaremos en esa variable. Acudimos a la bandeja "List", y seleccionamos el bloque "make a list", que conectamos a la ranura abierta a la derecha del bloque "initialize global (relative1quotes) to".



Con esto ya hemos creado una variable de tipo lista, pero de momento está vacía. Para asociar datos a esa lista debemos conectarlos a las ranuras abiertas a la derecha del bloque "make a list". En este caso, queremos rellenar la lista con los textos de las frases típicas del primer pariente, así que acudimos a la bandeja "Text", y sacamos tantos bloques de texto vacíos como frases queramos añadir. Para esta app, vamos a añadir al menos tres frases.

(Notar que, para añadir tres textos a nuestra lista de frases, hemos tenido que crear una ranura abierta extra a la derecha del bloque "make a list", que por defecto solo venía con dos. Esto podemos hacerlo pulsando en el icono azul del engranaje, y en la ventana emergente, arrastrando tantos bloques "item" como necesitemos a la derecha del bloque "list", ver figura).

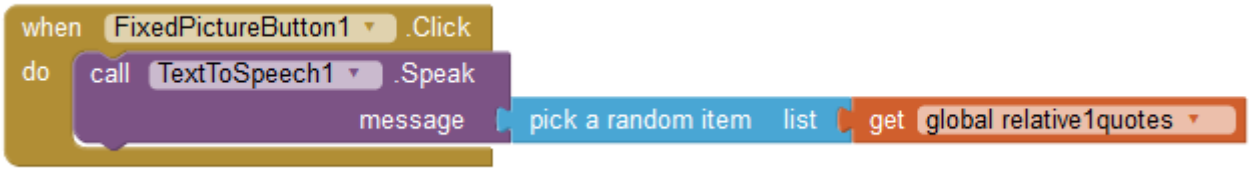




Ahora repetimos este proceso para definir la variable tipo lista que albergue las frases típicas del segundo pariente cuya foto fija tendremos en pantalla:

PROGRAMAR LOS BOTONES DE LAS FOTOS FIJAS.

Queremos que, al pulsar en cualquiera de los botones que muestran las fotos fijas de nuestros familiares, la app lea en voz alta una de sus frases típicas elegida al azar. Los dos programas son idénticos, así que aquí solo mostramos uno de ellos:

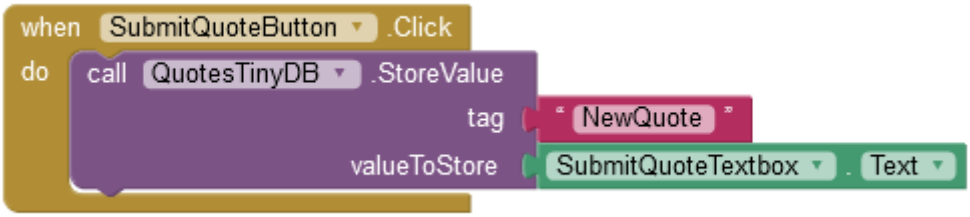


¿Cómo funciona este código? Cuando el usuario pulsa en el botón con la fotografía fija del primer familiar, el manejador de eventos "when (FixedPictureButton).Click" detecta esta acción, y en respuesta, llama a la función "(TextToSpeech1).Speak" para leer en voz alta una de las citas de este familar. Al conector abierto a la derecha de "call (TextToSpeech1).Speak" conectamos un bloque "pick a random item" (bandeja "List"), el cual elegirá un elemento al azar de la lista que le indiquemos en su ranura de parámetros "list". La lista de la que queremos obtener un elemento al azar es "relative1quotes".

ALMACENAR EL TEXTO ESCRITO EN LA CAJA DE TEXTOS EN UNA BASE DE DATOS "TINYDB".

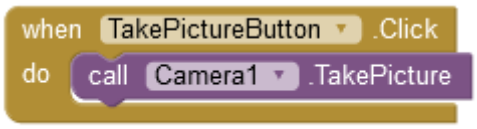
Ahora debemos programar el botón "SubmitQuoteButton" para que el usuario pueda guardar en la base de datos "QuotesTinyDB" la frase típica del familar cuya fotografía tomará justo después mediante el componente "Camera". A continuación, queremos enviar esa fotografía a una base de datos llamada "PhotoTinyDB". Por último, hemos de comprobar si el usuario ha añadido una cita a la foto recién tomada antes de leer nada o de mostrar el resultado.

La figura muestra el programa para guardar de forma persistente la cita escrita por el usuario: Como vemos, la acción de guardar una cita en una base de datos se limita a tomar un bloque "StoreValue", darle a la etiqueta un nombre, e indicar qué valor se guardará en la base de datos bajo esa etiqueta (en este caso, el texto escrito por el usuario en la caja de texto):



TOMAR UNA FOTOGRAFÍA.

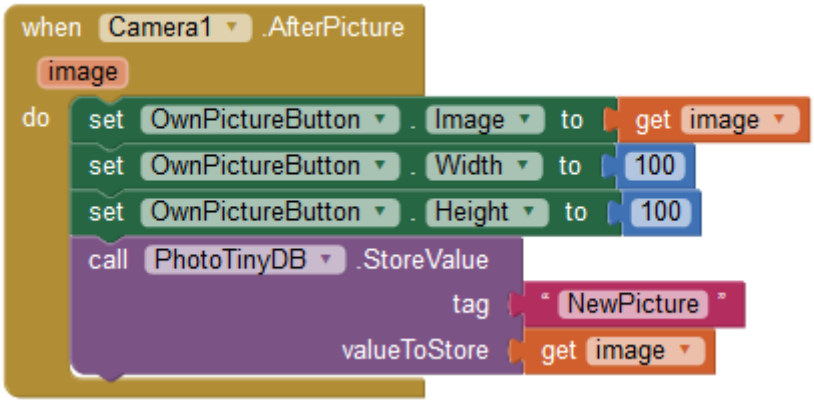
En el capítulo 3 ya vimos cómo funciona el componente "Camera". Para que la app active la cámara al pulsar el botón "TakePictureButton" usamos el manejador "when (TakePictureButton).Click", e insertando dentro de su sección "do" la función "(Camera1).TakePicture". La figura muestra el programa que implementa este comportamiento:



Al probar este programa veremos que, cuando el usuario pulsa el botón, la aplicación abre la cámara de fotos del móvil para tomar la fotografía del familiar.

ENVIAR LA FOTOGRAFÍA TOMADA CON LA CÁMARA A UNA BASE DE DATOS.

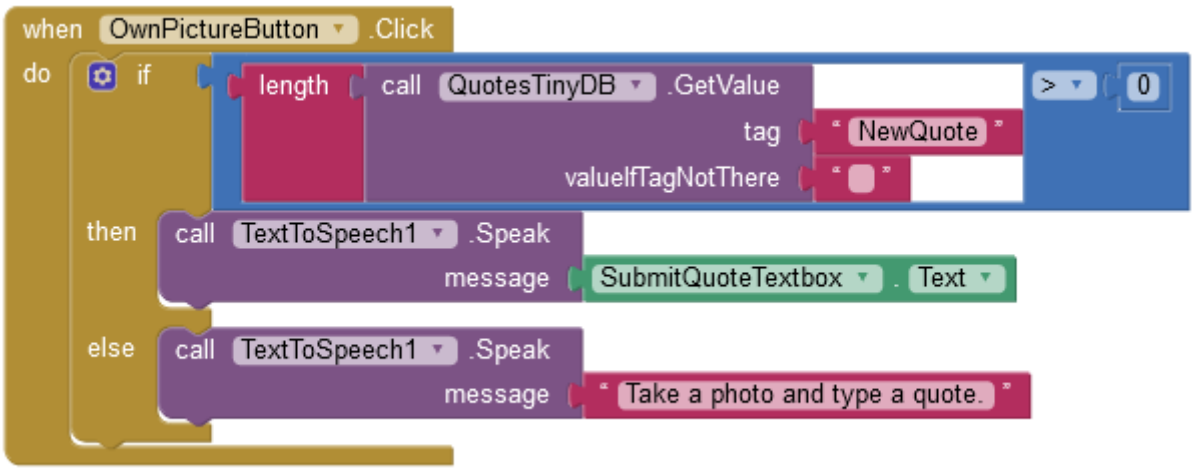
El manejador "when (Camera1).AfterPicture" sirve para gestionar qué ocurre justo después de que el usuario haya tomado una fotografía con la cámara del teléfono. Este manejador incluye un argumento "image" que nos permite acceder a la fotografía recién tomada. Podemos hacer una referencia a esa imagen pasando el ratón sobre el argumento "image" y obteniendo un bloque "get (image)".



Lo que queremos hacer aquí es fijar la propiedad "Image" del botón "OwnPictureButton" a la fotografía recién tomada, y guardar esa foto en la base de datos "PhotoTinyDB". También queremos redimensionarla, porque la cámara del dispositivo podría tomar una fotografía con un tamaño de 2048×1536 píxeles, demasiado grande como para poder mostrarla por pantalla. En este caso, vamos a cambiar el tamaño de la foto a 100×100 píxeles. La figura muestra el programa que implementa esta funcionalidad.

ACTIVAR EL BOTÓN "OWNPICTUREBUTTON".

Ahora vamos a hacer que, cuando el usuario pulse en el botón con la fotografía del familiar que acaba de tomar, la app lea en voz alta la cita que el usuario ha escrito en la caja de texto. La figura muestra el programa que se encarga de ello. Como vemos, este programa hace una comprobación previa, simplemente para ver si hay una cita almacenada en la base de datos "QuotesTinyDB", y si no la hay, para leer un texto estándar que anima al usuario a tomar una foto y a añadir una cita.



¿Cómo funciona este programa? Al pulsar el botón "OwnPictureButton", la app coprueba si la longitud del texto almacenado en la base de datos "QuotesTinyDB" bajo la etiqueta "NewQuote" es mayor que cero. Si esta condición es cierta, es porque el usuario ha escrito una cita y la ha guardado en la base de datos, y entonces la app puede leer en voz alta la cita escrita en la caja de texto "SubmitQuoteTextbox". Si la condición es falsa es porque el usuario no ha escrito o no ha enviado la cita a la base de datos, y en ese caso, la app le informa cómo proceder (a saber, tomar una foto de un familiar, y añadir una frase típica de ese familiar).

COMPROBAR UNA BASE DE DATOS "TINYDB".

En nuestros programas es muy común trabajar con variables y bases de datos, y a menudo tendremos que comprobar si hay algún dato guardado en esa variable o en esa base de datos.

En el caso de la app "InteractiveScrapbook", cuando el usuario arranca la app la primera vez, el botón que más tarde se convierte en una fotografía necesita hacer algo para indicarle al usuario cómo usar la app, y la caja de texto debe contener una pista que informe al usuario de que debe enviar la cita que escriba.

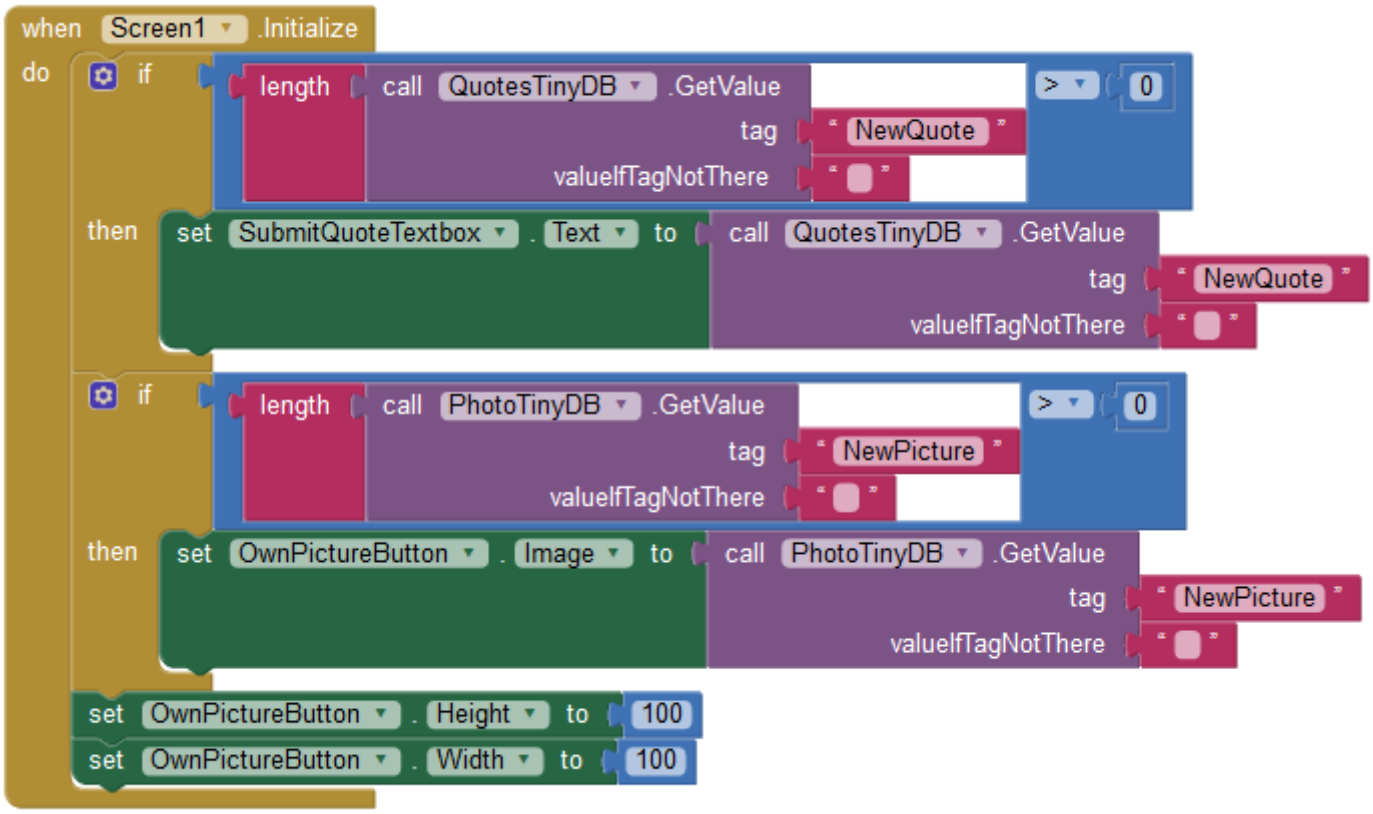
A continuación añadimos una tabla que describe todas estas funcionalidades, dependiendo de si el usuario acaba de arrancar la app por primera vez, o de si ya lo había hecho antes:

Elemento	¿Qué hace el usuario?	¿Qué aspecto debe tener?	¿Qué debe hacer?
Caja de texto "SubmitQuoteTextbox"	Acaba de abrir la app por primera vez.	Una caja de texto con la indicación "Insert quote text here".	Contener la indicación "Insert quote text here".

Caja de texto "SubmitQuoteTextbox"	Ha escrito una cita.	El texto de la cita que el usuario ha escrito.	Mantener el texto hasta que se haya enviado a una base de datos, y mostrarlo.
Botón "OwnPictureButton"	Acaba de abrir la app por primera vez.	Un botón con la imagen *Yourpicturehere.png*.	Leer en voz alta el mensaje "Take a photo and type a quote."
Botón "OwnPictureButton"	Ha tomado una fotografía (y la ha enviado a una base de datos).	La imagen 100×100 que acaba de tomar el usuario.	Leer en voz alta la cita que ha escrito el usuario en la caja de texto.

Como vemos, la apariencia y la funcionalidad de la caja de texto "SubmitQuoteTextbox" y del botón "OwnPictureButton" cambia después de que el usuario haya enviado la cita que ha escrito y la foto que ha tomado a sus respectivas bases de datos. Esto es importante, porque necesitamos que la app compruebe si hay algo en las bases de datos antes de decidir qué mostrar y qué hacer.

Así pues, lo primero que debemos hacer nada más arrancar la app es configurar las apariencias y comportamientos iniciales de los componentes "SubmitQuoteTextbox" y "OwnPictureButton" para que la app pueda funcionar correctamente. Y esto lo hacemos comprobando si hay valores asociados a las etiquetas correspondientes en las bases de datos "QuotesTinyDB" y "PhotoTinyDB". La figura muestra el programa que nos permite inicializar la app:



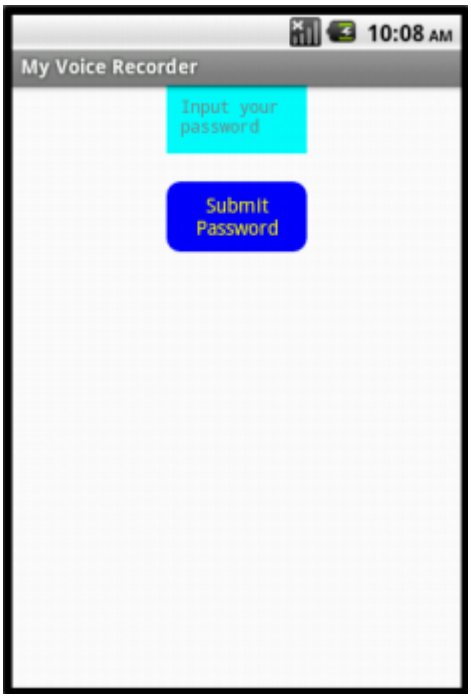
¿Cómo funciona este programa? Nada más arrancar la app se activa el manejador "when (Screen1).Initialize", que ejecutará loa bloques contenidos dentro de su sección "do". En primer lugar, la app comprueba si la longitud del valor contenido en la base de datos "QuotesTinyDB" bajo la etiqueta "NewQuote" es mayor que cero, esto es, si la base de datos contiene el texto de una frase típica. De ser así, obtiene ese valor de la base de datos y lo carga en la propiedad "Text" de la caja de texto "SubmitQuoteTextbox" para mostarlo por pantalla. A continuación, comprueba si la longitud del valor contenido en la base de datos "PhotoTinyDB" bajo la etiqueta "NewPicture" es mayor que cero, esto es, si la base de datos contiene una fotografía. De ser así, obtiene ese valor de la base de datos y lo carga en la

propiedad "Image" del botón "OwnPictureButton" para mostrar la foto por pantalla. Además, y en todo caso, reconfigura el tamaño de la imagen que mostrará el botón personalizable "OwnPictureButton".

7.10. MODIFICACIONES A "INTERACTIVESCROPBOOK".

- Cambia por completo el diseño de la app "InteractiveScropbook" para que añada dos botones de fotos personalizables, más botones con fotos fijas, etc.
- Intenta combinar lo que hemos aprendido en las apps "MyVoiceRecorder" e "InteractiveScropbook" para desarrollar una app donde el usuario pueda insertar las frases típocas oralmente, en lugar de tener que escribirlas.
- Piensa en cómo podrías ampliar la app "InteractiveScropbook" para que el usuario pueda crear una lista de frases típicas, de las cuales la app elegirá una aleatoriamente para leerla en voz alta.

7.11. COMENZAR CON LA APP "MYVOICERECORDER".



A continuación vamos a desarrollar una app que nos permita convetir a texto nuestra voz usando el reconocimiento de voz de Google Voice. Además, para asegurarnos de que las notas de voz que grabemos quedan protegidas, añadiremos una contraseña de forma que solo nosotros podamos acceder a ellas. Como hemos comentado antes, el reconocimiento de voz todavía no es perfecto, por lo que probablemente no conseguiremos un texto que refleje exactamente todo lo que hemos dicho.

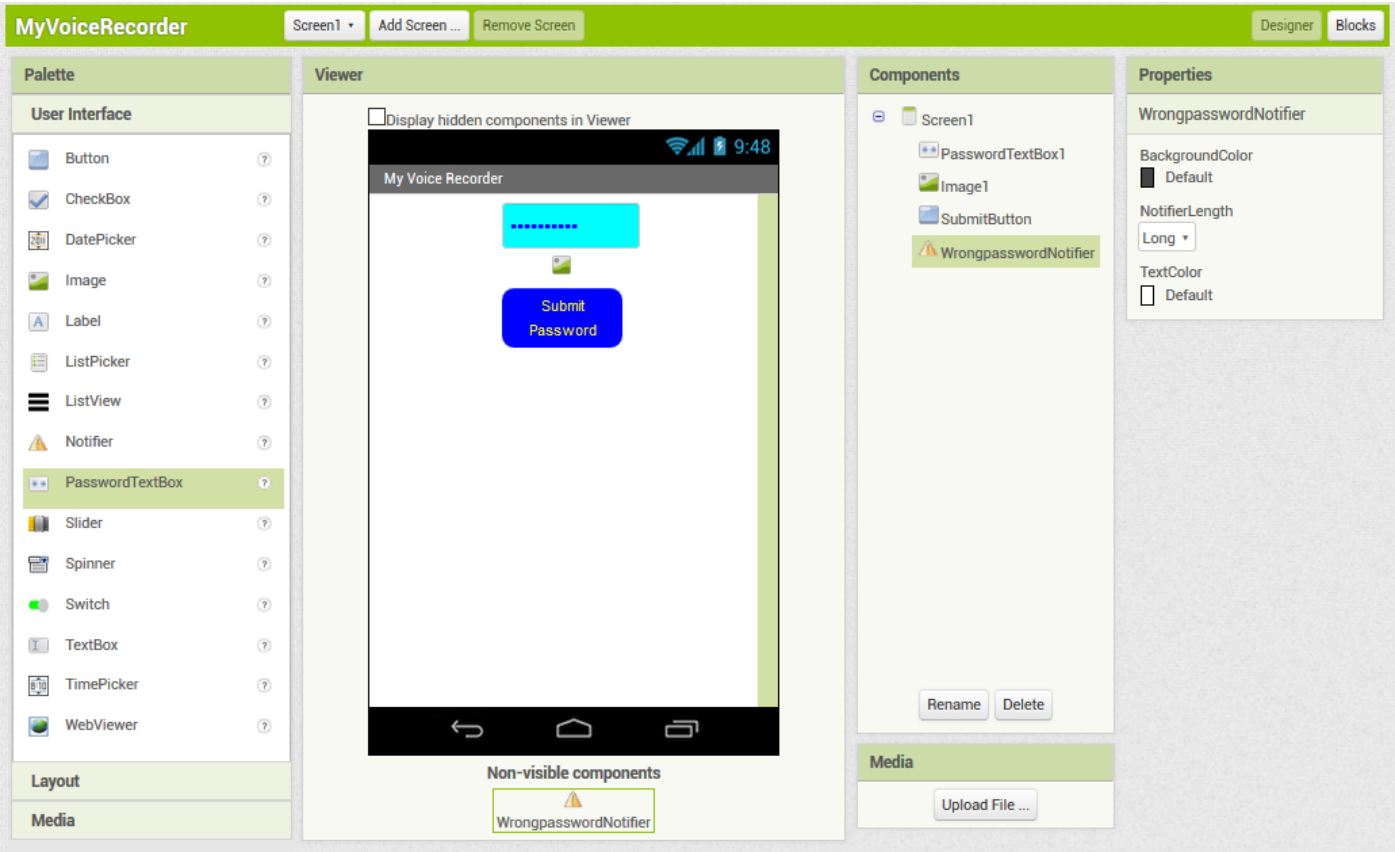
Esta app usa el servicio de reconocimiento de voz para grabar nuestras notas de voz. Solo se puede acceder a través de una pantalla protegida con contraseña, que nos conducirá a una segunda pantalla donde podremos grabar nuestra voz. Para probar esta app necesitamos disponer de un dispositivo, ya que el emulador no soporta la existencia de dos pantallas. Y además, el dispositivo necesita estar conectado a internet para poder usar el servicio de reconomiento de voz de Google Voice.

Esta es la primera vez en la que usaremos dos pantallas en una app, y la forma en la que construimos la app en estos casos es importante. En primer lugar, crearemos la pantalla protegida por contraseña (pantalla "Screen1"), y escribiremos los bloques de código para esa pantalla. Después añadiremos la segunda pantalla ("Voicescreen") donde podremos tomar las notas de voz, y escribiremos sus bloques de código. Finalmente probaremos la app ya terminada en el teléfono. Notar que en el Diseñador hay botones disponibles para crear una nueva pantalla, para eliminar una pantalla, y para cambiar de una pantalla a otra. Al cambiar a otra pantalla, también cambiarán los bloques para esa pantalla en el Editor.

7.12. DISEÑAR LOS COMPONENTES DE "SCREEN1".

La figura muestra la vista de Diseñador para la pantalla "Screen1", donde el usuario insertará la contraseña para acceder a la segunda pantalla de la app ("Voicescreen"). Añade los componentes necesarios y fija los valores de sus propiedades para que se parezca a la de la figura.

Notar que el componente donde el usuario ingresará la contreseña es un "PasswordTextBox", de la paleta "User Interface". Modifica su propiedad "Hint" para que muestre el texto "Input your password".

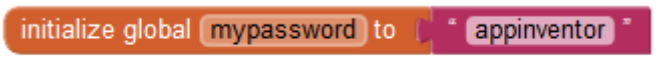


7.13. AÑADIR COMPORTAMENTOS A LOS COMPONENTES DE "SCREEN1".

Una vez añadidos los componentes a la pantalla "Screen1", vamos a programar su funcionamiento. Primero necesitamos definir la contraseña. A continuación abriremos la pantalla "Voicescreen" si la contraseña proporcionada es correcta, o lanzaremos un mensaje de error si es errónea. Después de haber programado estos comportamientos, crearemos la pantalla "Voicescreen".

DEFINIR LA CONTRASEÑA.

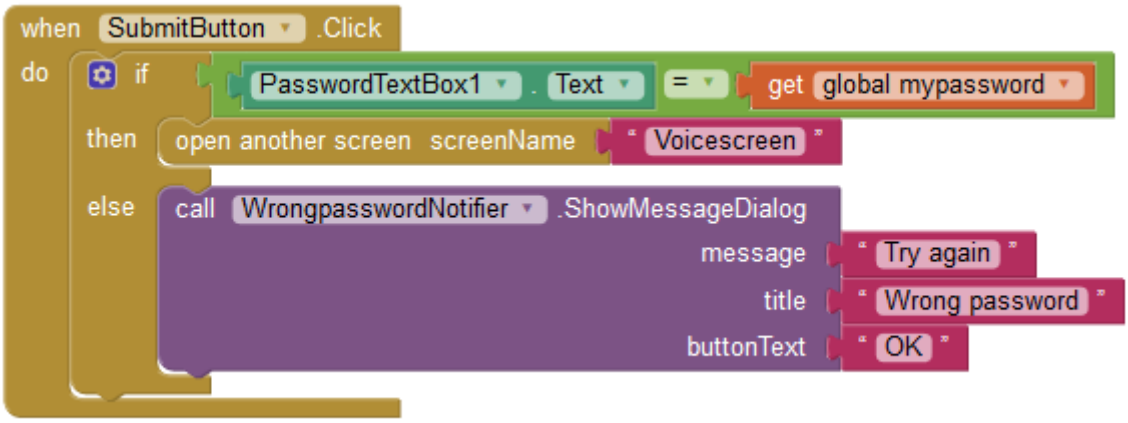
Antes de usar la contraseña en el código de esta app, debemos definir una variable "mypassword" que contenga la contraseña correcta (que es nuestro caso será "appinventor"). El código necesario para ello es muy sencillo:



COMPROBAR LA CONTRASEÑA Y ABRIR LA SEGUNDA VENTANA.

Cuando el usuario escriba su contraseña en la caja de texto "PasswordTextBox1" y pulse el botón "SubmitButton" queremos que ocurra algo que, en palabras, podríamos expresar así: "Si la contraseña escrita por el usuario coincide con la contraseña correcta, abrimos la ventana "Voicescreen". Si la contraseña no coincide, mostramos en un notificador un mensaje de error que indique que la contraseña es incorrecta, y que hay que probar de nuevo".

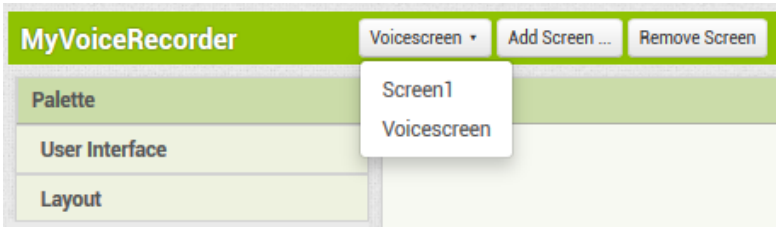
El programa que implementa esta funcinalidad es el siguiente:



En el bloque "open another screen", en el parámetro "screenName" debemos insertar un bloque de texto que especifique el nombre de la ventana que deseamos abrir ("Voicescreen"), antes incluso de que la propia ventana exista. Esto puede hacerse porque, cuando la segunda ventana esté creada y el usuario pulse el botón "SubmitButton", la app abrirá dicha ventana. Por supuesto, si probamos la app tal y como está ahora mismo fallará, puesto que todavía no hemos creado la segunda ventana.

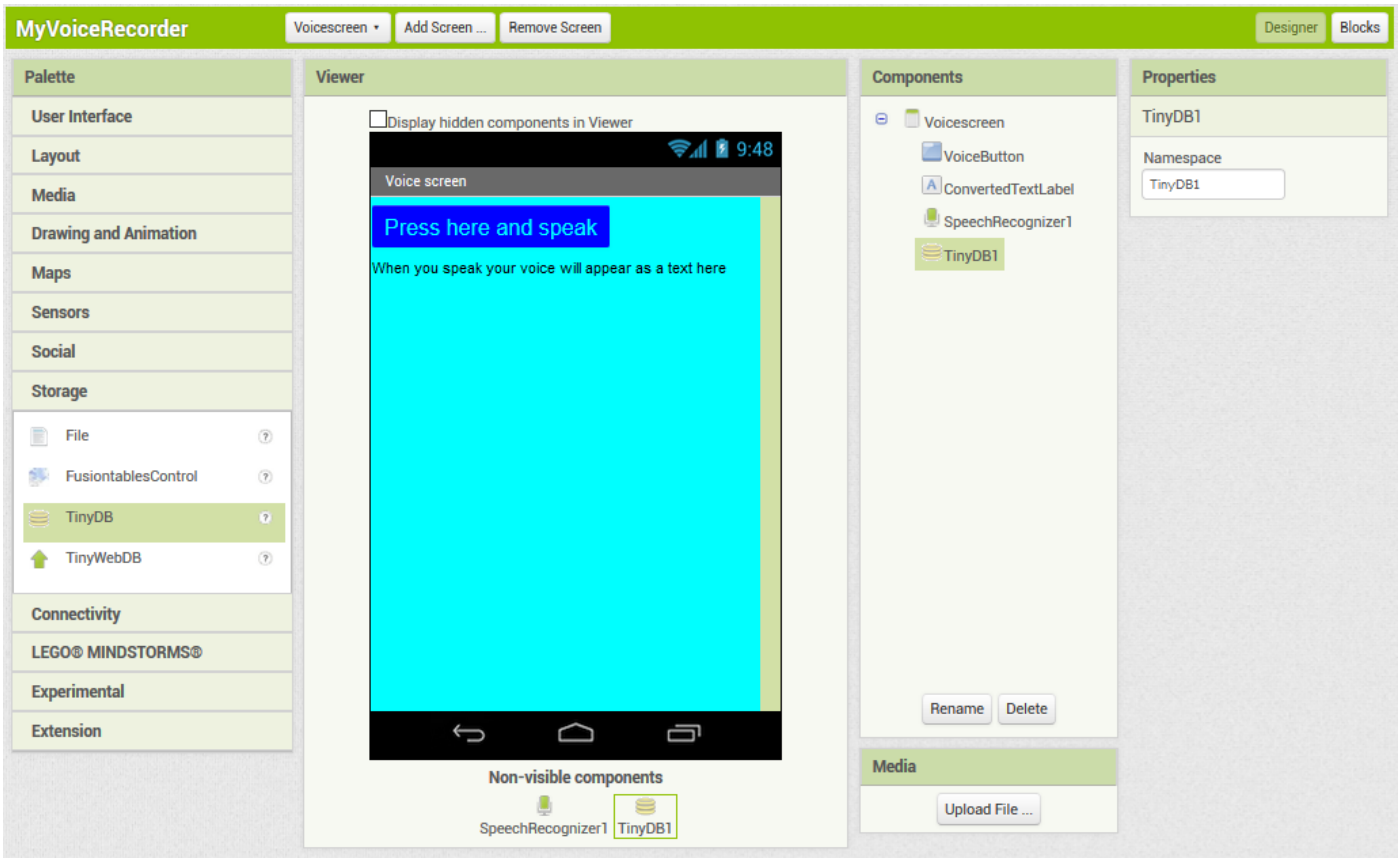
7.14. DISEÑAR LOS COMPONENTES DE "VOICESCREEN".

Cuando en el Diseñador de App Inventor pulsamos el botón "Add Screen …", aparece una ventana emergente que nos pide el nombre de la nueva ventana. Una vez proporcionado el nombre no se puede cambiar, así que debemos tener cuidado y no equivocarnos. Vamos a llamar a esta segunda ventana "Voicescreen". Cuando la hayamos añadido al proyecto, en la zona superior izquierda del Diseñador habrá un botón que nos permitirá cambiar entre la ventana "Screen1" y la ventana "Voicescreen".



Esta segunda pantalla es sencilla: Dispone de un botón llamado "VoiceButton" con el texto "Press here and speak". El texto producido por el reconocimiento de voz aparecerá en una etiqueta bajo el botón, que al empezar sólo muestra el mensaje "When you speak your voice will appear as a text here" (aunque también podemos inicializarla sin texto alguno). El texto convertido lo almacenaremos de forma persistente en el teléfono usando una base de datos "TinyDB".

La figura muestra la vista de Diseñador para la pantalla "Voicescreen". Añade los componentes necesarios y fija los valores de sus propiedades para que se parezca a la de la figura.

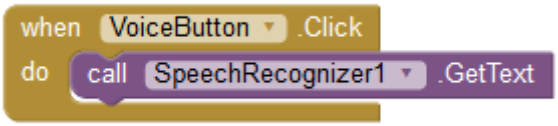


7.15. AÑADIR COMPORTAMENTOS A LOS COMPONENTES DE "VOICESCREEN".

Ahora que ya hemos creado y programado la pantalla de la contraseña, y tras haber creado la segunda pantalla, podemos empezar con el reconocimiento de voz. En primer lugar seleccionamos el manejador de evento "when (VoiceButton).Click" para llamar al componente "SpeechRecognizer". A continuación configuraremos una etiqueta para mostrar el resultado del reconocimiento de voz, esto es, la nota de voz del usuario que la app ha convertido a texto. Al mismo tiempo, guardaremos ese texto en una base de datos "TinyDB" para asegurarnos que la próxima vez que abramos la app, el texto en el que hemos convertido la nota de voz sigue apareciendo en la etiqueta.

ABRIR EL COMPONENTE "SPEECHRECOGNIZER".

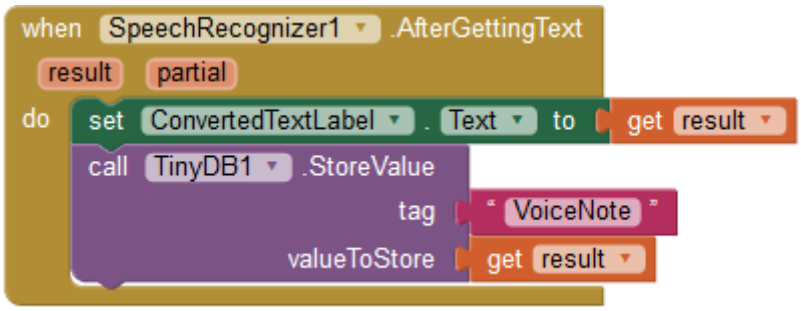
Para abrir el componente "SpeechRecongnizer" basta con activar el manejador de evento "when (VoiceButton).Click" y conectar dentro de su sección "do" el bloque de función "call (SpeechRecognizer1).GetText":



USAR EL TEXTO RECONOCIDO POR "SPEECHRECOGNIZER".

Después de que Google Voice haya grabado nuestra voz, queremos que el resultado del reconocimiento de voz aparezca en pantalla. El manejador de evento "when (SpeechRecognizer1).AfterGettingText" nos permite obtener texto resultado del reconocimiento de voz (el cual está almacenado en el argumento

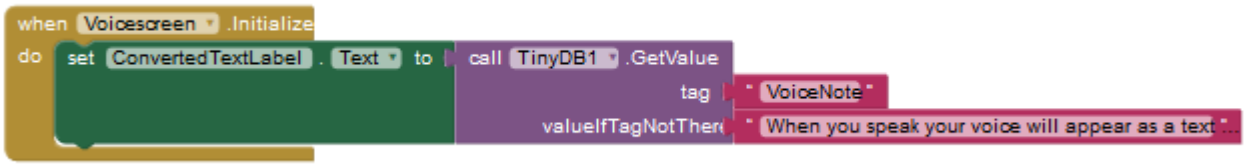
"result"), y hacer todas aquellas tareas que necesitemos hacer después de haber obtenido ese texto. En nuestro caso queremos mostrar el texto obtenido en la etiqueta "ConvertedTextLabel" y guardarlo de forma persistente en una base de datos "TinyDB" para que esté disponible la próxima vez que abramos la app:



¿Cómo funciona este programa? Cuando el componente "SpeechRecognizer1" obtiene el texto resultado del reconocimiento de voz, lanza el evento "AfterGettingText". El manejador de eventos "when (SpeechRecognizer).AfterGettingText" captura este evento, y en su argumento "result" almacena el texto arrojado por el reconocimiento de la voz. Cuando esto ocurre, el manejador fija la propiedad "Text" de la etiqueta "ConvertedTextLabel" al texto obtenido (el cual reside en el argumento "result"), y a continuación, llama a la función "(TinyDB1).StoreValue" para almacenar ese valor de tipo texto en la base de datos "TinyDB1" con la etiqueta identificativa "VoiceNote".

INICIALIZAR LA PANTALLA "VOICESCREEN".

La primera vez que abrimos la app, la etiqueta "ConvertedTextLabel" que alojará el texto resultado de la conversión aparece vacía. Tras hacer la conversión de voz a texto, la etiqueta muestra el texto obtenido. Ahora, para hacer que ese texto aparezca en la etiqueta la siguiente vez que abramos la app, debemos fijar la propiedad "(ConvertedTextLabel).Text" al contenido almacenado en la base de datos "TinyDB1":



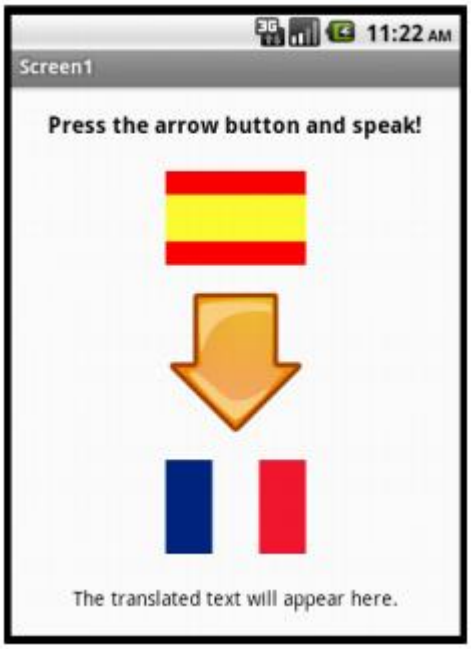
¿Cómo funciona el programa? Nada más arrancar la pantalla "Voicescreen", el manejador de eventos "when (Voicescreen).Initialize" fija la propiedad "Text" de la etiqueta "ConvertedTextLabel" al valor almacenado en la base de datos "TinyDB1" bajo la etiqueta "VoiceNote", usando la función "call (TinyDB1).GetValue". El valor que devuelve esta función es el texto reconocido la última vez que ejecutamos la app. Si la función no encuentra en la base de datos una etiqueta llamada "VoiceNote", el valor que devuelve por defecto es el texto "When you speak your voice will appear as a text here".

Con esto, hemos terminado la app. Lanzamos las pruebas en vivo para segurarnos de que funciona correctamente. (Recuerda: El teléfono móvil debe disponer de conexión a internet para poder conectarse al servicio de reconocimiento de voz de Google Voice).

7.16. MODIFICACIONES A "MYVOICERECORDER".

- Podemos ampliar la app "MyVoiceRecorder" para que la contraseña se guarde también en una base de datos "TinyDB", y para que pueda cambiarla el usuario.
- También podemos modificar la app para que podamos guardar más de una nota de voz, y para poder navegar a través de todas ellas.

7.17. COMENZAR CON LA APP "MYTRANSLATOR".

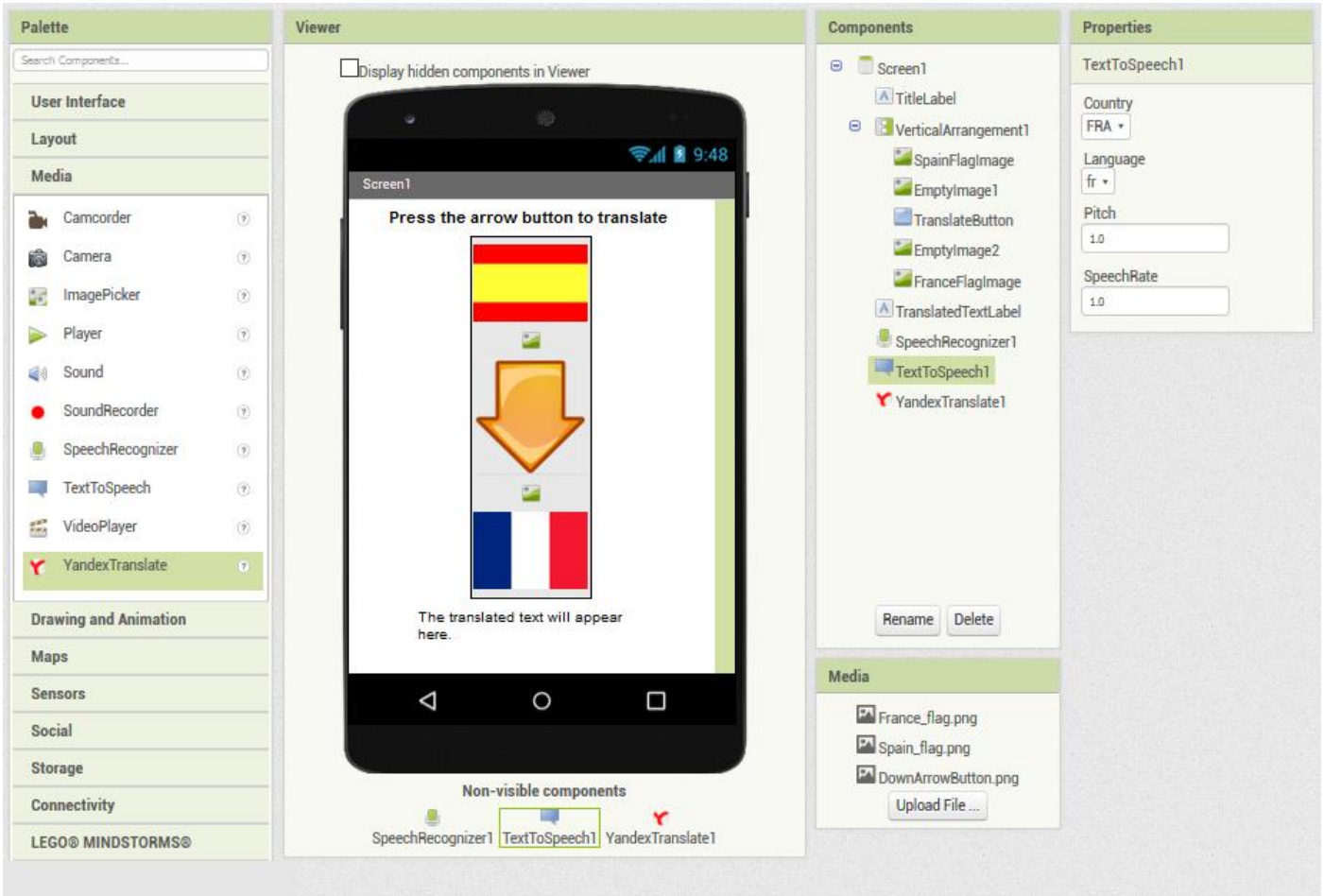


Para terminar el capítulo, vamos a construir una app que combine el uso de los componentes "SpeechRecognizer" y "TextToSpeech", junto con un nuevo componente "YandexTranslate" que permite traducir *textos* de un idioma a otro.

La app "MyTranslator" nos permitirá traducir del castellano al fracés empleando la voz. Esta aplicación contiene un botón que pulsa el usuario para activar el reconocimiento de voz. En ese momento, el usuario indica verbalmente la frase en castellano que desea traducir al francés, para que la app la escriba y la lea en voz alta en ese idioma. Necesitaremos un par de archivos para diseñar la interfaz de usuario de la app: Las imágenes de las banderas española y francesa, y la imagen de una flecha para el botón de traducción. Todas estas imágenes están disponibles la carpeta de archivos para los alumnos, con los nombres *Spain_flag.png*, *France_flag.png*, y *DownArrowButton.png*.

7.18. DISEÑAR LOS COMPONENTES DE "MYTRANSLATOR".

Para esta app vamos a hacer un diseño de pantalla muy sencillo, únicamente con una etiqueta descriptiva de la app, unas imágenes estáticas para albergar las dos banderas, un botón que le permita al usuario activar el reconocimiento de voz para iniciar la tradución, una segunda etiqueta para contener el texto traducido al fracés, y algunas imágenes vacías de separación (ver figura).



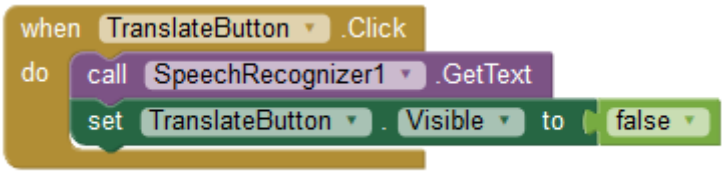
También necesitaremos añadir tres componentes no visibles: Un componente "SpeechRecognizer" para la conversión de la voz del usuario a texto (en castellano), un componente "YandexTranslate" para realizar la traducción del texto en castellano a texto en francés, y un componente "TextToSpeech" leer en voz alta el texto en francés.[10]

Para realizar una correcta lectura en voz alta del texto en francés, debemos configurar adecuadamente algunas propiedades del componente "TextToSpeech": La propiedad "Country" debe fijarse a "FRA", y la propiedad "Language" a "fr". Con ello especificamos una lectura a idioma francés de Francia.

7.19. AÑADIR COMPORTAMIENTOS A LOS COMPONENTES DE "MYTRANSLATOR".

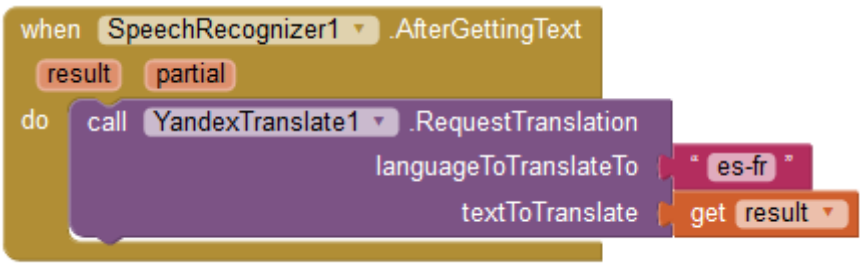
PROGRAMAR EL BOTÓN "TRANSLATEBUTTON".

En esta app queremos que, cuando el usuario pulse el botón "TranslateButton", el componente "SpeechRecognizer1" se ponga a escuchar la frase en castellano que el usuario le indique oralmente. En ese momento haremos que el botón deje de ser visible temporalmente, porque la app ya está realizando una traducción:



TRADUCIR AL FRANCÉS EL TEXTO RESULTADO DEL RECONOCIMIENTO DE VOZ.

Después de que el componente "SpeechRecognizer1" haya generado el texto resultado del reconocimiento del voz, queremos traducir ese texto en castellano a un texto en francés. Para ello utilizaremos el bloque de función "RequestTranslation" del componente "YandexTranslate1". La figura muestra el programa necesario:

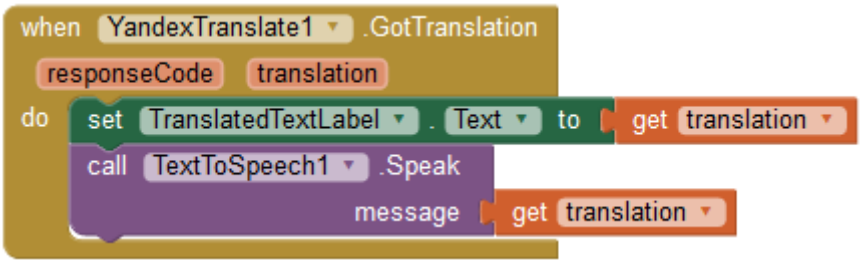


¿Cómo funciona este código? Cuando el componente "SpeechRecognizer1" obtiene el texto resultado del reconocimiento de voz lanza el evento "AfterGettingText", que capturamos con el manejador de evento correspondiente. El manejador "when (SpeechRecognizer1).AfterGettingText" incluye el argumento "result", donde se almacena el texto resultado de la conversión. En respuesta al evento "AfterGettingText", le solicitamos al componente "YandexTranslate1" que realice la traducción del texto de español a francés, mediante la función "(YandexTranslate1).RequestTranslation". Esta función necesita dos parámetros para funcionar: En el parámetro "languajeToTranslateTo" debemos especificar con un bloque de texto la conversión de idiomas que queremos realizar ("es-fr" en este caso), y en el parámetro "textToTranslate" debemos indicar qué texto queremos traducir (en este caso, el texto resultado del reconocimiento de voz almacenado en el argumento "result", y accesible mediante un bloque "get (result)").

[10] Al igual que el servicio de reconocimiento de voz Google Voice, el servicio de traducción Yandex Translate también necesita de conexión a Interner para funcionar.

MOSTRAR POR PANTALLA Y LEER EL TEXTO TRADUCIDO.

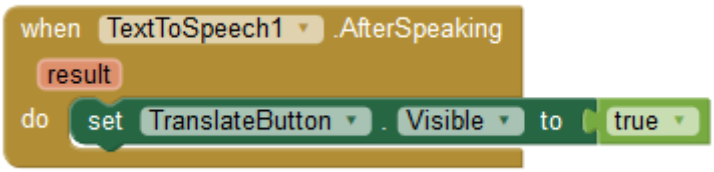
Una vez hemos traducido el texto a francés, queremos que la etiqueta "TranslatedTextLabel" lo muestre por pantalla, y que el componente "TextToSpeech" lo lea en voz alta. El programa que se encarga de ello es el siguiente:



¿Cómo funciona este código? Cuando la función "(YandexTranslate1).RequestTranslation" ha terminado de realizar la traducción, el manejador "when (YandexTranslate1).GotTranslation" captura dicho evento, y almacena el texto traducido en su argumento "translation". En respuesta a dicho evento, el manejador copia el texto traducido en la propiedad "Text" de la etiqueta "TranslatedTextLabel", y a continuación, llama a la función "(TextToSpeech1).Speak" para leer el texto traducido en voz alta. En ambos casos, hemos usado un bloque "get(translation)" para referenciar el texto traducido a mostrar en la etiqueta y a leer en voz alta.

VOLVER A MOSTRAR EL BOTÓN.

Finalmente, y después de haber leído el texto traducido, queremos volver a mostrar el botón de traducción. La figura muestra el programa necesario:



Y con este último programa hemos terminado la app. Vamos a probarla: Esta app no la podremos probar en el Emulador porque los servicios de reconocimiento de voz y de traducción necesitan conexión a Internet. Ahora, y una vez cargada la app en el dispositivo, al pulsar en el botón de la flecha debería abrirse el servicio de reconocimiento de voz Google Voice. Al decir en voz alta la frase en castellano a traducir, la app debería mostrar la misma frase en francés en la etiqueta de texto en la parte baja de la pantalla, y también debería leer ese texto en voz alta.

Cuidado: Algunas redes WiFi bloquean el servicio Yandex Translate, y la app se queda colgada esperando la traducción. Si nos ocurre esto, y tras haber cargado la app en el móvil, podemos probar a desactivar la conexión por WiFi para que el dispositivo se conecte a Yandex mediante datos móviles.

7.20. MODIFICACIONES A "MYTRANSLATOR".

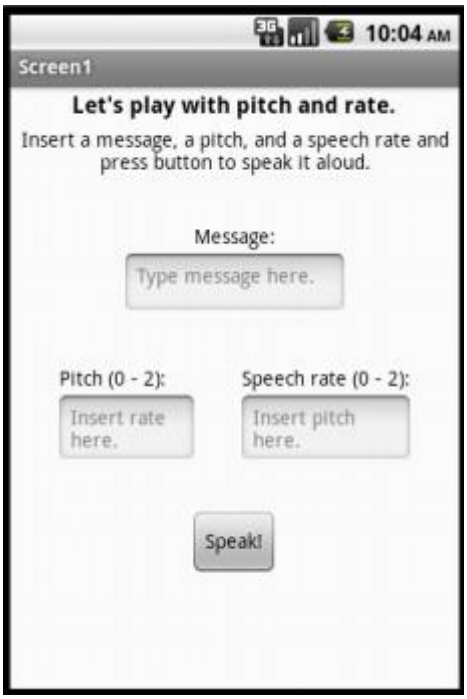
- Una primera mejora podría ser añadir un segundo botón que nos permita realizar la traducción inversa, esto es, que el usuario diga en voz alta una frase en francés, y que la app la muestre traducida al español y la lea en voz alta.
- Otra mejora sería construir una app que permita traducir del español a varios posibles idiomas, por ejemplo, al francés, al inglés, y al alemán. Para ello necesitaríamos varios botones, que le sirvan al usuario para especificar a qué idioma desea traducir.
- Una última modificación que podemos probar es hacer construir una app que permita realizar una taducción cruzada entre varios idiomas posibles. Así, si nuestra app incorpora los idiomas español, ruso,

y chino, la app sería capaz de traducir del español al chino, del español al ruso, del rsuao al español, del rudo al chino, del chico al español, y del chino al ruso.

7.21. EJERCICIOS DEL CAPÍTULO 7.

Ejercicio 7.1. Tono y ritmo de la locución.
Escribe una app en la que el usuario escriba un mensaje en una caja de texto. Cuando el usuario presione un botón de "Speak!", la app leerá en voz alta ese mensaje. En otras dos cajas de texto el usuario también podrá indicar el tono ("Pitch") y la velocidad ("SpeechRate") con los que se leerá el mensaje. El tono y la velocidad de la locución son dos propiedades del componente "TextToSpeech", cuyos valores podemos fijar con código mediante los bloques "set (TextToSpeech).Pitch to" y "set (TextToSpeech).SpeechRate to". La propiedad "Pitch" admite valores entre 0 y 2, donde 0 es un tono agudo y 2 es un tono grave. La propiedad "SpeechRate" también admite valores entre 0 y 2, donde 0 se corresponde con una locución lenta y 2 con una locución rápida.

Esta app debería inicializar correctamente el país y el idioma del componente "TextToSpeech" para que el mensaje se lea con el acento adecuado. Además, la app debería comprobar si el usuario ha insertado unos valores correctos para las cajas de texto del tono y del ritmo de la locución.



Ejercicio 7.2. DictaSMS.
Vamos a construir una aplicación en la que el usuario inserta un número de teléfono para enviarle un mensaje de texto. Pero en lugar de escribir el mensaje de texto, vamos a permitir que el usuario pueda dictar en voz alta el mensaje a enviar, y que la app muestre la transcripción del mensaje antes de enviarlo. La interfaz de la app debería parecerse a la de la figura.

Esta app debería comprobar que el usuario ha insertado un número de teléfomno antes de permitirle hablar en voz alta el mensaje a enviar. Además, la app debería tener deshabilitado el botón de enviar el ensaje hasta que se haya reconocido el mensaje hablado por el usuario.

8. SENSORES E INTEGRACIÓN DE CONTENIDOS WEB.

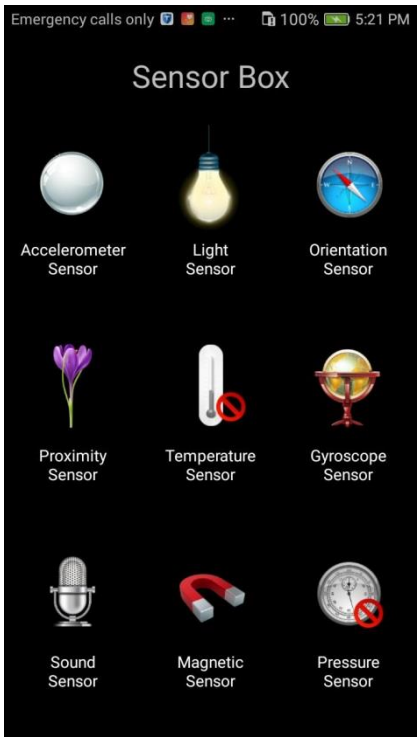
8.1. INTRODUCCIÓN.

En este capítulo aprenderemos a programar apps que emplean algunos de los componentes sensores que incluye App Inventor.También estudiaremos los componentes que les permiten a nuestras apps acceder a la información disponible en la web.

8.2. SENSORES.

Si apuntamos con nuestro teléfono al cielo, la app Google Sky Map nos indica hacia qué estremma estamos mirando. En algunos videojuegos de carreras podemos controlar la dirección del coche inclinando suavenete el dispositivo a derechas o izquierdas. Existen apps que son capaces de registrar la ruta que hemos recorrido en nuestra última sesión de footing. Incluso hay teléfonos que se ponen automáticamente en modo No Molestar cuando sus pantallas están mirando hacia abajo sobre una mesa. Todas estas apps funcionan porque los dispositivos móviles incorporan sensores para detectar la localización, la orientación, y la aceleración.

En las próximas secciones exploraremos algunos de los componentes sensores que incorpora App Inventor, como el sensor de localización (componente "LocationSensor"), el sensor de orientación (componente "OrientationSensor"), y el sensor de aceleración (componente "AccelerometerSensor"). Además, también aprenderemos algunos conceptos básicos del sistema de posicionamiento global GPS, cómo se mide la orientación mediante las medidas del pitch, roll, y azimuth, y entenderemos algunas técnicas matemáticas para procesar las lecturas del sensor acelerómetro.



> NOTA: Para poder probar las apps que de este capítulo necesitaremos un teléfono real, porque el emulador no dispone de ningún tipo de sensor. Para comprobar qué sensores incluye nuestro dispositivo móvil, podemos descargar de Google Play Store la app gratuíta Android Sensor Box de iMobLife (ver figura). Tal vez descubramos que nuestro dispositivo integra algunos sensores a los que App Inventor todavía no puede acceder (como por ejemplo, un sensor de proximidad o un sensor de luz). Pero no debemos decepcionarnos; App Inventor terminará incluyendo estas características, cuando este tipo de sensores sean habituales en la mayoría de dispositivos.

8.3. SENSOR DE LOCALIZACIÓN.

Los actuales dispositivos móviles son prácticamente ordenadores en miniatura que podemos llevar de un sitio a otro en nuestro bolsillo. El hecho de poder llevar con nosotros un miniordenador pone a disposición de las apps una información muy interesante: La localización actual del dispositivo. Conocer la ubicación de los dispositivos (y por tanto, de los usuarios de dichos dispositivos) puede ser muy útil en la vida diaria de las personas, pero también podría usarse incorrectamente para invadir nuestra privacidad.

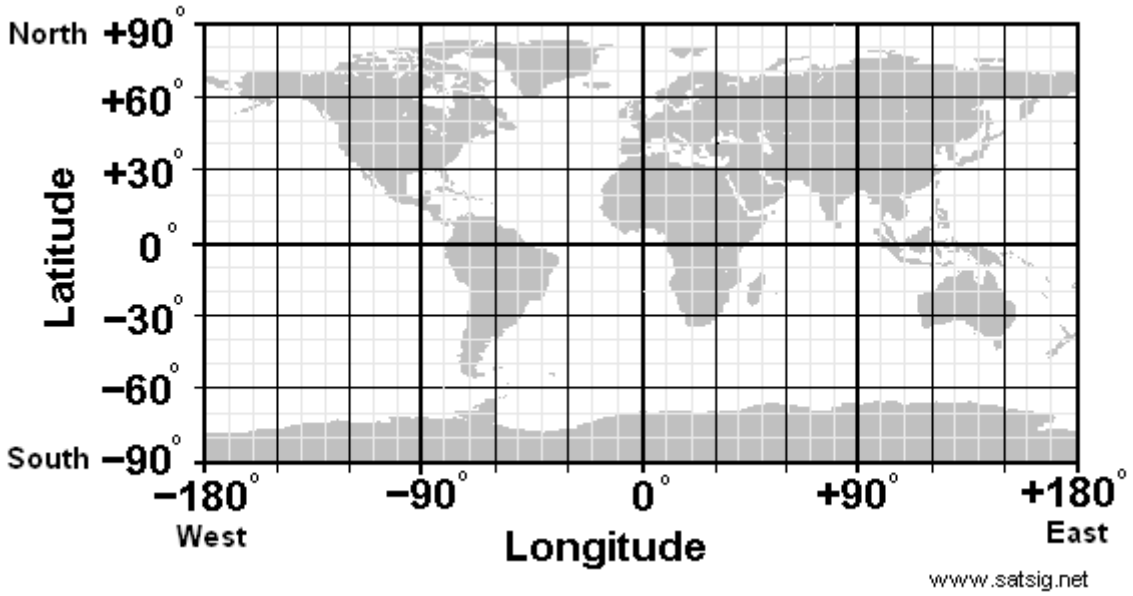
Un ejemplo de aplicación muy útil es la app "AndroidWhere" que construiremos más adelante en este capítulo. Esta app nos permite registrar una localización previa para poder volver a ella más adelante. Esta app es privada, ya que la información de nuestra localización se almacenara en una base de datos enn el propio dispositivo.

Sin embargo, la información de localización compartida también puede ser muy útil: Imaginemos un grupo de senderistas que necesiten conocer la localización de cada uno de sus miembros, para poder encontrar a uno de ellos que podría haberse extraviado.

Otro tipo de apps que utilizan la información de localización son las que utilizan herramientas de realidad aumentada. Estas apps usan nuestra posición y la orientación del móvil para proporcinar información útil acerca del entorno en el que nos encontramos. Así, imaginar una app de una inmobiliaria que nos permitiese apuntar nuestro teléfono a una casa para saber su precio de venta o de alquiler, o una app de un jardín botánico que al pasar cerca de cada planta nos informe acerca de su nombre, especie, y características.

EL SISTEMA DE POSICIONAMIENTO GLOBAL.

Antes de crear apps que registren y utilicen la localización, es conveniente entender cómo funciona el **sistema de posicionamiento global**, o GPS. Los datos de geolocalización se generan en una constelación de 24 satélites geosíncronos manenidos por el gobierno de los Estados Unidos. Si tenemos línea de visión de al menos tres de los satélites del sistema, nuestro dispositivo puede obtener una lectura. Una lectura GPS consiste en la información de la latitud, longitud, y altitud a la que se encuentra el dispositivo. La **latitud** es lo lejos que estamos hacia al norte o hacia el sur en relación al **ecuador**, donde los valores hacia el norte son positivos y hacia el sur negativos. El rango de valores posibles para la latitud es de -90 a 90 grados. El ecuador se localiza en una latitud de $0{,}0$ grados. La **longitud** es lo lejos que estamos hacia el este o hacia el oeste en relación al **Meridiano de Greenwich**, donde los valores hacia el este son positivos y hacia el oeste negativos. El Meridiano de Greenwich es una línea imaginaria que cruza el planeta de norte a sur que pasa a través de Greenwich, una localidad cercana a Londres donde se ubica el Observatorio Real, la organización que estableció el Meridiano de Greenwich como base de las medidas para astrónomos y navegantes. El rango de valores posibles para la longitud es de -180 a 180 grados. El Meridiano de Greenwich se ubica en una longitud de $0{,}0$ grados.

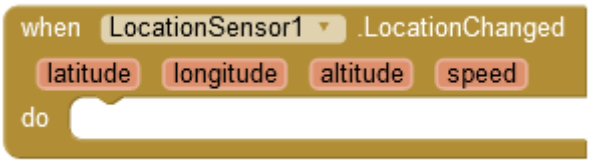


EL COMPONENTE "LOCATIONSENSOR".

App Inventor proporciona el componente "LocationSensor" para acceder a la información de localización GPS del dispositivo móvil. Este componente tiene unas propiedades llamadas "Latitude", "Longitude", y

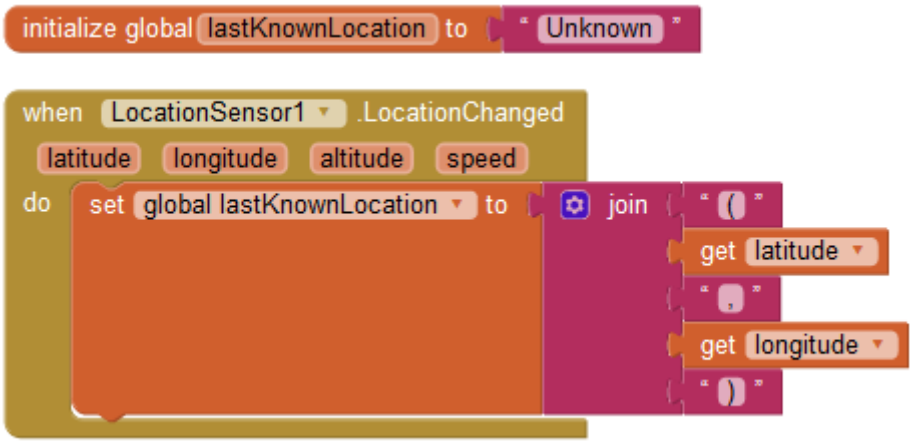
"Altitude" para almacenar las lacturas de latitud, longitud, y altitud registradas por el receptor GPS del dispositivo. Este componente también se comunica con Google Maps, para poder obtener una lectura de la dirección de la calle en la que estemos.

El principal manejador de evento de este componente es el bloque "when (LocationSensor).LocationChanged" mostrado en la figura:



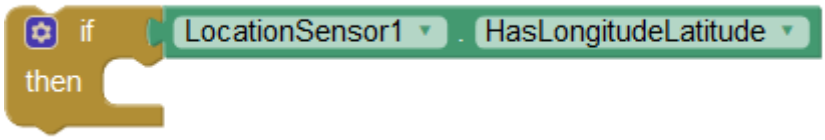
Este evento se dispara la primera vez que el sensor adquiere una lectura, y el resto de veces que el dispositivo se mueve lo suficiente como para que se reciba otra lectura con nuevos datos. A menudo suele haber un retraso de unos cuantos segundos antes de que la app registre la primera lectura, y en ocasiones el dispositivo no podrá hacer una lectura. Por ejemplo, si estamos en casa y no estamos conectados a una red WiFi, el dispositivo podría no obtener una lectura. Nuestro dispositivo también dispone de una opción para apagar el sensor GPS y ahorrar batería, y ésta es otra de las razones por las que el componente podría no obtener una lectura. Por todas estas razones, podría ocurrir que las propiedades del componente "LocationSensor" podrían no contener un valor hasta que no se dispare el evento "LocationChanged".

Una forma de tratar con la ausencia de datos de localización es crear una variable (por ejemplo, "lastKnownLocation", esto es, última localización conocida), inicializarla a "unknown" (es decir, desconocida), y a continuación hacer que el manejador de evento "when (LocationSensor).LocationChanged" cambie el valor de esa variable, como muestra el código de la figura:



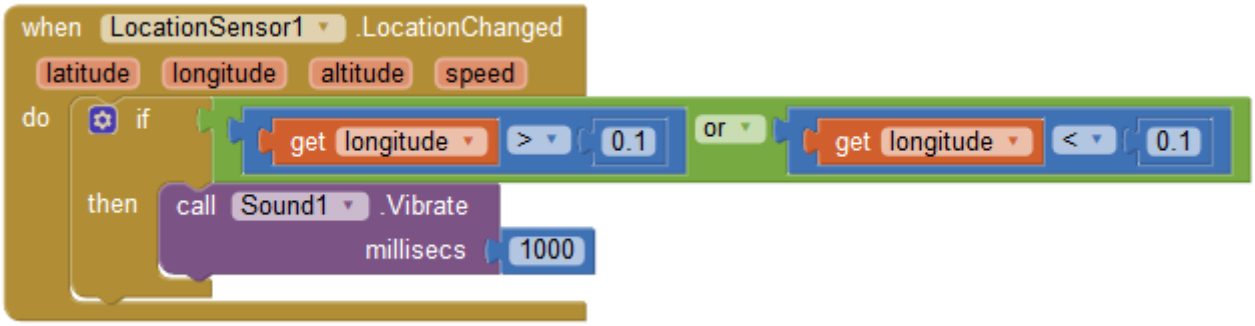
Programando el evento "when (LocationSensor).LocationChanged" de esta forma, siempre podremos mostrar o guardar en una base de datos la localización actual, indicando "unknown" (desconocida) hasta que la app no consiga registrar la primera lectura.

También podemos preguntar directamente al componente si el sensor ya ha adquirido una lectura mediante el bloque "(LocationSensor).HasLongitudeLatitude" ilustrado en la figura:



COMPROBACIÓN DE DEMARCACIONES.

Un uso muy común del evento "LocationChanged" es comprobar si un dispositivo está dentro de una demarcación o zona acotada. A modo de ejemplo, consideremos el código de la figura, que hace que el dispositivo vibre cada vez que el usuario se haya movido a una zona a $0{,}1$ grados de longitud más allá del Meridiano de Greenwich:



Estas comprobaciones tienen numerosas aplicaciones, como por ejemplo, avisar a los padres o profesores si un niño abandona el colegio.

FUENTES DE INFORMACIÓN DE LOCALIZACIÓN: GPS, WIFI, Y CELL ID.

Los dispositivos Android pueden determinar su localización de varias formas. El método más preciso (con un aprecisión de unos pocos metros) es a través del sistema GPS. Elmúico inconveninete es que no podremos obtener una lectura si estamos bajo techo, o en una zona con muchos edificios altos u otros obstáculos para la señal GPS. Recordemos que para poder recibir una lectura GPS necesitamos una línea de visión despejada de al menos tres satélites.

Si el GPS no está disponibe, o si el usuario lo ha deshabilitado, el dispositivo todavía puede determinar su posición mediante una red inalámbrica. Para ello, debemos estar cerca de un router WiFi, y la lectura de la posición que obtendremos srá la latitud y longitud de esta estación WiFi.

Una tercera opción mediante la que un dispositivo puede determinar su posición es mediante el *Cell ID* (identificador de celda). El Cell ID proporciona la localización del teléfono basándose en la intensidad de las señales emitidas por las torres de telefonía móvil más cercanas. Este mecanismo no es muy preciso a no ser que haya varias torres próximas a la zona donde se encuentra el dispositivo. Sin embargo, este método consume mucha menos batería en comparación con la conectividad GPS o WiFi.

8.3. COMENZAR CON LA APP "NOTEXTINGWHILEDRIVING2".

En el capítulo previo construimos la app "NoTextingWhileDriving1", que nos leía el contenido de un mensaje de texto SMS recibido en nuestro dispositivo móvil, y lo respondía automáticamente con un mensaje personalizable. Ahora vamos a modificar esta app para añadir la información de nuestra localización en esa respuesta automática.

Para ello, abrimos la app "NoTextingWhileDriving1" en App Inventor, y guardamos una nueva copia seleccionando la opción "Projects" → "Save project as…". Guardamos la copia del proyecto con el nombre "NoTextingWhileDriving2". Esta nueva app será una copia idéntica de "NoTextingWhileDriving1" sobre la que añadiremos la nueva funcionalidad recién indicada.

8.4. DISEÑAR LOS COMPONENTES DE "NOTEXTINGWHILEDRIVING2".

En esta nueva versión de la app solo necesitaremos añadir un nuevo componente: Un sensor de localización "LocationSensor" (disponible en la bandeja "sensors"). Se trata de un componente no visible que nada más soltarlo en el Visor, aparecerá en la zona inferior de componentes no visibles.

8.5. AÑADIR COMPORTAMIENTOS A LOS COMPONENTES DE "NOTEXTINGWHILEDRIVING2".

AÑADIR LA INFORMACIÓN DE LOCALIZACIÓN A LA RESPUESTA.

Como ya hemos mencionado antes, existen ciertas apps que son capaces de rastrean la localización del usuario. Aunque hoy día está en cuestión la legalidad de que ciertas aplicaciones nos tengan permanentemente localizados, hay apps que hacen un uso muy interesante y útil de esta información. Por ejemplo, hay apps que permiten encontrar a un niño perdido, o a un senderista que se haya salido de la ruta señalizada.

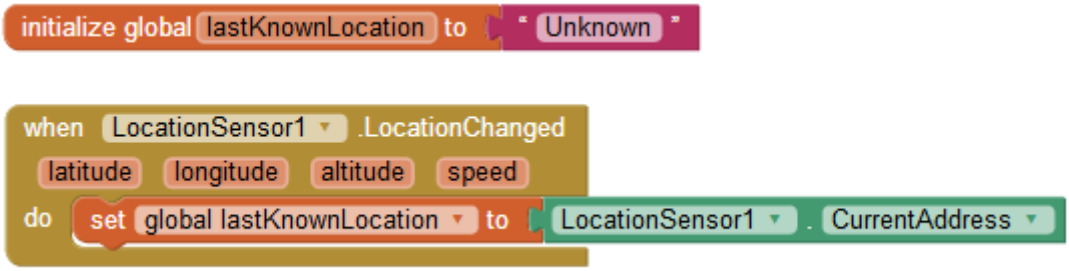
En esta segunda versión de la app vamos a usar la información de localización para añadir un poco más de información a la respuesta automática. En lugar de decir simplemente "I'm driving right now, I'll contact you shortly", podríamos construir un mensaje de respuesta del tipo "I'm driving right now, and I'm currently at 2431 Cherry Avenue". Esta información puede ser muy útil para un amigo o un familiar que esté esperando muestra llegada.

El componente "LocationSensor" nos permite acceder al GPS del teléfono para obtener la localización del dispositivo. Además de indicar la latitud y la longitud de la posición en la que se encuentra el teléfono, "LocationSensor" también utiliza Google Maps para proporcionar la dirección actual.

Recordar que el componente "LocationSensor" no siempre ofrece una lectura, y nuestro código debe asegurar que usamos el componente de forma correcta. En concreto, nuestra app debe responder adecuadamente al manejador de evento "when (LocationSensor).LocationChanged". El evento "(LocationSensor).LocationChanged" se dispara siempre que el sensor de localización del teléfono obtiene una primera lectura, y cuando el teléfono se mueve y genera una nueva lectura. Vamos a usar los bloques listados en la tabla para escribir el programa mostrado en la figura, que se encargará de responder al evento "(LocationSensor).LocationChanged" ubicando la dirección actual en una variable a la que llamaremos "lastKnownLocation". Más adelante cambiaremos el mensaje de respuesta para incorporar esta dirección.

Block type	Drawer	Purpose
`initialize global variable ("lastKnownLocation")`	Variables	Create a variable to hold the last read address.
`text ("unknown")`	Text	Set the default value in case the phone's sensor is not working.
`LocationSensor1.Location Changed`	LocationSensor1	This is triggered on the first location reading and every location change.
`set global lastKnownLoca tion to`	Drag from initialize global block.	Set this variable to be used later.
`LocationSensor1.CurrentAd dress`	LocationSensor1	This is a street address such as "2222 Willard Street, Atlanta, Georgia."

¿Cómo funciona el programa? El evento "(LocationSensor1).LocationChanged" se activa la primera vez que el dispositivo obtenga una lectura de su localización, y también cada vez que el dispositivo se mueva y se genere una nueva lectura. En tal caso, llamamos a la función "(LocationSensor1).CurrentAddress" para obtener la dirección actual del dispositivo, y la almacenamos en la variable "lastKnownLocation".



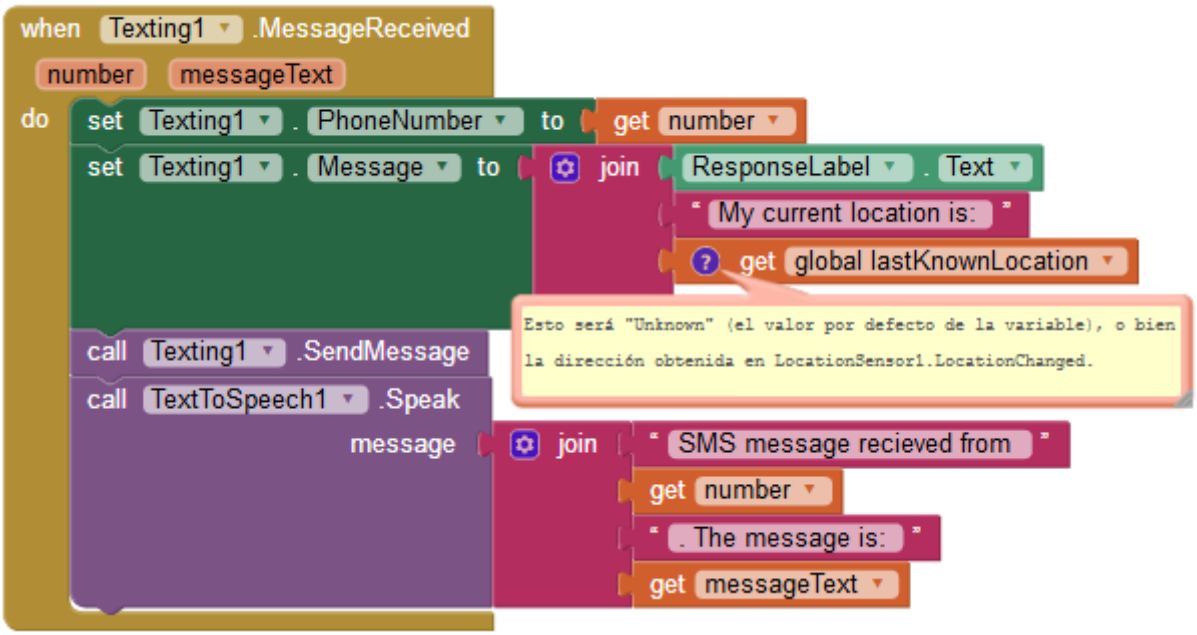
Notar que con estos bloques ya hemos hecho la mitad del trabajo. Ahora solo queda que la app incorpore esta información de localización al texto de la respuesta automática que se envía al emisor del mensaje SMS. Eso es precisamente lo que haremos a continuación.

ENVIAR LA LOCALIZACIÓN COMO PARTE DE LA RESPUESTA AUTOMÁTICA.

Usando la variable "lastKnownLocation" podemos modificar el manejador de evento "when (Texting1).MessageReceived" para añadir la información de localización a la respuesta automática. La siguiente tabla lista los bloques necesarios para hacerlo:

Block type	Drawer	Purpose
`join`	Text	concatenate some text together
`ResponseLabel.Text`	MessageTextBox	This is the (custom) message in the text box.
`text` ("My last known location is:")	Text	This will be spoken after the custom message (note the leading space).
`get global lastKnownLocation`	LocationSensor	This is an address such as "1600 Pennsylvania Ave NW, Washington, DC 20500."

El programa final debe quedar como muestra la figura:



¿Cómo funciona el programa? Este programa opera de forma conjunta con el manejador "when (LocationSensor1).LocationChanged" y la variable "lastKnownLocation". Como vemos en la figura, en vez de enviar directamente el mensaje contenido en "ResponseLabel.Text", la app construye un mensaje usando el bloque "join". Este bloque combina el texto de respuesta almacenado en "ResponseLabel.Text", el texto "My current location is: ", y el contenido de la variable "lastKnownLocation".

El valor por defecto de la variable "lastKnownLocation" es "Unknown", de forma que si el sensor de localización no ha generado una lectura, la segunda parte del mensaje de respuesta contendrá el texto "My current location is: Unknown". Si ya disponemos de una lectura, la segunda parte del mensaje será algo parecido a "My current location is: 1600 Pennsylvania Ave NW, Washington, DC 20500".

Para terminar, vamos a comprobar la app. Desde un segundo teléfono móvil enviamos un mensaje de texto al teléfono que está ejecutando la aplicación. El teléfono que ejecuta la app debería responder automáticamente incluyendo la información de localización. De no ser así, debemos asegurarnos de que en ese teléfono tenemos activados el GPS y el servicio de localización.

8.6. MODIFICACIONES A "NOTEXTINGWHILEDRIVING2".

- Podemos escribir una tercera versión de esta app que envíe respuestas personalizadas dependiendo de si el usuario está en unos ciertos márgenes de latitud y longitud. A mdo de ejemplo, la app podría determinar si estamos en el aula 222, y enviar un mensaje del tipo "Alejandro is in classroom 222 and can't text you back right now".

8.7. SENSOR DE ORIENTACIÓN.

Como ya hemos visto en capítulos previos, podemos usar el componente "OrientationSensor" para permitir que el usuario pueda controlar la acción de un videojuego inclinando el dispositivo. Pero también podemos utilizarlo como un arújula para determinar la dirección (norte/sur, este/oeste) en la que está apuntando el dispositivo.

El componente "OrientationSensor" reporta cinco lecturas que se almacenan en las propiedades correspondientes. Estas ropiedades no nos resultará familiares a menos que seamos pilotos de aviones o ingenieros aeronáuticos:

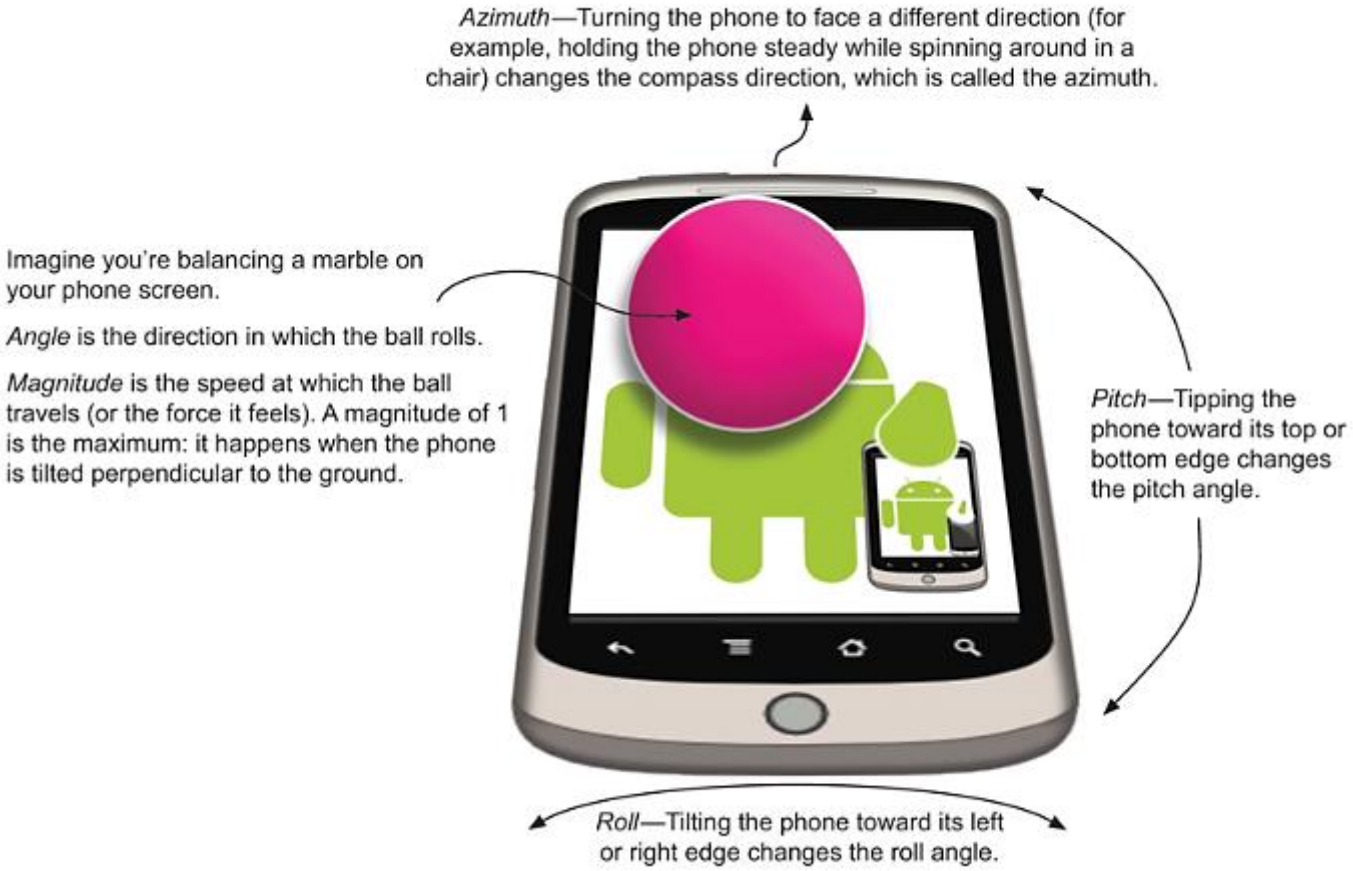
- "Roll" (alabeo): Izquierda – derecha.
  El ángulo de alabeo (propiedad "Roll") indica la inclinación del dispositivo en torno a un eje que lo atraviesa a lo largo de su longitud (ver figura). El alabeo es de $0°$ cuando el dispositivo está equilibrado, aumenta hasta los $90°$ cuando el dispositivo se inclina hacia su lado izquierdo, y disminuye hasta los $-90°$ cuando el dispositivo se inclina hacia su lado derecho.

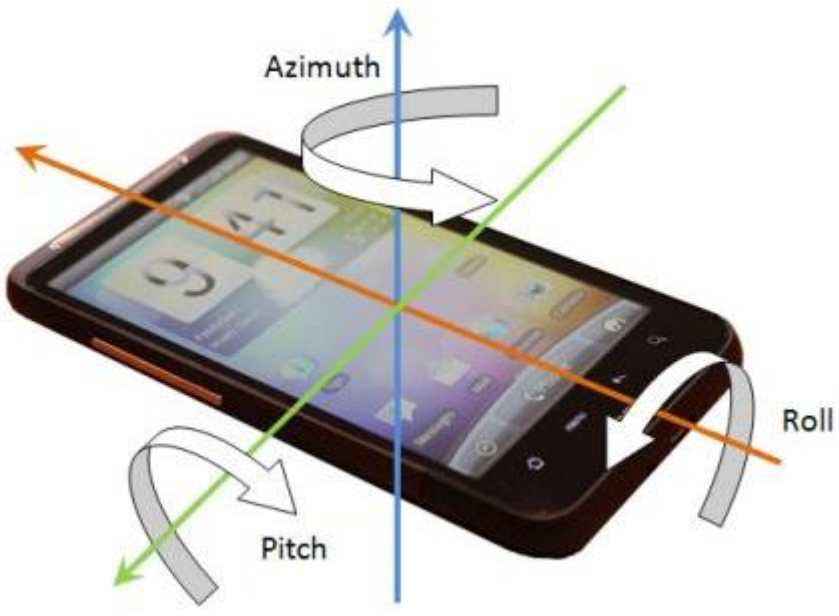
- "Pitch" (cabeceo): Arriba – abajo.
  El ángulo de cabeceo (propiedad "Pitch") indica la inclinación del dispositivo alrededor de un eje lateral a lo largo de su anchura (ver figura). El cabeceo es de $0°$ cuando el dispositivo está nivelado, aumenta hasta los $90°$ conforme el dispositivo se inclina de forma que su morro apunte hacia abajo, y llega hasta los $180°$ cuando le damos la vuelta. De forma análoga, cuando inclinamos el dispositivo de forma que su cola pasa a apuntar hacia abajo, el cabeceo empieza a disminuir hasta los $-90°$ y llega a tomar un valor de $-180°$ cuando le damos la vuelta en el otro sentido.

- "Azimuth" (guiñada): Brújula.
  El ángulo de guiñada (propiedad "Azimuth") indica el giro del dispositivo alrededor de un eje vertical perpendicular al dispositivo (ver figura). La guiñada es $0°$ cuando el morro del dispositivo está apuntando al norte, $90°$ cuando apunta al este, $180°$ cuando punta al sur, y $270°$ cuando punta al oeste.

- "Magnitude".
  La propiedad "Magnitude" devuelve un número entre 0 y 1 que indica cuánto se ha inclinado el dispositivo. Su valor proporciona la rapidez de una bola que estuviese rodando sobre la superficie del dispositivo.

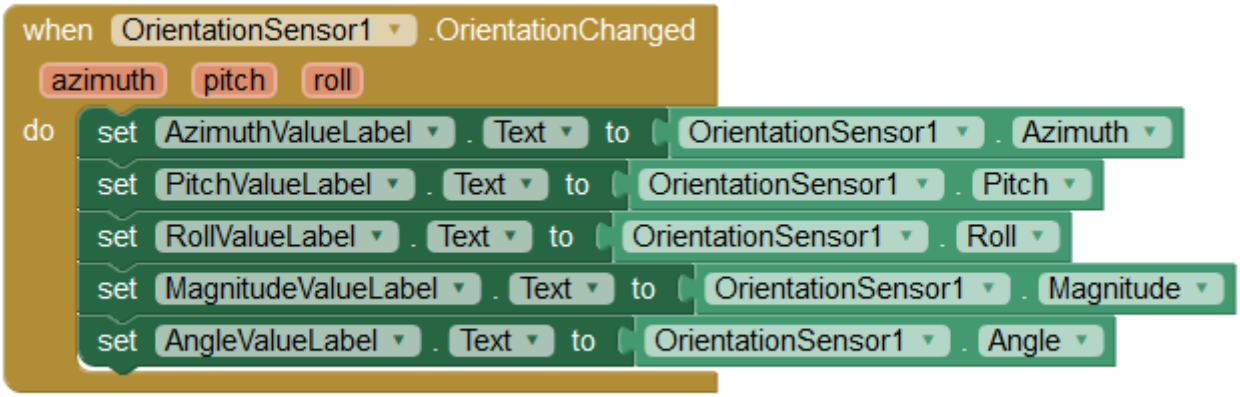
- "Angle".
  La propiedad "Angle" devuelve la dirección en la que se ha nclinado eldispositivo. Su valor indica la dirección de la velocidad de una bola que estuviese rodando sobre la superficie del dispositivo.





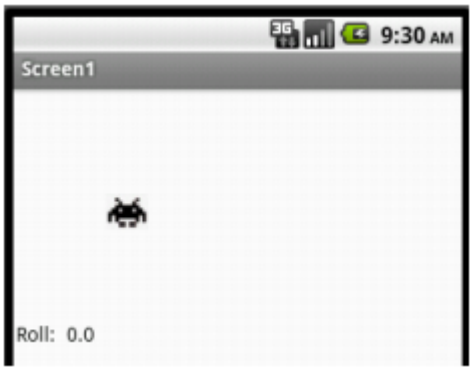
El componente "OrientationSensor" proporciona el evento "OrientationChanged", que se actva cada vez que cambia la orientación del dispositivo. Para entender estas propiedades, vamos a escribir una app que ilustre cómo cambian conforme el usuario inclina el dispositivo. La interfaz de usuario de esta app es muy sencilla,

ya que solo incluye cinco etiquetas estáticas para mostrar el nombre de cada propiedad, y otras cinco etiquetas para mostrar de forma dinámica el valor actual de esas propiedades. (Por supuesto, también necesitamos añadir un componente "OrientationSensor"). Una vez diseñada la pantalla de la app, escribimos el código mostrado en la figura:

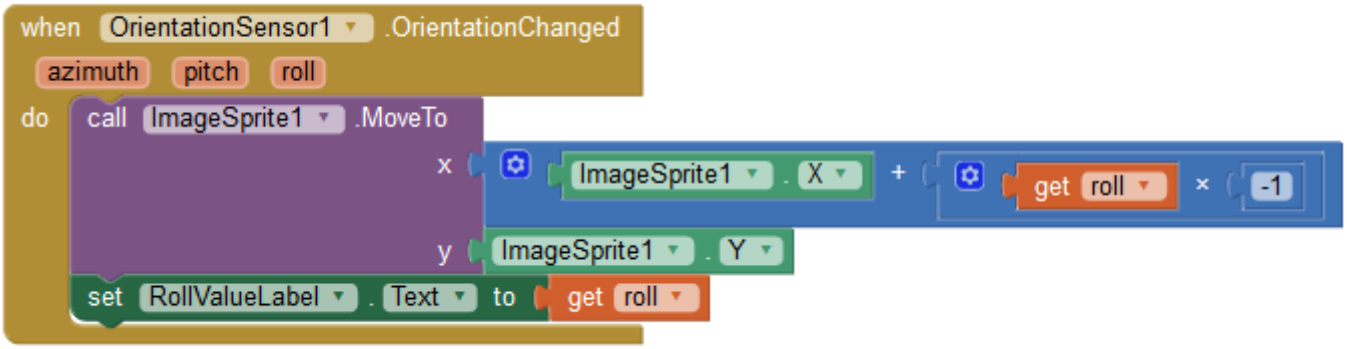


USAR EL PARÁMETRO "ROLL" PARA MOVER UN OBJETO.

Vamos a tratar de mover un sprite a la derecha o a la izquierda basándonos en la inclinación del dispositivo, tal y como haríamos en un videojuego de conducción. Para ello, arrastramos un "Canvas" al Visor, y fijamos sus propiedades "Width" y "Height" a "Fill parent" y 200 píxeles, respectivamente. A continuación añadimos un componente "ImageSprite" o un componente "Ball" al lienzo, y bajo él un par de etiquetas para mostrar el valor actual de la propiedad "Roll", como muestra la figura. (La imagen para el sprite se llama *RollSprite.png* y está disponible en los archivos de los alumnos).



La propiedad "Roll" del componente "OrientationSensor" nos indicará si el teléfono se ha inclinado hacia la izquierda o hacia la derecha. (Si mantenemos el teléfono bocaarriba y lo inclinamos ligeramente hacia la izqiuerda, obtendremos una lectura positiva; si lo inclinamos ligeramente hacia la derecha, obtendremos una lectura negativa). Por consiguiente, podemos hacer que el usuario mueva un objeto horizontalmente con un manejador como el de la figura:



Este código multiplica el valor del "Roll" por -1 porque inclinar el dispositivo hacia la izquierda proporciona un valor de "Roll" positivo, y ello debería mover el objeto hacia la izquierda (lo que hace la coordenada x más pequeña).

Notar que esta app solo funciona si el dispositivo está en modo "Portrait" (esto es, en vertical), y no en modo "Landscape" (apaisado). Si inclinamos demasiado el dispositivo hacia la izquierda, la pantalla cambiará a apaisado y el sprite quedará bloqueado en el lado izquierdo de la pantalla. Esto ocurre porque, cuando el

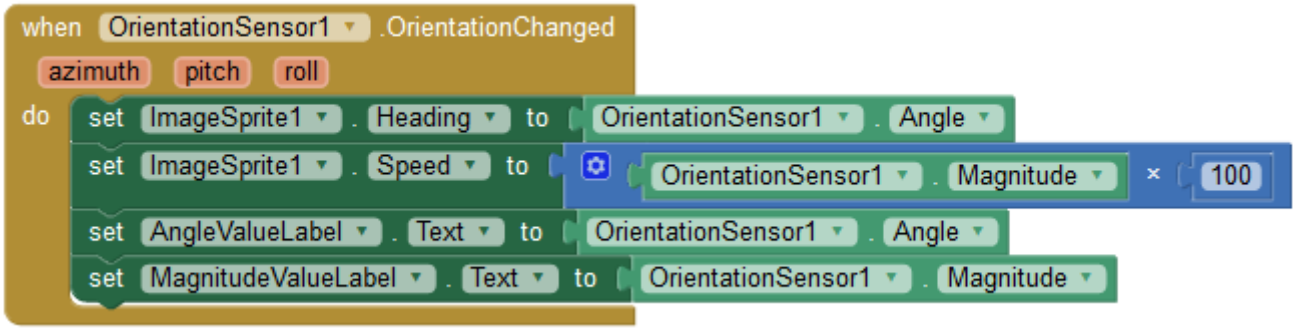
dispositivo está de lado, también está inclinado hacia la izquierda y siempre obtiene una lectura positiva para el "Roll". Una lectura positiva siempre hará que la coordenada x se vuelva más pequeña.

Recordar que App Inventor incluye la propiedad "Screen1.ScreenOrientation", que podemos usar para bloquear la orientación de la pantalla a vertical si no queremos que el dispositivo cambie automáticamente entre ambos modos.

USAR LOS PARÁMETROS "ANGLE" Y "MAGNITUDE" PARA MOVER UN OBJETO EN UNA DIRECCIÓN CUALQUIERA.

El ejemplo previo movía el sprite a la derecha o a la izquierda. Si queremos permitir un movimiento en una dirección cualquiera, podemos usar las propiedades "Angle" y "Magnitude" del componente "OrientationSensor". Estas fueron las propiedades que usamos para mover a la mariquita del juego que programamos en la app "LadubugChase" del capítulo 6.

En la siguiente figura podemos ver los bloques de una app de pruebas en la que el usuario inclina el dispositivo para mover un personaje en una dirección cualquiera. (Para este ejemplo necesitaremos algunas etiquetas para mostrar los valores actuales de "Angle" y "Magnitude" y un componente "ImageSprite").



La propiedad "Magnitude" toma valores entre 0 y 1, e indica cuánto se ha inclinado el dispositivo. En esta app, el sprite se mueve más rápido conforme aumenta el valor de "Magnitude".

USAR EL TELÉFONO COMO BRÚJULA.

Las aplicaciones tipo brújula y apps como Google Sky Map necesitan sabar la orientación del móvil en relación al mundo, indicando si mira al norte o sur, este u oeste. Sky Map usa esta información para mostrar datos de las constelaciones hacia las que el teléfon está apuntando.

La lectura del parámetro "Azimuth" es muy útil en estos casos. La propiedad "Azimuth" toma valores entre 0 y 360 grados, donde $0°$ es norte, $90°$ este, $180°$ sur, y $270°$ oeste. Así, una lectura de 45 grados significa que el teléfono está apuntando hacia el noreste, 135 grados significa sureste, 225 grados es suroeste, y 315 grados indica noroeste.

Para poner en práctica estas ideas, en la siguiente sección vamos a programar una brújula sencilla.

8.8. COMENZAR CON LA APP "COMPASS".

En esta sección vamos a programar una aplicación llamada "Compass" que implemente una brújula. Cuando movamos el teléfono, la brújula rotará de forma que el norte siempre se alinee con el polo norte magnético de la Tierra (siempre que no haya otros campos magnéticos más intensos en las proximidades del teléfono). Eso significará que la dirección de la brújula que está en la parte superior de la pantalla es la dirección hacia la que estamos mirando. La app también incluye una etiqueta que nos dice la dirección actual en

relación al norte. Por ejemplo, si estamos mirando hacia el sur, la letra S de la brújula estará en la parte superior del teléfono, y la etiqueta mostrará el valor 180 grados.

Para esta appp necesitaremos la imagen de una brújula, que podremos encontrar en la carpeta de los alumnos con el nombre *compass.png*.

8.9. DISEÑAR LOS COMPONENTES DE "COMPASS".

La tabla a continuación muestra los componentes que necesitaremos para construir esta app, junto con los valores a ajustar para sus propiedades:

Compass			
Screen1 properties	`Title`: Compass `AlignHorizontal`: Center `ScreenOrientation`: Portrait `Scrollable`: No (unselected)		
Components	**What do I rename it?**	**What does it do?**	**What properties do I set?**
`Label`	`HeadingLabel`	Displays the direction in which the top of the phone is pointing (in degrees)	`FontBold`: Yes (selected) `FontSize`: 28 `Text`: "Heading"
`Canvas`	`Canvas`	Holds the compass face	`Width` and `Height`: Fill Parent
`ImageSprite`	`CompassSprite`	Displays the compass face	`BackgroundImage`: compass.png `Rotates`: Yes (selected) `Width` and `Height`: Automatic
`OrientationSensor` Palette group: Sensors	`OrientationSensor1`	Tells you the azimuth—the phone's current heading based on the Earth's magnetic field	`Enabled`: Yes (selected)

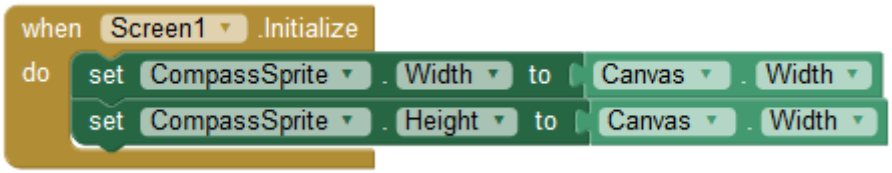
En este caso, es especialmente importante acordarnos de habilitar la propiedad "Rotates" del componente "CompassSprite", para permitir que la imagen del sprite rote para apuntar en la dirección en la que está apuntando el dispositivo.

8.10. AÑADIR COMPORTAMIENTOS A LOS COMPONENTES DE "COMPASS".

REDIMENSIONAR EL TAMAÑO DE LA IMAGEN DE LA BRÚJULA.

Una característica interesante de esta app es que la imagen de la brújula cambiará de tamaño automáticamente para ajustarse al tamaño de pantalla del dispositivo, independientemente del modelo de teléfono o de tableta que estemos usando. Para conseguir este funcionamiento debemos ajustar algunas propiedades de la pantalla y del lienzo, y programar algunos comportamientos para redimensionar la imagen de la brújula en función del tamaño de la pantalla. Notar que la altura y anchura del lienzo se han fijado a "Fill parent" para que el lienzo aduiqera automáticamente la altura y anchura de la pantalla del dispositivo. Solo con esto, la imagen de la brújula adquiriría una forma elíptica.

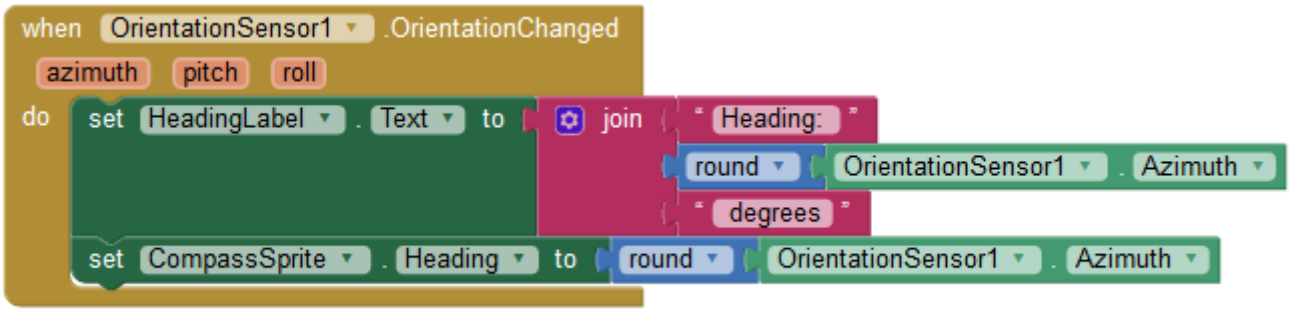
Para hacer que el sprite de la brújula tenga la misma altura y anchura que el lienzo al que pertenece, debemos escribir el siguiente código:



En este programa simplemenete usamos código para fijar la altura y la anchura de "CompassSprite" iguales a la altura y a la anchura del lienzo que lo contiene, lo que significa que la brújula tendrá una forma perfectamente circular, y que ocupará toda el ancho horizontal de la pantalla. Notar que estamos usando el manejador de evento "Screen1.Initialize" para ejecutar este código en cuanto arranque la app. El usuario probablemente ni notará que la imagen de la brújula está siendo redimensionada.

HACER QUE LA BRÚJULA APUNTE EN LA DIRECCIÓN CORRECTA.

En esta app necesitamos saber en qué dirección está apuntando el teléfono para actualizar la etiqueta y rotar la imagen de la brújula para hacerla concidir con esa dirección. Aquí vamos a usar la propiedad "Azimuth", que indica la dirección en la que está apuntando el morro del teléfono. Además, también queremos mostrar la lectura de "Azimuth" en la etiqueta "HeadingLabel" en la parte superior de la pantalla. El programa que codifica estos dos comportamientos se muestra en la figura:



¿Cómo funciona este código? Cada vez que el manejador de evento "when (OrientationSensor).OrientationChanged" detecte un cambio en la orientación del dispositivo, el programa cambiará la propiedad "Heading" del objeto "CompassSprite" para que pase a apuntar en la dirección en la que está apuntando el dispositivo. Este valor está almacenado en la propiedad "Azimuth" del sensor de orientación. Además, actualizará el texto mostrado por la etiqueta "HeadingLabel" para que indique la dirección en la que está apuntando el dispositivo con un texto del tipo "Heading: 217 degrees".

Ahora, la lectura del "Azimuth" proporciona un montón de decimales de precisión. Eso está bien si disponemos de un terminal altamente preciso y queremos usarlo en aplicaciones que requieran una gran exactitud, como por ejemplo, el guiado automática de un avión. Pero en nuestro caso solo necesitaremos ser precisos, digamos, hasta el grado más cercano. Por esta razón estamos redondeando la lectura del "Azimuth" usando el bloque "round" de la bandeja "Math".

Con esto hemos terminado. Pero antes de examinar otros sensores, vamos a usar otra vez el sensor de orientación (la brújula) para realziar un asombroso truco de magia.

8.11. SENSOR DE ACELERACIÓN.

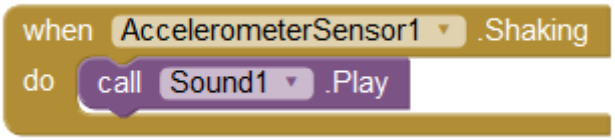
La **aceleración** es la tasa de cambio de la velocidad a lo largo del tiempo. Si pisamos el pedal del acelerador de nuestro coche, el coche se acelera, es decir, su velocidad aumenta a un cierto ritmo.

El sensor acelerómetro de nuesro dispositivo Android mide la aceleración, pero su marco de referencia no es el dispositivo en reposo, sino el dispositivo en caída libre. Esto significa que si dejamos caer el móvil, e sensor registrará una aceleración de 0. Dichode forma sencilla, las lecturas del sensor ya incluyen la gravedad.

Dejando de lado estas cuestiones puramente físicas, en esta sección vamos a explorar el uso del sensor acelerómetro. Incluso construiremos una app que podría servir para salvar vidas.

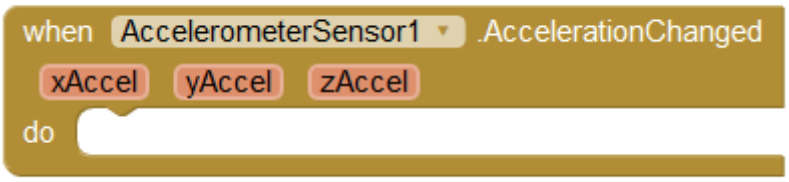
RESPONDER A LA SACUDIDA DEL DISPOSITIVO.

En el capítulo 2 implementamos la app "HelloPurr", en la que ya usamos el componente "AccelerometerSensor". En esa app usaamos el evento "(AccelerometerSensor).Shaking" para hacer que el gato maullase al agitar el teléfono, como muestra la figura:



USAR LAS LECTURAS DEL SENSOR ACELERÓMETRO.

Al igual que los otros sensores, el acelerómetro tiene un evento que se dispara cuando las lecturas cambian, a saber, "(AccelerometerSensor).AccelerationChanged". Este evento reporta tres argumentos, que se corresponden con las componentes de la aceleración en las tres dimensiones x, y, y z:

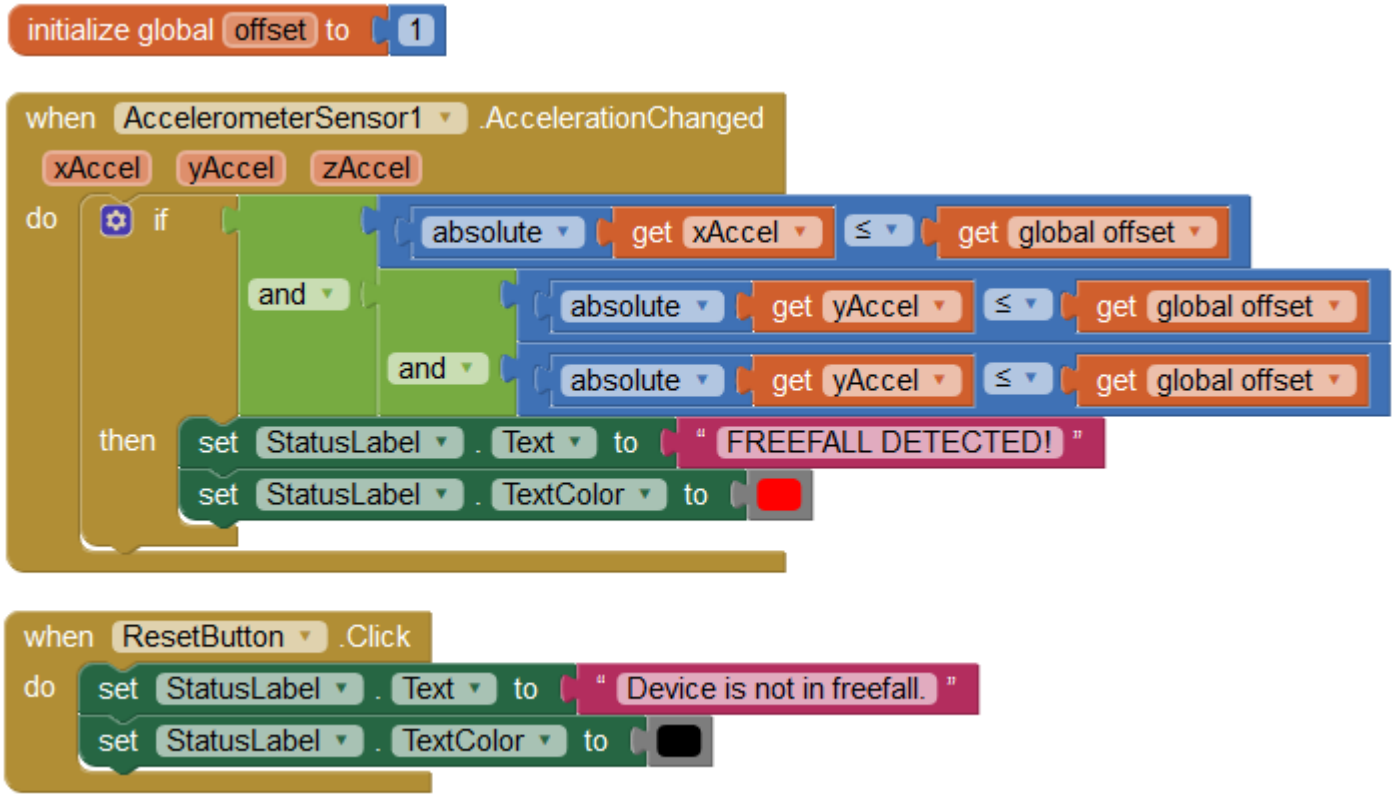


- "xAccel": Es positivo cuando inclinamos el dispositivo hacia la derecha (esto es, cuando elevamos su lado izquierdo), y negativo cuando inclinamos el dispositivo hacia la izquierda (es decir, cuando elevamos su lado derecho).
- "yAccel": Es positivo cuando elevamos la cola del dispositivo, y negativo cuando elevamos el morro.
- "zAccel": Es positivo cuando la pantalla del dispositivo está mirando hacia arriba, y negativo cuando está mirando hacia abajo.

DETECTAR UNA CAÍDA LIBRE.

Sabemos que si los registros de aceleración están próximos a 0, el dispositivo está en caída libre hacia el suelo. Con esto en mente, podemos detectar una caída libre comprobando las lecturas del sensor en el evento "(AccelerometerSensor).AccelerationChanged". Siguiendo este enfoque podríamos escribir un programa que detecte si una persona mayor se ha caído, y enviar auromáticamente un mensaje SMS de alerta (a un familiar, a la policía, etc.).

La figura a continuación muestra los bloques de una app que simplemente reporta la ocurrencia de un evento de caída libre (y que le permite al usuario pulsar un botón de Reset para volver a comprobarlo).[11]



Cada vez que el sensor obtenga una nueva lectura, los bloques comprueban las dimensionaes x, y, y z para comprobar si están próximas a 0 (si su valor absoluto es menor que 1). Si las tres están conjuntamente cerca de 0, la app cambia una etiqueta de estado para indicar que el teléfono está en caída libre. Cuando el usuario pulsa el botón "ResetButton", la etiqueta vuelve a su estado original ("Device is not in freefall").

8.12. ACCEDER A PÁGINAS WEB DESDE NUESTRA APP.

En la segunda parte de este capítulo vamos a explorar los componentes que nos permiten acceder a la información de la Web. Aprenderemos cómo mostrar una página web dentro de la interfaz de usuario de nuestra app, y también presentaremos las APIs y cómo acceder a la información de un servicio web.

Hoy día, las apps y las páginas web suelen incluir contenidos que provienen de múltiples fuentes de datos, y la mayoría de los sitios web se diseñan para permitir la interoperabilidad entre ellos.

App Inventor ofrece dos opciones para acceder a la web desde nuestra app:

1) El componente "WebViewer": Este componente nos permite mostrar páginas web dentro de nuestra app. Así, podemos usar el componente "WebViewer" para mostrar una página de Google Maps que indique la posición actual del usuario, una página Twitter que muestre los temas candentes (trending topics) relacionados con nuestra app, o una página del sitio nba.com que muestre las estad´´isticas de nuestros jugadores favoritos.

   El componente "WebViewer" es como el componente "Canvas", en el sentido que define un subpanel en la pantalla. Pero mientras que el componente "Canvas" se usa para las animaciones y para los dibujos, el

[11] Podemos pinchar dos veces con el botón derecho del ratón sobre un bloque, y elegir las opciones "Inline inputs" y "External inputs" para cambiar la forma en la que se muestran los bloques. Eso es precisamente lo que hemos hecho con los dos bloques "and" para reducir la anchura del manejador de eventos.

componente "WebViewer" muestra una página web. Odeos iaginar que el "WebViewer" es como un mini navegador que se incrusta en la pantalla de neustra app, pero que no dispone de las funciones avanzadas que ofrecen navegadores como Firefox, Chrome, Safari, etc.

2) El componente "ActivityStarter": Este componente se sale de nuestra app y muestra las páginas web usando el navegador que tengamos instalado en nuestro teléfono. El componente "ActivityStarter" le permite al usuario ver la página web en la aplicación de navegador que elija, de forma que pueda utilizar todas las funcionalidades propieas de ese navegador. También es posible no permitir que el usuario elija, y especificar exactamente que navegador se usará. (Esta opción no es recomendable, porque si el usuario no tiene instalado ese navegador en su dispositivo, la app producirá un error). El usuario puede volver del navegador a la app cerrando la app del navegador.

La ventaja del componente "WebViewer" es que es sencillo, y nestra app sigue controlando la pantalla (el usuario no se perderá en las pantallas de otras apps que estén abiertas). La ventaja del componente "ActivityStarter" es que el usuario puede elegir el navegador al que esté más acostumbrado, y podrá acceder a características que el componente "WebViewer" no ofrece, como añadir una página web a sus favoritos.

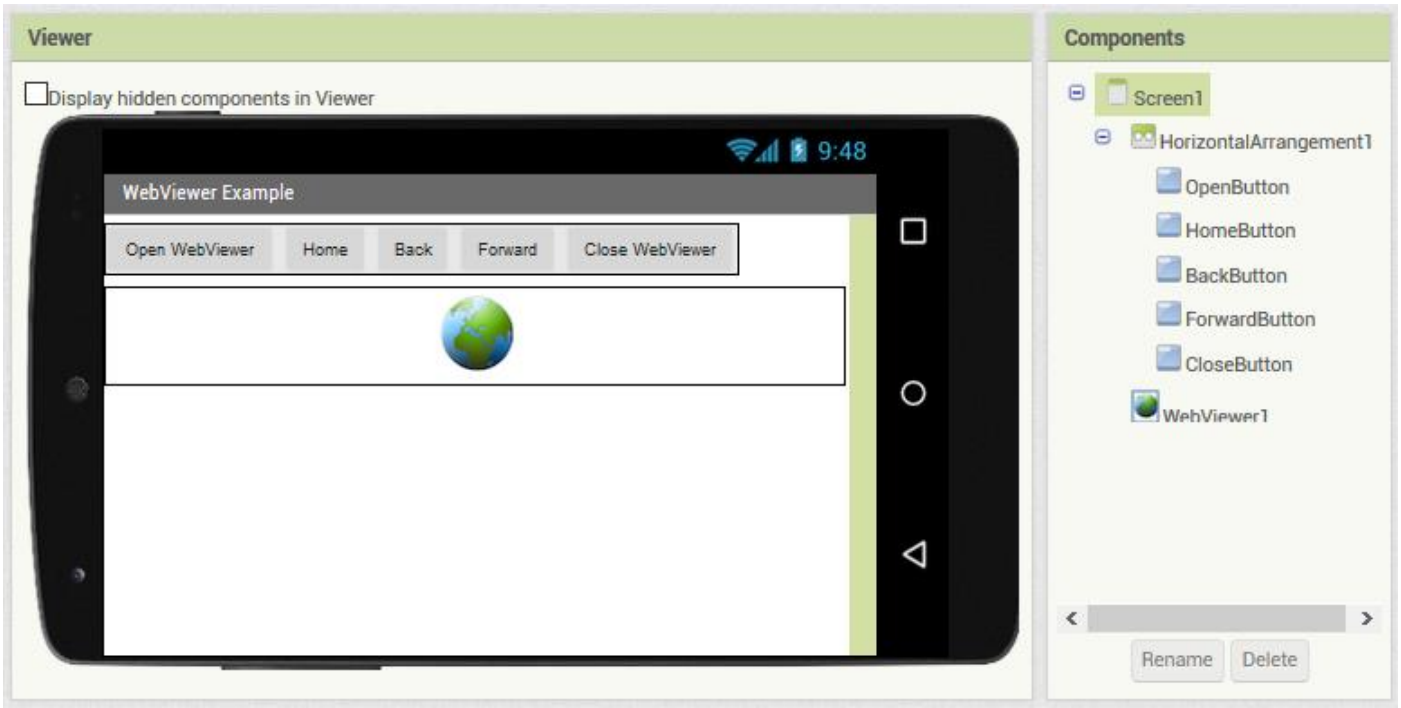
Vamos a poner en práctica ambos componentes.

USAR EL COMPONENTE "WEBVIEWER".

Para probar el componente "WebViewer" vamos a construir una nueva app a la que llamaremos "TestWebViewer". Para construirla, seguimos los siguientes pasos:

- En la propiedad "Screen1.Title" escribimos el texto "WebViewer example", y fijamos la propiedad "Screen1.ScreenOrientation" a "Landscape".

- A continuación añadimos un organizador "HorizontalArrangement" con cinco botones, como muestra la figura:



- Añadimos un componente "WebViewer" de la paleta "User Interface", y fijamos las propiedades como indicamos:

  - "FollowLinks": Esta propiedad le permite al "WebViewer" alamcenar un historial de páginas para que los botones "Back" y "Forward" puedan funcionar. Activamos esta propiedad.
  - "HomeUrl": Sirve para indicar la dirección web que se abrirá cuando arranque la app. Aquí heos puesto la página detecnología del periódico El País, https://www.elpais.com/tecnologia/.
  - "PromptForPermission" y "UsesLocation" impiden que las páginas web puedan acceder a uesra localización, a menos que les demos permido. Para ello, activamos "PromptForPermission" y desactivamos "UsesLocation".

Con esto hemos construido una interfaz de usuario básica para una app de navegador web básico. Podríamos mejorarlo, por ejemplo, añadiendo una caja de texto en la que el usuario pudiera escribir una dirección web (URL). Sin embargo, es bastante improbable que un usuario elija usar el componente "WebViewer" de App Inventor para navegar por la web. Este componente suele usarse para proporcionar un enlace específico a la página web a la que queremos llevar al usuario desde nuestra app. Así, por ejemplo, podríamos usar el componente "WebViewer" para lo siguiente:

- Proporcionar información adicional (ayuda, pistas, y trucos) para un videojuego que hayamos hecho.

- Enlazar con Google Play Store, donde podríamos tener otros juegos nuestros a la venta.

- Configurar una encuesta online (usando Google Docs o SurveyMonkey) para saber qué opinan los usuarios de nuestra app.

  NOTA: **URL (Localizador Uniforme de Recursos)**.
  El nombre técnico correcto para una dirección web es URL, o Localizador Uniforme de Recursos (Uniform Resource Locator). Un URL como http://www.elpais.com/tecnologia/ contiene tres elementos:

  - http:// son las siglas de HyperText Transfer Protocol, y significa que vamos a ver las páginas web con hipertextos, esto es, con enlaces. En ocasiones veremos https://, que es la versión segura del protocolo http.

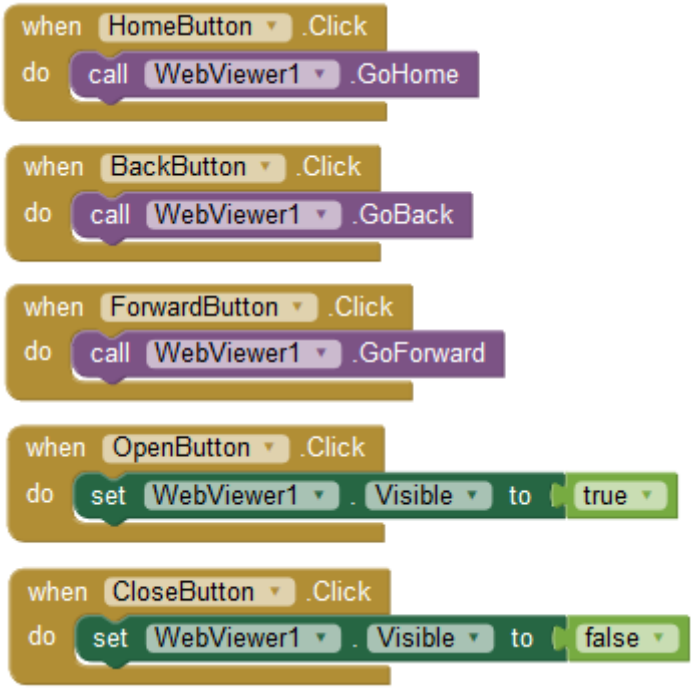
  - www.elpais.com es el nombre de dominio, esto es, la etiqueta identificativa de la dirección IP única del ordenador que aloja esta página web, y al que queremos acceder. (El .com indica que es una empresa, com = company).

  - /tecnología/ es la ruta del archivo concreto al que deseamos acceder, y funciona igua que las carpetas de nuestro ordenador.

  Cuando una URL termina con un nombre de archivo, como /instructions.html, la app recupera ese archivo. Si la dirección termina con la barra /, la app buscará un archivo llamado index.html en esa ruta.

Si probamos la app tal y como está, veremos que la página de tecnología de El País aparece en la pantalla de nuestro dispositivo, y podremos navegar entre las distintas páginas pulsando los hiperenlaces. Pero los botones que hemos creado todavía no hacen nada.

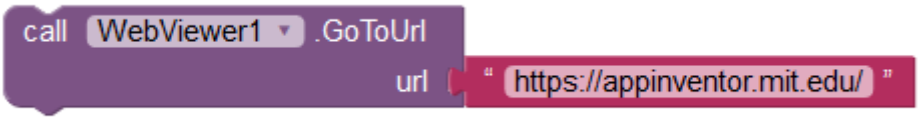
Para dotarles de funcionalidad, vamos a escribir los programas que muestra la figura. Como podemos ver, el componente "WebViewer" incluye muchas funciones preconstruidas para navegar, lo que nos permite codificar fácilmente el comportamiento de los botones:



La experiencia de navegación mediante "WebViewer" adolece de algunas limitaciones:

- Los botones desaparecen cuando nos desplazamos bajando por la página web. Si queremos usarlos, tendremos que volver a desplazarnos hacia arriba hasta la parte superior de la pantalla.
- Si el usuario pulsa el botón "Atrás" del teléfono (en vez de pulsar el botón "Back" que nosotros hemos creado), sale de la app.

También es posible cambiar la páina web actual mediante código, usando el bloque "call (WebViewer1).GoToUrl". En el ejemplo a continuación hemos fijamos la página web de App Inventor escribiendo directamente su dirección en un bloque de texto. Tmbién podríamos incluir una caja de texto en la app a modo de barra de direcciones, y usar lo que escriba el usuario como parámetro del bloque "call (WebViewer1).GoToUrl".



A continuación vams a explrar el componente "ActivityStarter", que es más flexible.

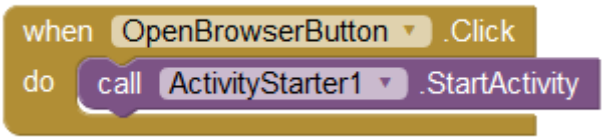
USAR EL COMPONENTE "ACTIVITYSTARTER".

Para probar el componente "ActivityStarter" vamos a construir una app llamada "TestActivityStarter". Seguimos estos pasos para construirla:

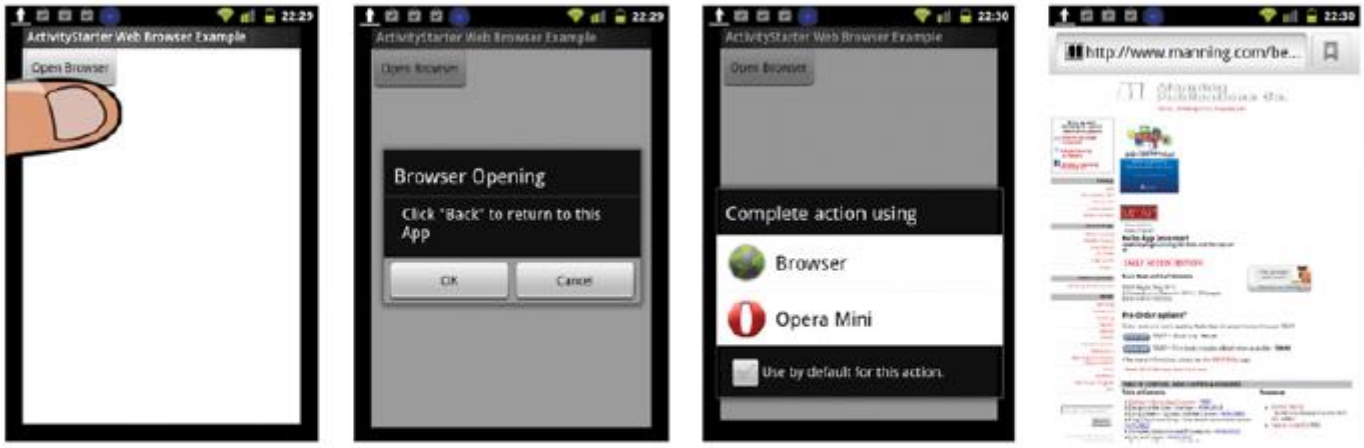
- Fijamos la propiedad "Screen1.Title" a "ActivityStarter Web Browser Example".
- Añadimos un botón (al que llamaremos "OpenBrowserButton"), un componente "ActivityStarter" (de la paleta "Connectivity"), y un notificador.
- Fijamos las propiedades del componente "ActivityStarter" como indicamos:
  - "Action": fijamos esta propiedad a "Android.intent.action.VIEW". Esto sirve para decir que queremos mostrar algunos datos al usuario.

- "DataUri": Aquí escribimos la dirección de la página web a la que queremos acceder, a saber, https://www.elpais.com/tecnologia/. Como el Identificador de Recurso Único (URI) es la dirección de una página web, Android nos permitirá elegir una app de enytre aquellas que nos permitan mostrar páginas web.

Ahora, para activar el componente "ActivityStarter", podríamos usar el bloque "call (ActivityStarter1).StartActivity". Para activar esta acción, podríamos emplear con el manejador de eventos "when (OpenBrowserButton).Click", como muestra la figura:



Sin embargo, en este caso vamos a hacer que el usuario abandone la pantalla de la app para ir a su navegador preferido, por lo que hemos añadido un componente "Notifier" para indicarle cómo volevr a la app cuando hayan terminado de navegar. La figura ilustra cómo funciona la secuencia:



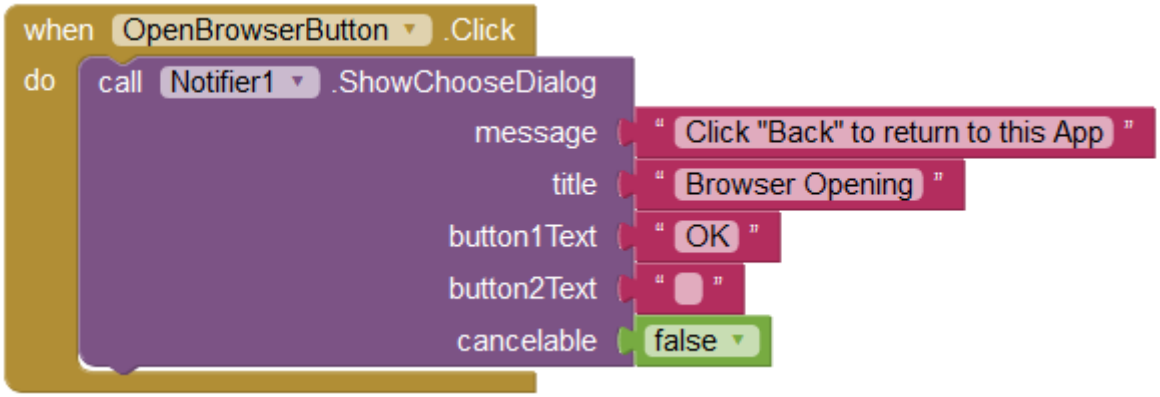
1. The user clicks Open Browser, triggering ...

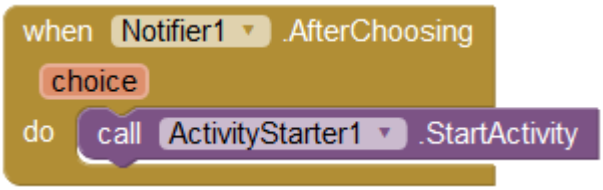
2. ... the `Notifier`, which checks that the user wants to open the browser and tells them how to get back to this app.

3. Clicking OK presents a list of apps that can open web pages if you have more than one browser installed.

4. The browser opens. Clicking the phone's Back button returns the user to the app (step 1).

De la secuencia ilustrada vemos que el evento "when (OpenBrowserButton).Click" abre el notificador, llamando a la función "(Notifier1).ShowChooseDialog". Cuando el usuario pulse en el botón de "OK" del notificador, el notificador lanza un evento "(Notifier1).AfterChoosing", que usaremos para lanzar el componente "ActivityStarter", mediante el bloque "(ActivityStarter1).StartActivity":





Esta app debería ser capaz lanzar nuestro navegador preferido. La probamos para cerciorarnos que ése es el caso. El componente "ActivityStarter" es versátil, y también puede lanzar otras apps como la de la cámara, Google Maps, e incluso otras apps que nosotros hayamos escrito. Además, si le proporcionamos la URL a un recurso específico relacionado con una app, Android presentará más opciones en la lista de la app. Por ejemplo, si tenemos una app de YouTube instalada en nuestro dispositivo, y usamos una URL de YouTube, "ActivityStarter" nos presentará la opción de ver el video en el navegador o en la app de YouTube. La documentación que explica cómo hacer esto en App Inventor está accesible en el enlace http://appinventor.mit.edu/explore/ai2/activity-starter.html.

8.13. COMENZAR CON LA APP "ANDROIDWHERE"

En esta sección vamos a usar el componente "WebViewer" para implementar una app en la que necesitaremos integrar en la pantalla la página web de Google Maps.

Imaginemos la situación en la que vamos de visita con nuestro coche a una ciudad que no conocemos. Aparcamos el coche, y tras hacer la correspondiente visita turística, no recordamos dónde lo estacionamos. Una app interesante sería aquella en la que, tras aparcar el coche, pulsamos un botón para que el móvil detecte la localización actual y guarde las coordenadas GPS de la ubicación del coche. Después, cuando volvemos a abrir la app, el móvil nos guía desde nuestra nueva posición a la ubicación donde dejamos el coche. Ésta es precisamente la app que construiremos en las siguientes secciones.

Nos conectamos a la web de App Inventor y comenzamos un nuevo proyecto al que llamaremos "AndroidWhere". Fijamos el título de la pantalla a "Android, Where is My Car?". Conectamos nuestro dispositivo o el emulador para realizar pruebas en vivo.

8.14. DISEÑAR LOS COMPONENTES DE "ANDROIDWHERE".

La interfaz de usuario para la app "AndroidWhere" consta de varias etiquetas para mostrar la posición grabada previamente y nuestra posición actual, y de algunos botones para registrar una localización y para mostrar la ruta hacia ella (ver figura).

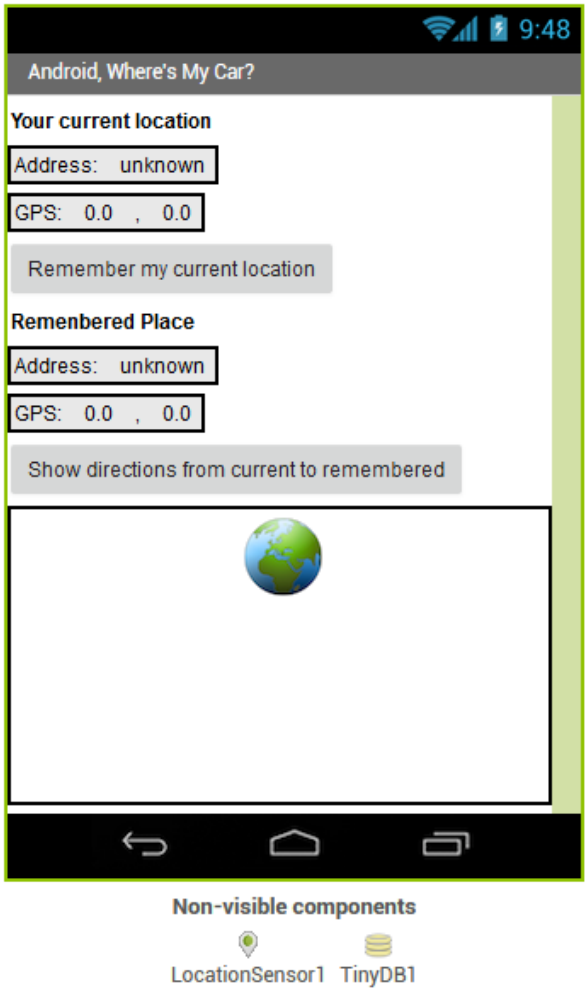
En esta app necesitaremos algunas etiquetas que simplemente muestren un texto estático, y otras que mostrarán los datos del sensor de localización. Para estas últimas proporcionaremos un valor por defecto de $(0,0)$, que cambiará conforme el GPS comience a adquirir información de posición. También necesitaremos tres componentes no visibles: Un componente "LocationSensor" para obtener la posición actual, un componente "TinyDB" para guardar localizaciones de forma persistente, y un componente "WebViewer" para mostrar (integrada en la pantalla de la app) la ruta en Google Maps desde la posición actual a la posición previamente guardada. La tabla a continuación lista el conjunto de componentes que deberemos añadir a nuestra app de forma que adquiera la apariencia mostrada en la figura.

Component type	Palette group	What you'll name it	Purpose
Label	User Interface	CurrentHeaderLabel	Display the header "Your current location".
HorizontalArrangement	Layout	HorizontalArrangement1	Arrange the address info.
Label	User Interface	CurrentAddressLabel	Display the text "Address:".
Label	User Interface	CurrentAddressDataLabel	Display dynamic data: the current address.
HorizontalArrangement	Layout	HorizontalArrangement2	Arrange the GPS info.
Label	User Interface	GPSLabel	Display the text "GPS:".
Label	User Interface	CurrentLatLabel	Display dynamic data: the current latitude.
Label	User Interface	CommaLabel	Display ",".
Label	User Interface	CurrentLongLabel	Display dynamic data: the current longitude.
Button	User Interface	RememberButton	Click to record the current location.
Label	User Interface	RememberedHeaderLabel	Display the text "Remembered Place".
HorizontalArrangement	Layout	HorizontalArrangement3	Arrange remembered address info.
Label	User Interface	RememberedAddressLabel	Display the text "Remembered Place".
Label	User Interface	RememberedAddressDataLabel	Display dynamic data: the remembered address.
HorizontalArrangement	Layout	HorizontalArrangement4	Arrange remembered GPS info.
Label	User Interface	RememberedGPSLabel	Display the text "GPS".
Label	User Interface	RememberedLatLabel	Display dynamic data: the remembered latitude.
Label	User Interface	Comma2Label	Display ",".
Label	User Interface	RememberedLongLabel	Display dynamic data: the remembered longitude.
Button	User Interface	DirectionsButton	Click to show the map.

Component type	Palette group	What you'll name it	Purpose
LocationSensor	Sensors	LocationSensor1	Sense GPS info.
TinyDB	Storage	TinyDB1	Store the remembered location persistently.
WebViewer	User Interface	WebViewer1	Show directions.

Ajustamos las propiedades de los componentes listados como se indica a continuación:

- Fijamos la propiedad "Text" de las etiquetas con el texto especificado en la tabla.
- Fijamos la propiedad "Text" de las etiquetas para los datos GPS dinámicos a "0.0".
- Fijamos la propiedad "Text" de las etiquetas para las direcciones dinámicas a "unknown".
- Desactivamos la propiedad "Enabled" de los botones "RememberButton" y "DirectionsButton".
- Desactivamos la propiedad "Scrollable" de "Screen1" para que el componente "WebViewer" se ajuste al tamaño de la pantalla.



8.15. AÑADIR COMPORTAMIENTOS A LOS COMPONENTES DE "ANDROIDWHERE".

Para esta app necesitaremos implementar los siguientes comportamientos:

- Cuando el componente "LocationSensor" obtenga una lectura, mostramos los datos de la localización actual en las etiquetas apropiadas en la interfaz de usuario. Esto le permitirá al usuario saber que el sensor ha obtenido una localización, y que está listo para recordarla.

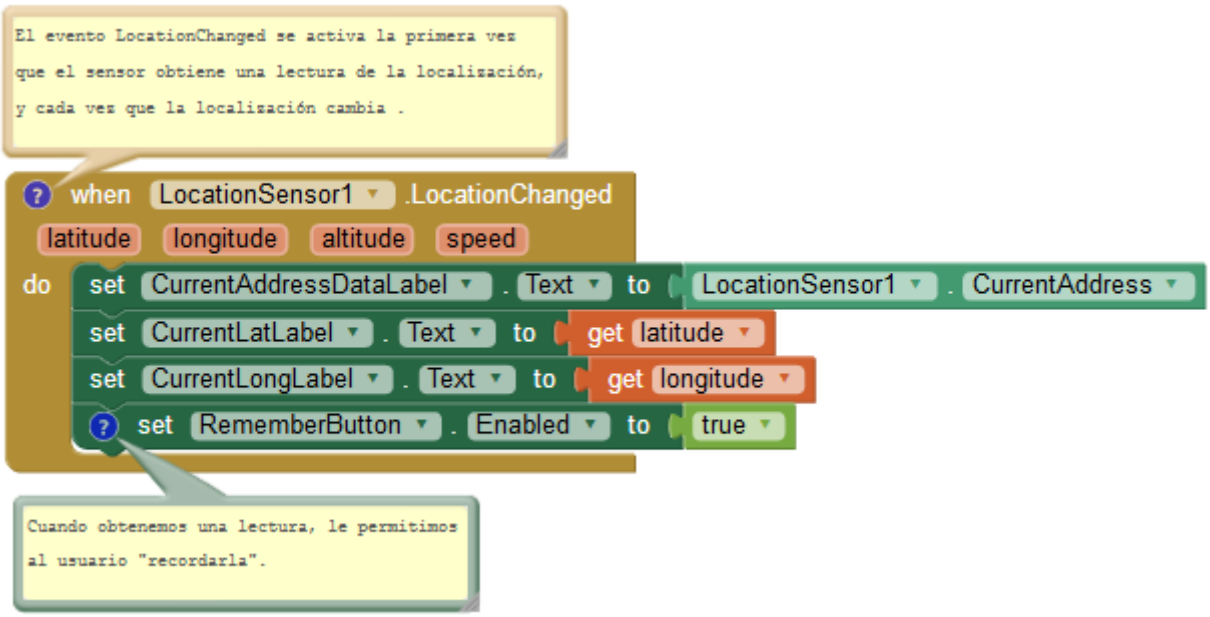
- Cuando el usuario presione el botón "RememberButton", copiamos los datos de la localización actual en las etiquetas de la localización recordada. También tendremos que guardar la información recordada para que esté disponible incluso si el usuario cierra y vuelve a abrir la app.
- Cuando el usuario presione el botón "DirectionsButton", lanzamos la aplicación Google Maps en el visor web, para mostrar la ruta a la localización recordada.
- Cuando volvemos a arrancar la app, cargamos la localización recordada desde la base de datos.

MOSTRAR LA LOCALIZACIÓN ACTUAL.

El evento "when (LocationSensor).LocationChanged" se activa cuando cambia la localización del dispositivo, y también cuando el sensor obtiene su primera lectura. En ocasiones, el sensor tarda unos segundos en adquirir esa primera lectura, y otras veces ni siquiera es capaz de obtener una lectura si la señal GPS está bloqueada. Cuando obtenga una lectura de la localización, la app debería ubicar esos datos en las etiquetas apropiadas. La tabla a continuación lista los bloques que necesitaremos usar para implementar este comportamiento:

Block type	Drawer	Purpose
`LocationSensor1.LocationChanged`	LocationSensor1	This is the event handler that is triggered when the phone receives a new GPS reading.
`set CurrentAddressDataLabel.Text to`	CurrentAddressDataLabel	Place the new data into the label for the current address.
`LocationSensor1.CurrentAddress`	LocationSensor1	This property gives you a street address.
`set CurrentLatLabel.Text to`	CurrentLatLabel	Place the latitude into the appropriate label.
`get latitude`	Drag out from LocationChanged event	Plug into `set CurrentLatLabel.Text to`.
`set CurrentLongLabel.Text to`	CurrentLongLabel	Place the longitude into the appropriate label.
`value longitude`	Drag out from LocationChanged event	Plug into `set CurrentLongLabel.Text to`.
`set RememberButton.Enabled to`	RememberButton	Remember the reading for current location.
`true`	Logic	Plug into `set RememberButton.Enabled to`.

La figura muestra el programa necesario para hacerlo:



¿Cómo funciona este código? La figura muestra que "latitude" y "longitude" son parámetros del manejador de evento "when (LocationSensor).LocationChanged". Podemos tomar sendos bloques "get" para acceder a los valores de estos dos parámetros pasando el ratón por encima de ellos. "LocationSensor.CurrentAddress" no es un argumento, sino una propiedad del componente "LocationSensor", y podemos acceder a ella en la bandeja de "LocationSensor". ¿De dónde sale esta información? El sensor de localización hace una llamada a Google Maps para obtener la dirección asociada a la ubicación GPS correspondiente, y posteriormente la almacena en esta propiedad.

El manejador "when (LocationSensor).LocationChanged" también activa el botón "RememberButton" (poniendo a "true" su propiedad "enabled"). Recordemos que, al crearlo en el Diseñador, inicializamos este botón desactivado, porque nada más arrancar la app el usuario no tiene información de localización que pueda recordar, hasta que el sensor obtenga una primera lectura. En el manejador hemos programado este comportamiento.

Vamos a probar la app. Probablemente querremos movernos para comprobar cómo se actualiza la información de localización, y para ello, tendremos que empaquetar la app e instalarla en nuestro teléfono. Esto se hace en App Inventor seleccionando "Build" → "App (provide QR code for .apk)". Al ejecutar la app, deberíamos ver que hay datos GPS apareciendo en las etiquetas correspondientes, y que el botón "RememberButton" se activa nada más obtener la primera lectura. Si no obtenemos una primera lectura en unos pocos segundos, debemos comprobar los ajustes de Localización y Seguridad de nuestro teléfono, y/o salir al exterior para asegurarnos de tener señal GPS.

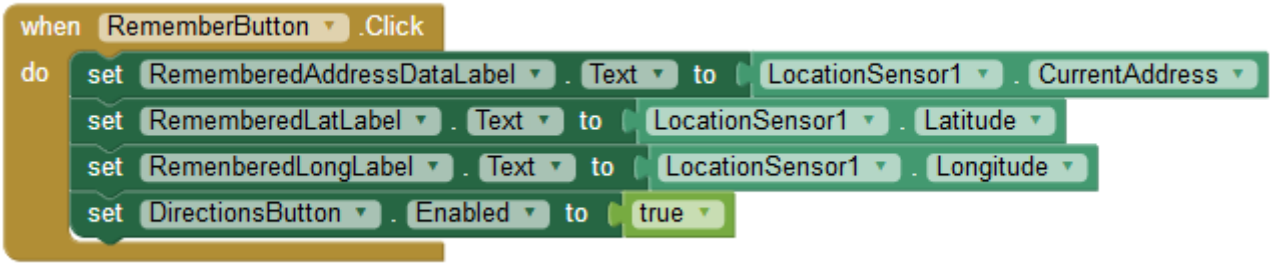
REGISTRAR LA LOCALIZACIÓN ACTUAL.

Cuando el usuario presione el botón "RememberButton", los datos de localización más recientes deben ubicarse en las etiquetas que muestran los datos recordados.

La tabla muestra los bloques que necesitamos para implementar esta funcionalidad:

Block type	Drawer	Purpose
`RememberButton.Click`	RememberButton	Triggered when the user clicks "Remember."
`set RememberedAddressDataLabel.Text to`	RememberedAddressDataLabel	Place the sensor's address data into the label for the remembered address.
`LocationSensor1.CurrentAddress`	LocationSensor1	This property gives you a street address.
`set RememberedLatLabel.Text to`	RememberedLatLabel	Place the latitude sensed into the "remembered" label.
`LocationSensor.Latitude`	LocationSensor1	Plug into `set RememberedLatLabel.Text to`.
`set RememberedLongLabel.Text to`	RememberedLongLabel	Place the longitude sensed into the "remembered" label.
`LocationSensor.Longitude`	LocationSensor1	Plug into `set RememberedLongLabel.Text to`.
`set DirectionsButton.Enabled to`	DirectionsButton	Map the remembered place.
true	Logic	Plug into `set DirectionsButton.Enabled to`.

El programa que debemos escribir es el siguiente:



¿Cómo funciona este código? Cuando el usuario pulsa el botón "RememberButton", las lecturas actuales del sensor de localización se insertan en las etiquetas "Remembered", para que esa información quede registrada.

Este código también activa el botón "DirectionsButton". Pero cuidado, esto podría darnos problemas si el usuario presiona el botón "DirectionsButton" de forma inmediata, porque la localización recordada será la misma que la localización actual, y el mapa que aparecería mostraría una ruta desde la posición en la que estamos a la misma posición en la que seguimos estando. Pero no es probable que ningún usuario haga esto; después de que el usuario se haya movido, la localización actual y la localización recordada ya no serán la misma.

Probamos nuestra app: Descargamos la nueva versión empaquetada de la app a nuestro teléfono móvil, y nos cercioramos de que, al presionar el botón "RememberButton", los datos de localización de la posición actual se copian en las etiquetas para la localización recordada.

MOSTRAR LA RUTA DESDE LA POSICIÓN ACTUAL A LA POSICIÓN RECORDADA.

Cuando el usuario pincha en el botón "DirectionsButton", la app debería abrir Google Maps y mostrar la ruta desde la localización actual del usuario a la localización recordada (por ejemplo, el sitio donde aparcó su coche).

El componente "WebViewer" nos permite mostrar (integrada en nuestra app) cualquier página web, incluyendo Google Maps. Eso podemos hacerlo llamando al procedimiento "WebViewer.GoToURL" para abrir un mapa, pero la URL del mapa debe incluir la ruta desde la localización actual a la localización recordada.

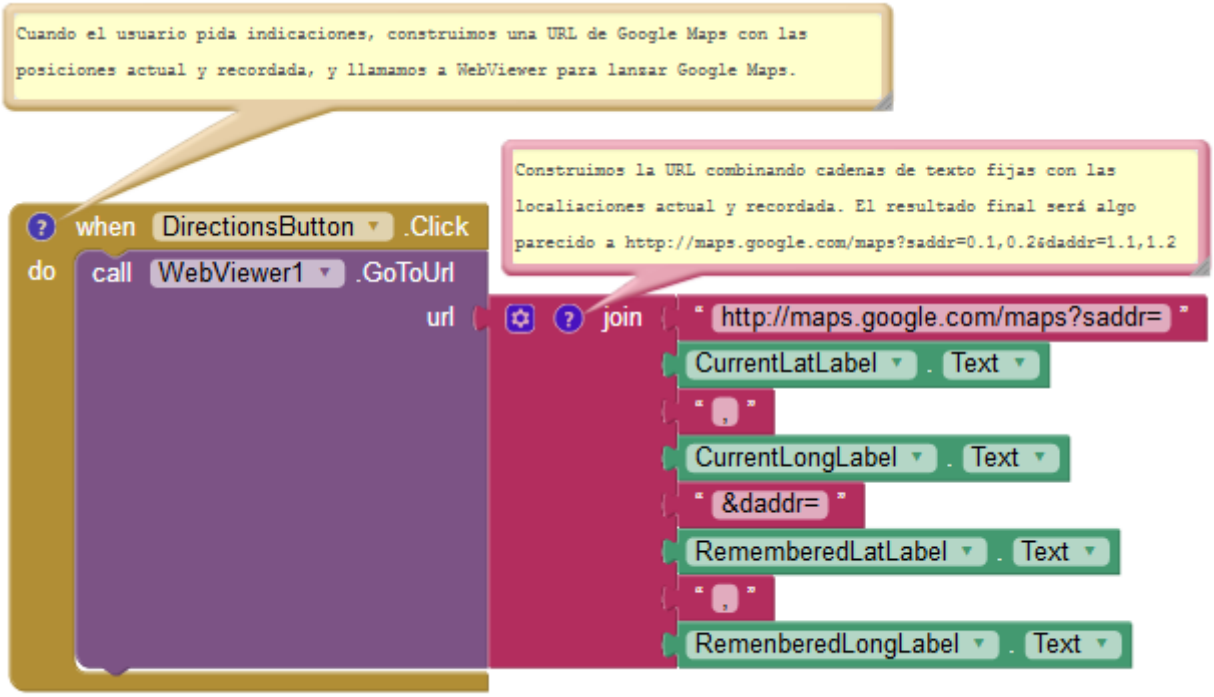
Una forma de mostrar una ruta entre dos localizaciones en Google Maps es hacerlo con una URL como la siguiente: http://maps.google.com/maps?saddr=37.82557,-122.47898&daddr=37.81079,-122.47710. Escribimos esta URL en la barra de direcciones de un navegador para comprobar que funciona correctamente.

En nuestra app necesitamos construir la URL y fijar dinámicamente su dirección origen (parámetro "saddr", o *source address*) y su dirección destino (parámetro "daddr", o *destination address*) mediante bloques. En capítulos previos ya hemos concatenado varios trozos de texto usando la función "join". Aquí haremos lo mismo para construir la URL del mapa de la ruta, combinando los datos de las localizaciones actual y recordada. Después conectaremos esta URL a la ranura para parámetros de la función "WebViewer.GoToURL". La tabla lista los bloques necesarios para hacer todo esto:

Block type	Drawer	Purpose
`DirectionsButton.Click`	DirectionsButton	Triggered when the user clicks "Directions."
`WebViewer1.GoToURL`	WebViewer1	Set the URL for the map that you want to bring up.
`join`	Text	Build a URL from multiple parts.
`text` ("http://maps.google.com/maps?saddr=")	Text	The fixed part of the URL, the source address.
`CurrentLatLabel.Text`	CurrentLatLabel	The current latitude.
`text` (",")	Text	Put a comma between the latitude and longitude values.
`CurrentLongLabel.Text`	CurrentLongLabel	The current longitude.
`text` ("&daddr=")	Text	The second parameter of the URL, the destination address.
`RememberedLatLabel.Text`	RememberedLatLabel	The remembered latitude.
`text` (",")	Text	Put a comma between the values for latitude and longitude.
`RememberedLongLabel.Text`	RememberedLongLabel	The remembered longitude.

La figura muestra el programa que implementa esta funcionalidad. Cuando el usuario pincha en el botón "DirectionsButton", el manejador de eventos asociado a esta acción construye una URL para un mapa de Google Maps y llama a la función "WebViewer.GoToURL" para abrir el mapa, como muestra la figura. Observar cómo usamos la función "join" para construir la URL combinando partes de texto fijas con los datos de localización de las posiciones actual y recordada. La URL resultante está formada por el dominio de Google Maps (http://maps.google.com/maps), junto con dos parámetros de URL, a saber, "saddr" y

"daddr", que especifican las localizaciones de origen y destino de la ruta que mostrará el mapa. Para nuestra app, el parámetro "saddr" se fija a la latitud y la longitud de la localización actual, y el parámetro "daddr" a la latitud y longitud de la localización guardada.



Vamos a probar la app: Descargamos la nueva versión de la app a nuestro teléfono. Cuando el sensor GPS registre una lectura pinchamos en el botón "RememberButton" para guardarla, y nos vamos a dar una vuelta. Cuando presionemos el botón "DirectionsButton", el visor Web debería mostrar un mapa de Google Maps con la ruta que nos lleva de vuelta a la localización recordada. Después de ello, clicamos en el botón "hacia atrás" unas cuantas veces, y comprobamos que el teléfono vuelve a nuestra app.

ALMACENAR PERSISTENTEMENTE LA LOCALIZACIÓN RECORDADA.

Llegados a este punto tenemos ya una app plenamente operativa que recuerda una localización inicial y dibuja una ruta para retornar a esa ubicación desde la posición actual del usuario, sea cual sea. Sin embargo, si el usuario guarda una localización y cierra la app, los datos almacenados se pierden y no estarán disponibles al volver a arrancar la app. Aquí necesitamos que el usuario pueda guardar la localización de su coche, cerrar la app, moverse por la ciudad o acudir a un cierto evento, y volver a arrancar la app para obtener la ruta que le lleva de vuelta a la localización guardada.

Para ello, necesitamos guardar los datos de forma persistente en una base de datos "TinyDB", como ya hicimos en capítulos previos. El esquema que usaremos aquí es similar al que usamos en la app "NoTextingWhileDriving":

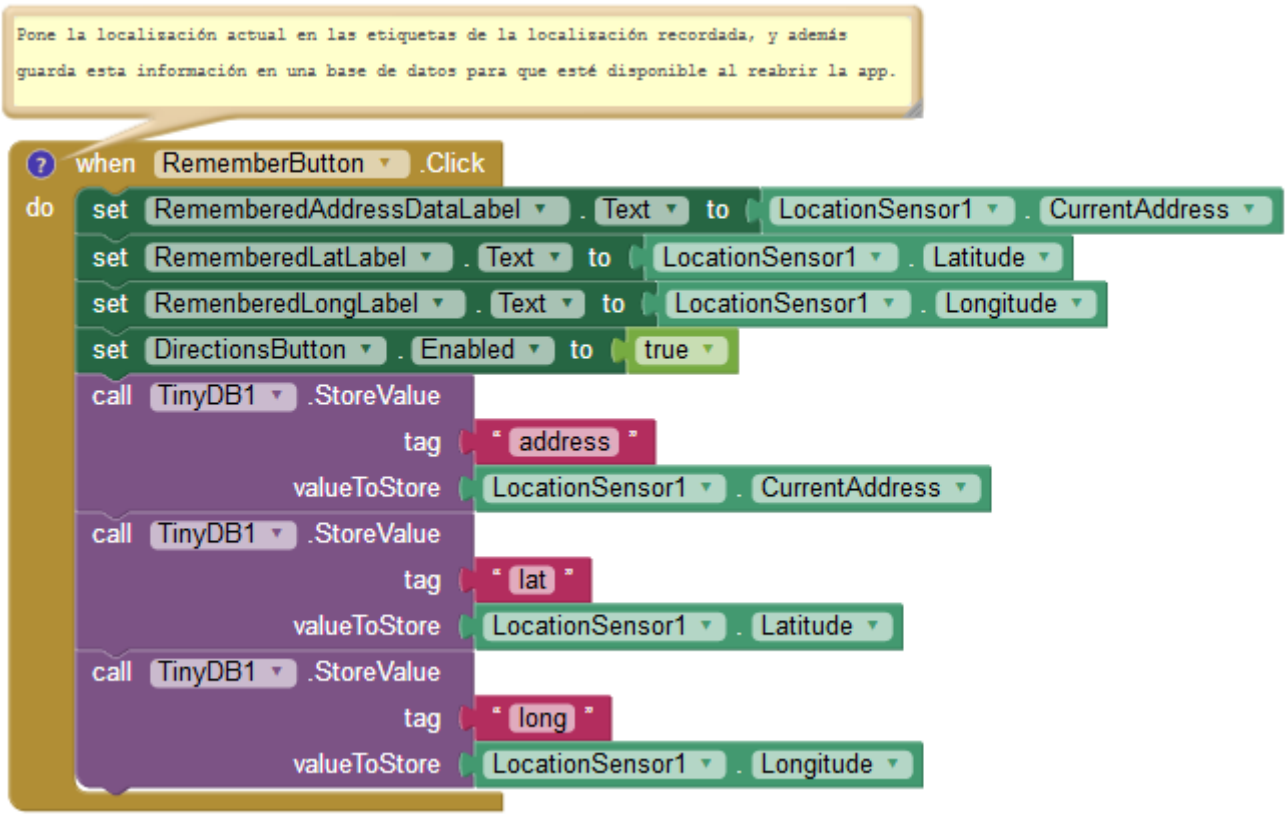
1) Cuando el usuario presione el botón "RememberButton", almacenaremos la información de localización en la base de datos.
2) Al volver a abrir la app, cargaremos los datos de localización previamente guerdados en esa base de datos.

Vamos a comenzar modificando el manejador de eventos "when (RememberButton).Click" para que almacene la información de la localización recordada en la base de datos. Para guardar la latitud, longitud, y dirección de la localización recordada deberemos hacer tres llamadas a la función "TinyDB.StoreValue".

La tabla lista todos los bloques necesarios para hacerlo.

Block type	Drawer	Purpose
`TinyDB1.StoreValue` (3)	TinyDB	Store the data in the device database.
`text` ("address")	Text	Plug this into the "tag" socket of `TinyDB1.StoreValue`.
`LocationSensor1.CurrentAddress`	LocationSensor1	The address to store persistently; plug this into the "value" socket of `TinyDB1.StoreValue`.
`text` ("lat")	Text	Plug this into the "tag" socket of the second `TinyDB1.StoreValue`.
`LocationSensor1.CurrentLatitude`	LocationSensor1	The latitude to store persistently; plug this into the "value" socket of the second `TinyDB1.StoreValue`.
`text` ("long")	Text	Plug this into the "tag" socket of the third `TinyDB1.StoreValue`.
`LocationSensor1.CurrentLongitude`	LocationSensor1	The longitude to store persistently; plug this into the "value" socket of the third `TinyDB1.StoreValue`.

La figura muestra el código para programar este comportamiento. Las llamadas a la función "(TinyDB1).StoreValue" copian los datos de localización presentes en las propiedades de "LocationSensor" a la base de datos. Como recordaremos, la función "(TinyDB).StoreValue" necesita que le especifiquemos los valores de los parámetros "tag" y "value" para poder funcionar. El parámetro "tag" sirve para identificar los datos que queremos almacenar, y el parámetro "value" representa los datos que vamos a guardar (en este caso, la información de localización registrada por "LocationSensor").



RECUPERAR LA LOCALIZACIÓN GUARDADA AL REABRIR LA APP.

Si guardamos información en una base de datos es para poder recuperarla después. En esta app, si el usuario guarda la información de localización de una cierta ubicación, queremos que la app recupere esa información de la base de datos y la muestre donde corresponda al volver a lanzar la app.

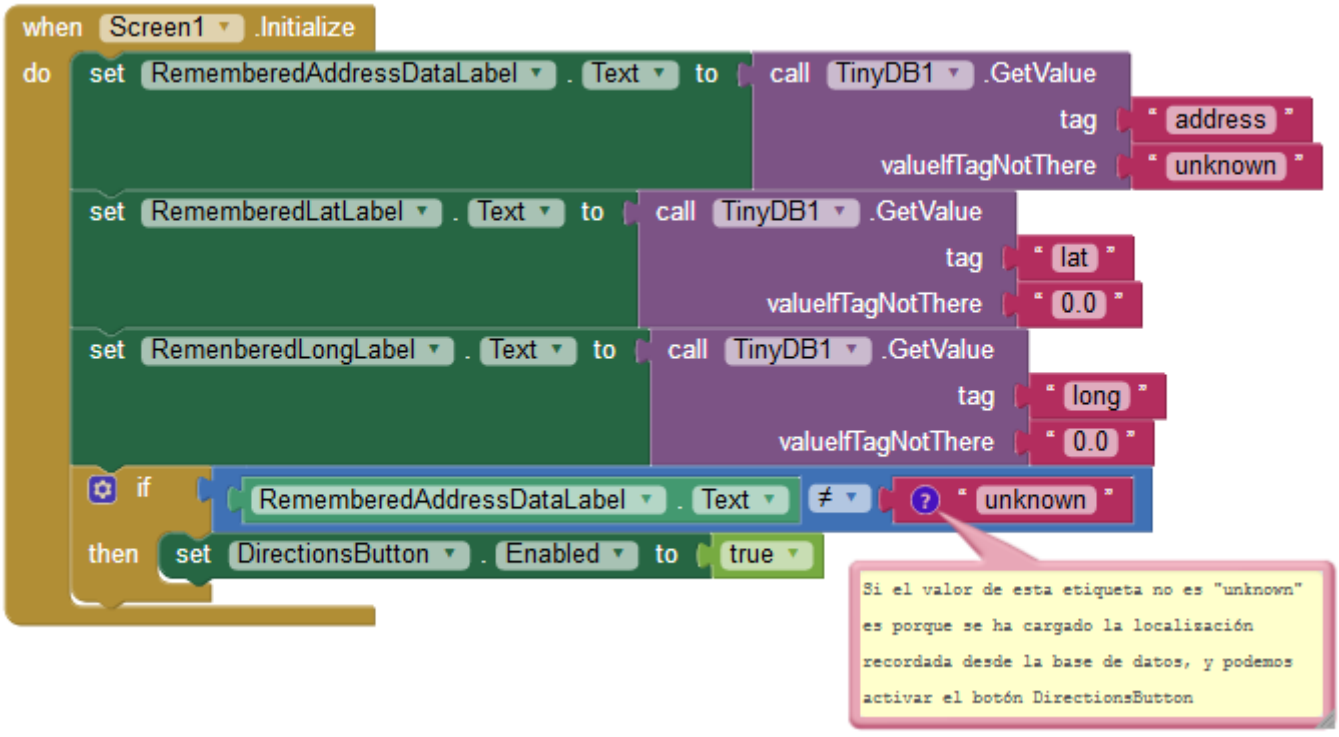
Como ya hemos indicado en capítulos previos, el evento "(Screen).Initialize" se activa siempre que nuestra app se carga. Cuando se abre una app, una de las tareas más habituales es recuperar información de una base de datos, y eso es precisamente lo que queremos hacer aquí.

Vamos a usar la función "TinyDB.GetValue" para recuperar los datos GPS de la localización recordada. Como debemos recuperar la dirección, latitud, y longitud recordadas, deberemos hacer tres llamadas a esta función. Como hicimos en la app "NoTextingWhileDriving", primero necesitaremos comprobar si hay datos disponibles. (En efecto, si es la primera vez que usamos la app, "TinyDB.GetValue" devolverá un texto vacío). A modo de ejercicio, intenta escribir el programa necesario para programar esta funcionalidad, y compara tu código con el mostrado en la figura.

Para entender el programa de la figura, consideremos dos casos de uso: (1) Un usuario abriendo la app por primera vez, y (2) un usuario abriendo la app más tarde cuando ya ha guardado datos de localización. La primera vez que el usuario abre la app no habrá información que cargar desde la base de datos. Las siguientes veces que el usuario arranque la app, y si hay datos guardados, querremos que la app cargue esa información desde la base de datos.

El programa llama a la función "TinyDB1.GetValue" tres veces, una por cada uno de los datos que guardamos previamente, a saber, "address", "lat", y "long". El parámetro "valueIfTagNotThere" se fija al valor por defecto que debería recuperar cada una de estas llamadas si no encuentran datos en la base de datos.

El bloque "if" se usa para determinar si deberíamos activar el botón "DirectionsButton". Este botón debería habilitarse si se recuperan datos de la base de datos. La comprobación que hacemos es mirar si el valor de la etiqueta "RememberedAddressDataLabel" se ha fijado al valor por defecto, "unknown". Si no es "unknown" es porque se ha cargado una dirección, lo que significa que se ha recuperado información desde la base de datos, y en ese caso, activamos el botón.



Vamos a probar la app. Empaquetamos y descargamos la nueva versión de la app en nuestro teléfono. Clicamos en "RemenberButton" y nos aseguramos que se guarda la información de localización. A continuación cerramos la app y volvemos a abrirla. En las etiquetas de los datos recordados deberían aparecer los datos de localización previamente guardados, y el botón "DirectionsButton" debería parecer habilitado.

8.16. MODIFICACIONES A "ANDROIDWHERE".

- Podemos intentar crear una app llamada "Android, Where Is Everyone?", que le permita a un grupo de personas rastrear el paradero de cada una de ellas. Esta app sería muy útil al hacer senderismo por el campo, o para controlar a un grupo de niños en una excursión. Los datos de esta app son compartidos, y necesitaremos usar una base de datos web y el componente "TinyWebDB" en lugar del componente "TinyDB".
- Crea una app llamada "MigasDePan" que controle nuestros movimientos guardando en una lista cada cambio en nuestra localización. Solo deberíamos registrar una nueva localización si nuestra posición ha cambiado en una cierta cantidad, o si ha pasado una cierta cantidad de tiempo desde que guardamos la última localización.

8.17. EJERCICIOS DEL CAPÍTULO 8.

Ejercicio 8.1. Además del sensor de orientación, los teléfonos también incluyen un sensor ecelerómetro. Escribe una app de pruebas muestre las lecturas ofrecidas por estos dos sensores, para observar cómo cambian conforme movemos el teléfono. Un ejemplo sería una app como la de la figura:

PISTA 1: Como quieres monitorizar constantemente el cambio en las lecturas de estos dos sensores, probablemente necesitarás emplear un temporizador que se dispare de forma muy frecuente.

PISTA 2: Se recomienda usar un organizador para mostrar adecuadamente todos los datos que deseamos mostrar.

Ejercicio 8.2. Lost and Found.
Escribe una app que nos permita saber exactamente dónde estamos exactamente (latitud, longitud, y dirección) al pulsar un botón. La interfaz de usuario debería parecerse a la siguiente:

La imagen del mapa la encontraremos en los archivos de los alumnos, con el nombre *foldedmap.png*. La tabla lista los componentes que necesitaremos para construir la app, qué nombre les daremos, cuál es su función dentro de la app, y qué valores debemos ajustar a sus propiedades:

Lost & Found			
Screen1 properties	`AlignHorizontal`: Center `AlignVertical`: Center `ScreenOrientation`: Portrait `Scrollable`: Yes (selected) `Title`: Lost & Found `BackgroundColor`: Orange		
Components	**What do I rename it?**	**What does it do?**	**What properties do I set?**
`Vertical-Arrangement`	`Vertical-Arrangement1`	Allows you to position the image and labels down the page	
`Button`	`Locationbutton`	When clicked, the current location is recalled	`FontSize`: 20 `Text`: "Click twice to find out where you are"
`Image`	`MapImage`	Displays a map image	`Picture`: foldedmap.png

6 `Labels`	`Addresslabel`
	`Addressdatalabel`
	`Latitudelabel`
	`Latitudedatalabel`
	`Longitudelabel`
	`Longitudedatalabel`
		Displays `Text`
		Displays the address
		Displays `Text`
		Displays the latitude
		Displays `Text`
		Displays the Longitude
			`Text`: "Address"
			`Text`: "Address will appear here"
			`Text`: "Latitude"
			`Text`: "Latitude will appear here"
			`Text`: "Longitude"
			`Text`: "Longitude will appear here"

`LocationSensor` Palette group: Sensors	`Locationsensor1`	Identifies the current location of the phone	

Ejercicio 8.3. Usa los sensores del teléfono para crear una aplicación antirrobo que funcione como se indica:

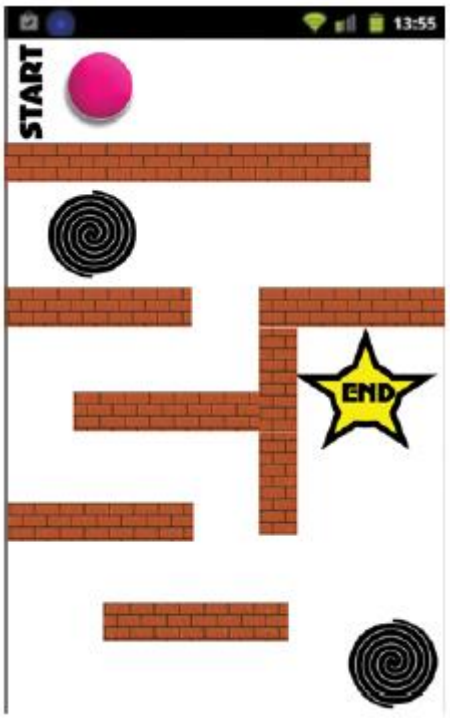
a) El usuario escribe una contraseña y pulsa en un botón "ActivateButton" para activar la alarma, tras lo cual dejan el móvil.
b) Al pasar 10 segundos, el teléfono activa su sensor de orientación.
c) Si ocurre un evento "OrientationChanged", el teléfono nos da 10 segundos para escribir la contraseña y pulsar un botón "DeactivateButton" (Desactivación).
d) Si no se inserta la contraseña correcta, el teléfono hace sonar una alarma hasta que se escriba la contraseña correcta.

AMPLIACIÓN: Haz que el teléfono envíe un mensaje del tipo "¡Ayuda, me están robando el teléfono!" a otro teléfono móvil. El mensaje enviado debe incluir la información sobre la localización actual del teléfono robado.

Ejercicio 8.4. Laberinto.

Crea un juego del laberinto como el mostrado en la figura. El usuario conduce una pelota desde la posición START hasta la posición END, a través de un laberinto con múltipels muros y agujeros. La pelota, los muros, y los agujeros son sprites independientes. Las reglas del juego son las siguientes:

- La pelota siempre empieza en una misma posición (x, y) específica. (Guarda esta posición en una o varias variables).
- El usuario inclina el dispositivo para mover la pelota.
- Si la pelota colisiona contra un muro, rebota (o si queremos hacer el juego más difícil, la pelota retorna a la posición START).
- Si la pelota se cuela en (colisiona contra) un agujero, hemos perdido y la partida termina. (El juego muestra un mensaje indicando que hemos perdido).
- Si la pelota llega a la posición END (esto es, si colisiona contra la estrella END), el juego muestra un mensaje de felicitación.
- La app debería incluir un botón de Nueva Partida para hacer volver al juego a la situación original.



AMPLIACIÓN: Modifica el juego para que lleve la cuenta de la puntuación, y para poder jugar en múltiples niveles.

Ejercicio 8.5. Crea una webquest que le lleve al usuario de tour a través de varias páginas web sobre un cierto tema de tu interés.

Ejercicio 8.6. Crea una app de cuestionario que formule al usuario múltiples preguntas, cuya respuesta requiera la visita de las webs indicadas como pista tras cada pregunta.

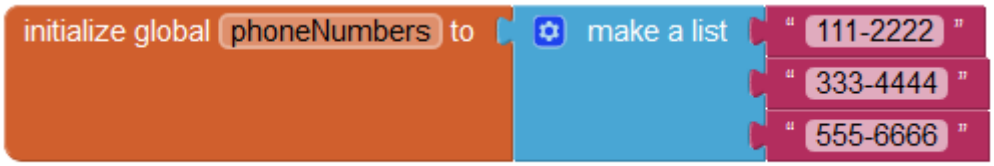
Ejercicio 8.7. Crea una aplicación con múltiples botones para abrir YouTube, Facebook, Twitter, etc. Al pulsar en cada botón, un componente "ActivityStarter" abre en el navegador favorito del usuario la aplicación seleccionada.

9. LISTAS.

9.1. VARIABLES TIPO LISTA.

Como ya sabemos, las apps responden ante eventos y toman decisiones. Este tipo de procesado es fundamental en programación. Sin embargo, otra parte fundamental en las apps son los datos, esto es, la información que la app procesa. Normalmente, los datos que maneja una app no se limitan a ser posiciones de memoria individuales, como la puntuación de un videojuego. De hecho, lo más habitual es que se traten de **listas** de datos interrelacionados. Muchas apps necesitan trabajar con listas de datos: Por ejemplo, Facebook procesa nuestra lista de amigos, y nos informa ante posibles cambios en su estado. En este capítulo construiremos una app de un cuestionario con sendas listas de preguntas y respuestas.

Para especificar una lista en App Inventor, primero debemos crear una variable que la aloje. Pero en este caso, en vez de reservar una sola posición de memoria para almacenar esa variable, App Inventor reservará un conjunto de posiciones de memoria para guardar los distintos elementos de la lista. Para indicar que una variable es de tipo lista, usaremos el bloque "make a list" o el bloque "create empty list". Por ejemplo, la variable "phoneNumbers" de la figura define una lista de tres números de teléfono (definidos como bloques de texto):

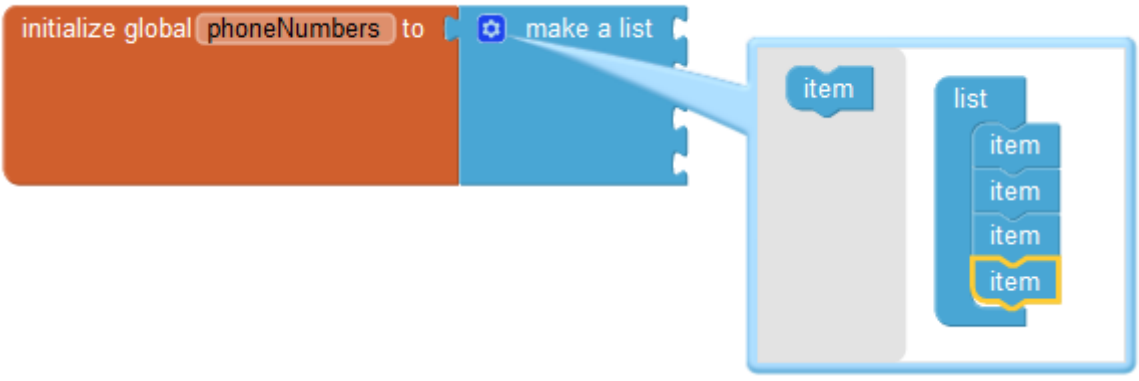


En este capítulo aprenderemos la forma en la que App Inventor maneja las listas de datos. Aquí presentaremos los fundamentos de la programación con listas de datos estáticos, en las que los objetos de la lista no cambian, y de la programación con listas de datos dinámicos, en las que el usuario puede agregar, cambiar, o quitar objetos de la lista. De esta forma, aprenderemos a trabajar con listas y exploraremos estructuras de datos más complejas, como las listas de listas.

9.2. TRABAJAR CON LISTAS.

CREAR UNA VARIABLE DE TIPO LISTA.

Para crear una variable de tipo lista en el Editor de Bloques usamos un bloque "initialize global variable", y le conectamos el bloque "make a list". El bloque "make a list" lo encontraremos en la bandeja "Lists", y solo dispone de dos conectores para incluir dos elementos en la lista. Pero podemos especificar el número de coenctores (el número de elementos) que necesitamos en la lista pinchando en el icono azul del engranaje, y añadiendo bloques "item" a la ranura abierta del bloque "list", como vemos en la figura:



Podemos conectar cualquier tipo de datos a los puertos del bloque "make a list". En el ejemplo de la lista de números de teléfono, los objetos de la lista deben ser textos, y no números, porque los números de

teléfono tienen guiones, paréntesis, y otros símbolos que no podemos poner en un objeto de tipo número, y porque tampoco vamos a relaizar ningún tipo de operación con esos números.

SELECCIONAR UN OBJETO DE UNA LISTA.

Cunado nuestra app se esté ejecutando, seguramenre necesitaremos seleccionar los objetos de la lista, por ejemplo, una pregunta en particular conforme el usuario realiza un cuestionario, o un número de teléfono concreto elegido de entre una lista. Podemos acceder a los objetos de una lista usando un **índice**, esto es, especificando la posición de ese objeto dentro de la lista. Si una lista tiene tres objetos, podemos acceder a cada uno de ellos mediante los índices 1, 2, y 3. Para ello podemos usar el bloque "select list item", como muestra la figura:

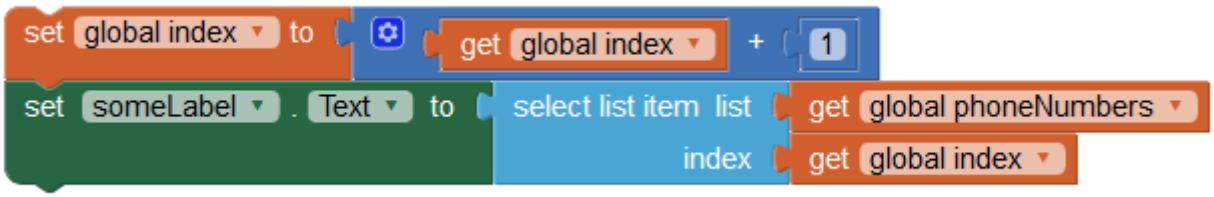
Al bloque "select list item" debemos indicarle la lista a la que queremos acceder (parámetro "list") y el índice del objeto que queremos seleccionar (parámetro "index"). En este ejemplo estamos seleccionando el segundo objeto de la lista "phoneNumbers", que es el númerod de teléfono "$333-4444$".

USAR UN ÍNDICE PARA RECORRER UNA LISTA.

En muchas aplicaciones definiremos una lista de datos, y le permitiremos al usuario *recorrer* la lista. La app "GeographyQuiz" que programaremos en este capítulo constituye un buen ejemplo: En esta app le planteamos al usuario un cuestionario de geografía. Cada vez que el usuario pulsa en el botón Next (siguiente), la app pasa selecciona el siguiente objeto de una lista de preguntas, y lo muestra por pantalla.

En la sección previa mostramos cómo seleccionar el segundo objeto de una lista, ¿pero cómo seleccionar el *siguiente* objeto? Cuando recorremos una lista, el índice del objeto que estamos seleccionando cambia cada vez: Este índice es nuestra posición actual dentro de la lista. Por consiguiente, necesitamos definir una variable para representar esa posición actual. "index" es el nombre que se le suele dar a la variable encargada de ello, y habitualmente se inicializa a 1 (la primera posición de la lista), como muestra la figura:

Cuando el usuario hace algo para moverse al siguiente objeto de la lista, lo que hacemos es incrementar la variable "index" en una unidad (sumándole un valor de 1), y a continuación, seleccionamos el objeto de la lista en ese índice actualizado. La figura muestra los bloques necesarios para hacerlo:



EJEMPLO: RECORRER UNA LISTA DE COLORES DE PINTURA.

Consideremos una app en la que el usuario puede ver cómo quedará cada color de pintura antes de pintar su casa, pulsando en un botón "ColorButton". Cada vez que el usuario pulse el botón, el color del botón cambia. Cuando el usuario pasa por todos los colores posibles, la app vuelve al primer color de la lista.
Para este ejemplo vamos a usar unos pocos colores básicos. Sin embargo, podríamos personalizar esta app a n uestro gusto iterando a través de cualquier conjunto de colores.

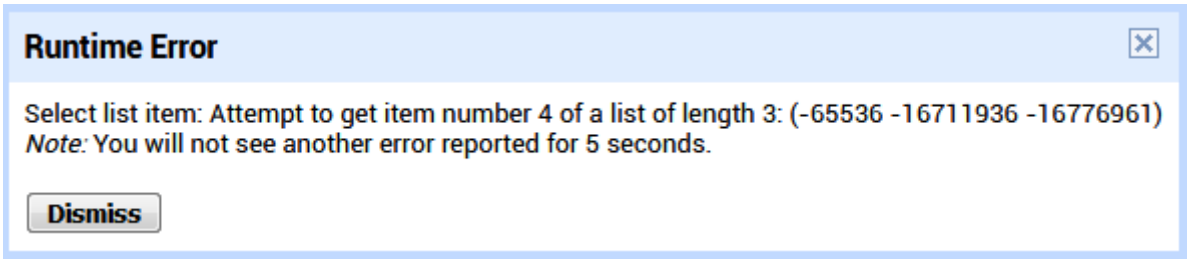
El primer paso es definir una variable de tipo lista para la lista de colores, e inicializarla a los colores que contendrá esa lista, como muestra la figura:

A continuación dfinimos una variable índice que almacene la posición actual en la lista. Deberíamos inicializarla a 1. Podemos darle un nombre descriptivo, como "currentColorIndex", pero si en la app no tenemos múltiples índices y no hay posibilidad de confusión, podemos llamarla simplemente "index":

El usuario pasa al siguiente color de la lista pulsando el botón "ColorButton". Cada vez que pulse, el índice debería incrementarse y el color de fondo del botón (propiedad "BackgroundColor") debería cambiar al color actualmente seleccionado de la lista, como muestra la figura:

Suponer que el color de fondo del botón los hemos fijado inicialmente a rojo en el Diseñador de Componentes. La primera vez que el usuario pulse el botón, la variable "index" cambia de su valor inicial 1 al valor 2, y el color de fondo del botón cambia al segundo elementos de la lista, que es el color verde. La segunda vez que el usuario pulse el botón, la variable "index" cambia de 2 a 3 y el color de fondo cambia a azul.

¿Pero qué crees que ocurrirá la siguiente vez que el usuario pulse el botón? Si piensas que ocurrirá un error, estás en lo cierto: La variable "index" toma el valor 4 y la app tratará de seleccionar el cuerto elemento de la lista, pero la lista solo tiene tres objetos. La app se cerrará abruptamente, y el usuario verá un mensaje de error como el mostrado en la figura:



Obviamente, no es deseable que el usuario final vea un mensaje como ése. Para evitar este problema, añadimos un bloque "if" para comprobar si hemos llegado al último color de la lista. En ese caso, cambiamos el valor de "index" de vuelta a 1 para volver a mostrar el primer color de la lista. El código para hacerlo se muestra en la figura:

Cuando el usuario pulse el botón, se incrementa la variable "index" y se comprueba si su nuevo valor es demasiado grande. Para ello comparamos el valor actual de "index" con el bloque "length of list", no con 3. De esta forma nuestra app seguirá funcionando aunque añadamos más elementos a la lista. Comprobando si el valor del índice es mayor que la longitud de la lista (en vez de comprobar si el índice es mayor que el número 3), hemos eliminado una *dependencia de código* en nuestra app. Una **dependencia de código** es un término de programación que describe un código que se define de forma demasiado específica y que carece de flexibilidad. Así, si cambiamos algo en algún sitio (en nuestro caso, si añadimos objetos a la lista de colores) deberemos buscar cada instancia en la que hayamos usado esa lista y cambiarla explícitamente.

Como nos podremos imaginar, este tipo de dependencias se vuelven muy enrevesadas, y habitualmente producen más malfuncionamientos de los queseremos capaces de detectar. De hecho, nuestra app adolece de otra dependencia de código en su programa. ¿Puedes ver dónde está?

Si cambiásemos el primer color de nuestra lista de rojo a algún otro color, la app no funcionará correctamente a menos que nos acordemos de cambiar en el Diseñador de Componentes el color de fondo inicial del botón a ese nuevo color. Una forma de eliminar esta dependencia de código es fijar el valor inicial de la propiedad "ColorButton.BackgroundColor" al primer color de la lista de colores, en vez de fijarlo a un color específico en el Diseñador. Como este cambio implica un comportamiento que ocurre nada más arrancar la app, vamos a incluirlo dentro del manejador de evento "when (Screen1).Initialize", como muestra la figura:

9.3. CREAR FORMULARIOS DE ENTRADA Y DATOS DINÁMICOS.

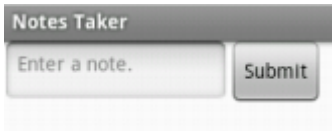
La aplicación previa de los colores usaba una lista *estática*, esto es, una lista cuyos elementos los define el programador, y no pueden cambiar a menos que no cambiemos el código del programa. Sin embargo, es muy habitual que las apps deban tratar con datos dinámicos, a saber, información que cambia cuando el usuario inserta nuevos objetos, o cuando se cargan nuevos elementos de una base de datos o de una web. En esta sección crearemos una app llamada "NoteTaker" para tomar notas, en la que el usuario escribe notas en un formulario y además puede ver todas las notas previamente escritas.

DEFINIR UNA LISTA DINÁMICA.

Las apps como "NoteTaker" comienzan con una lista vacía. Cuando queremos que una lista comience vacía, definimos una variable lista y la inicializamos vacía con el bloque "empty list", como muestra la figura:

AÑADIR UN OBJETO A LA LISTA DINÁMICA.

La primera vez que alguien lance la app, la lista "notes" estará vacía. Pero cuando el usuario escriba datos en un formulario y pulse el botón Submit (enviar), la nueva nota debería añadirse a la lista. El formulario podría ser tan simple como el mostrado en la figura:



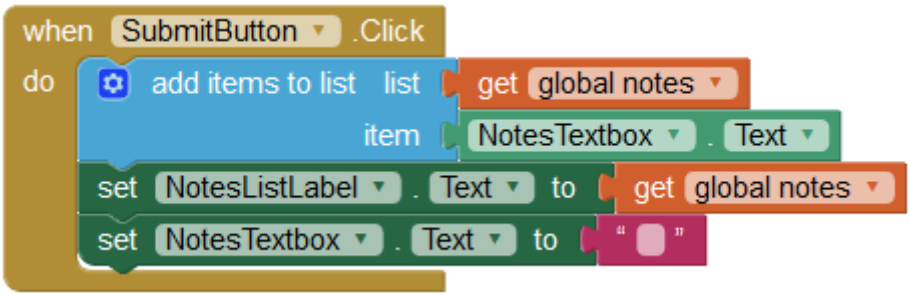
Cuando el usuario escribe una nota en la caja de texto y pulsa el botón Submit, la app llama a la función "add ítems to list" para añadir un nuevo elemento a la lista y borra de la caja de texto la nota añadida, como muestra la figura:

El bloque "add ítems to list" se encuentra en la bandeja "Lists". Pero debemos tener cuidado: También hay un bloque "append to list", de uso muy poco frecuente, que se usa para añadir toda una lista a otra lista.

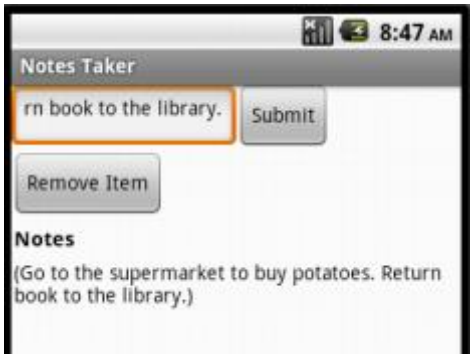
MOSTRAR UNA LISTA.

Como en el caso del resto de variables, los contenidos de una variable tipo lista no son visibles para el suaurio de la app. Los bloques de la figura previa añaden objetos a la lista cada vez que se invoca el evento "(SubmitButton).Click", pero el usuario no recibirá ningún tipo de información indicándole que la lista setá creciendo, a no ser que añadamos un código que muestre los contenidos de la lista.

La forma más sencilla de mostrar una lista en la inbteraz de usuario de una app es usar el mismo método que empleamos para mostrar números o textos, esto es, poner la lista en la propiedad "Text" de un componente "Label", como ilustra la figura:



Por desgracia, esta forma de mostrar las listas no es muy elegante, ya que pone la lista entre paréntesis, con cada elemento de la lista separado por un espacio, y no necesariamente en la misma línea. Por ejemplo, si el usuario escribe "Go to the supermarket to buy potatoes" como primera nota de la lista, y "Return book to the library" como segundo elemento, la app mostrará las notas de forma similar a lo ilustrado en la figura:



En próximas secciones veremos una forma más sofisticada de mostrar una lista.

QUITAR UN ELEMENTO DE UNA LISTA.

Podemos quitar un elemento de una lista usando el bloque "remove list item", como muestra la figura:

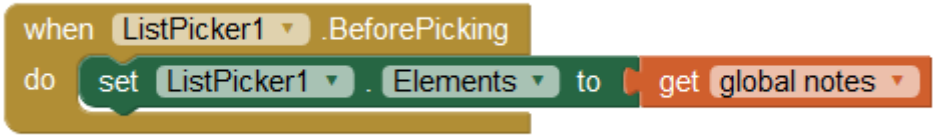
Los bloques de la figura retiran el segundo elemento de la lista "notes". Sin embargo, lo habitual es que no queramos quitar un elemento fijo (por ejemplo, el segundo elemento de la lista); más bien deberemos proporcionar un mecanismo para que el usuario elija qué elemento quiere eliminar.

Para ello, podemos usar el componente "ListPicker" para permitirle al usuario elegir un objeto de la lista. El componente "ListPicker" viene con un botón asociado. Cuando el usuario pulsa el botón, el componente "ListPicker" muestra los objetos de la lista, y el usuario puede elegir uno de ellos. Cuando el usuario elige un objeto, la app puede pasar a eliminarlo.

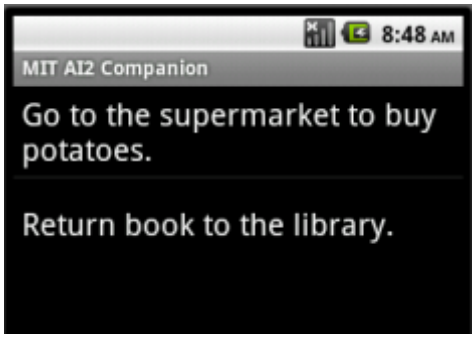
El componente "ListPicker" es fácil de programar si entendemos sus eventos clave ("BeforePicking" y "AfterPicking") y sus propiedades clave ("Elements", "Selection", y "SelectionIndex"), ver tabla:

Event	Property
`BeforePicking`: Triggered when button is clicked.	`Elements`: The list of choices.
`AfterPicking`: Triggered when user makes a choice.	`Selection`: The user's choice.
	`SelectionIndex`: Position of choice.

Cuando el usuario pulsa el botón asociado a un componente "ListPicker" se activa el evento "(ListPicker).BeforePicking". En el manejador de evento "when (ListPicker).BeforePicking" fijamos la propiedad "(ListPicker).Elements" a la variable tipo lista para poder mostrar los datos contenidos en la lista. Para la app "NotesTaker", fijaremos la propiedad "Elements" a la variable "notes" que contiene nuestra lista de notas, como en el programa de la figura:

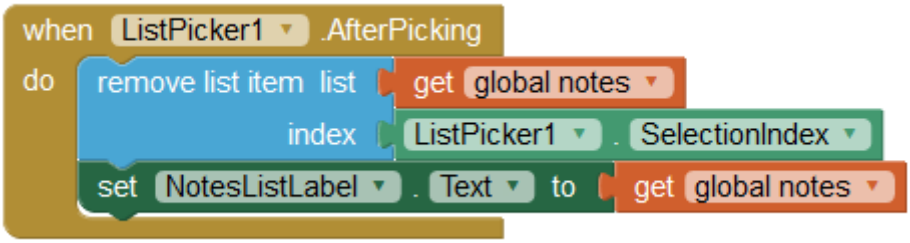


Con estos bloques, los objetos de la lista de notas aparecerán en el "ListPicker". Si hibiera dos notas, éstas aparecerían como en la figura:



Ahora, cuando el usuario elige un elemento de la lista en el "ListPicker", se activa el evento "(ListPicker).AfterPicking". En su manejador de evento podemos accder a la selección del usuario a través de la propirad "(ListPicker).Selection".

Pero nuestro objetivo en este ejemplo es quitar un elemento de la lista, y el bloque "remove item from list" espera recibir un índice, y no el propio elemento a eliminar. La propiedad "Selection" del componente "ListPicker" es el dato (esto es, la nota), y no el índice de ese dato en la lista. Por consiguiente, aquí necesitamos usar la propiedad "SelectionIndex", que es la que alberga el índice del elemento que hemos seleccionado. Este valor es el que debemos proporcionar al parámetro "index" del bloque "remove item from list", como ilustra la figura:



9.4. COMENZAR CON LA APP "PARISMAPTOUR".

En este capítulo vamos a construir una app turística para un viaje a París que incluya mapas de Google Maps. Recordemos que App Inventor proporciona dos componentes que nos serán útiles en esta tarea: El componente "ActivityStarter" permite lanzar otras aplicaciones (como Google Maps) desde nuestra app, y el componente "WebViewer" muestra cualquier página web dentro de un panel en nuestra app. Usaremos estos dos componentes para construir dos versiones diferentes de nuestra app de viajes.

9.5. DISEÑAR LOS COMPONENTES DE "PARISMAPTOUR".

Creamos un nuevo proyecto en App Inventor llamado "ParisMapTourV1". (Más adelante haremos la segunda versión, a la que llamaremos "ParisMapTourV2"). La interfaz de usuario de la app tiene un componente "Image" con una imagen de Paris, un componente "Label" con un texto, un componente "ListPicker" que vendrá acompañado de su botón asociado, y para la primera versión de la app, un componente "ActivityStarter" (no visible).

La tabla resume los componentes necesarios, la bandeja a la que pertenecen, qué nombre les pondremos, y su función dentro del proyecto. Añade estos componentes en el Diseñador para que la app se parezca a la mostrada en la figura previa.

Component type	Palette group	What you'll name it	Purpose
`Image`	User Interface	`Image1`	Show a static image of Paris on screen.
`Label`	User Interface	`Label1`	Display the text "Discover Paris with your Android!"
`ListPicker`	User Interface	`ListPicker1`	When clicked, a list of destination choices will appear.
`ActivityStarter`	Connectivity	`ActivityStarter1`	Launch the Maps app when a destination is chosen.

9.6. AJUSTAR LAS PROPIEDADES DE ACTIVITY STARTER.

"ActivityStarter" es un componente con el que podemos lanzar desde nuestra app cualquier aplicación de Android, como Google Maps o una de nuestras propias aplicaciones. Como vamos a construir una app para una guía turística de Paris, queremos que la app lance la aplicación Maps para mostrar un mapa basado en el destino elegido por el usuario. Después, el usuario podrá presionar en un botón para volver a nuestra app y elegir un destino diferente.

"ActivityStarter" es un componente de bajo nivel en el que tendremos que ajustar algunas propiedades con información que le sería familiar a un programador de Java Android SDK, pero que es totalmente ajena al resto de las personas. Para esta app debemos insertar las propiedades especificadas en la tabla. Pero debemos ser muy cuidadosos, porque estos datos deben escribirse respetando las mayúsculas y las minúsculas.

Property	Value
`Action`	android.intent.action.VIEW
`ActivityClass`	com.google.android.maps.MapsActivity
`ActivityPackage`	com.google.android.apps.maps

Hay una propiedad más, llamada "DataUri", que permite especificar una URL para lanzar un mapa específico en Google Maps. Pero esta propiedad debe ajustarse en el Editor de Bloques y no en el Diseñador de Componentes porque es *dinámica*, lo que significa que cambiará dependiendo de si el usuario decide visitar la torre Eiffel, el museo del Lovre, o la catedral de Notre Dame.

9.7. VERSIÓN 1: TOUR POR PARIS CON UN ACTIVITY STARTER.

En el Editor de Bloques hemos de definir una lista de destinos y dos comportamientos:

- Al arrancar, la app debe cargar los destinos posibles en el componente "ListPicker", para que el usuario pueda elegir uno de ellos.
- Cuando el usuario elige uno de los destinos disponibles en el "ListPicker", se lanza la aplicación Maps para mostrar un mapa de ese destino. En esta primera versión de la app simplemente abriremos Google Maps en el navegador del dispositivo, y le indicaremos que realice una búsqueda del destino elegido por el usuario.

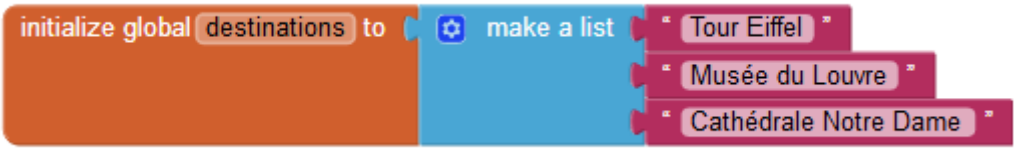
CREAR UNA LISTA DE DESTINOS.

Abrimos el editor de bloques y creamos una variable con la lista de destinos de Paris. Para ello, usamos los bloques listados en la tabla.

Block type	Drawer	Purpose
`initialize global` ("Destinations")	Variables	Create a list of the destinations.
`make a list`	Lists	Add the items to the list.
`text` ("Tour Eiffel")	Text	The first destination.
`text` ("Musée du Louvre")	Text	The second destination.
`text` ("Cathédrale Notre Dame")	Text	The third destination.

Cuando agreguemos el bloque "make a list" a nuestra app aparecerá por defecto con solo dos conectores disponibles. Podemos añadirle más conectores clicando en el icono azul del engranaje, y añadiendo todos los elementos "item" que necesitemos (tres en total para nuestra app).

Después de haber habilitado los tres conectores, creamos tres bloques de texto (uno para cada destino posible), y los enganchamos a los tres conectores disponibles, como muestra la figura:

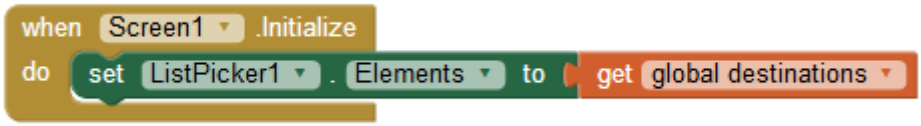


PERMITIR AL USUARIO ELEGIR UN DESTINO.

La lista que acabamos de definir no aparece en la interfaz de usuario (ninguna variable lo hace). Usaremos un componente "ListPicker" para mostrar la lista de opciones entre las que el usuario puede elegir. Primero cargamos los destinos posibles en el componente "ListPicker" fijando su propiedad "Elements" a la lista "destinations" recién creada. Y como esto solo debemos hacerlo una vez al arrancar la aplicación, definiremos este comportamiento en el manejador de eventos "when (Screen1).Initialize". Los bloques necesarios para construir este código se listan en la tabla:

Block type	Drawer	Purpose
`Screen1.Initialize`	Screen1	This event is triggered when the app starts.
`set ListPicker1.Elements to`	ListPicker1	Set this property to the list that you want to appear.
`get global destinations`	Drag out from variable initialization block	The list of destinations.

La figura muestra el programa necesario:



¿Cómo funcuiona este código? El manejador "when (Screen1).Initialize" se activa nada más arrancar la app. La figura ilustra que el manejador de eventos fija la propiedad "Elements" del componente "ListPicker" a la lista "destinations" para que aparezcan los tres destinos posibles como opcines de selección para el usuario.

Vamos a probar la app: En App Inventor pinchamos en "Connect" y configuramos las pruebas en vivo en nuestro dispositivo Android o en el emulador. A continuación, y ya en el dispositivo o emulador, presionamos el botón con el texto "Choose Paris destination". El componente "ListPicker" debería aparecer con los tres destinos posibles. Por supuesto, todavía no ocurre nadacuando seleccionamos uno de los tres destinos.

ABRIR MAPAS CON UNA BÚSQUEDA DE URL.

Ahora vamos a programar la app para que cuando el usuario seleccione uno de los destinos, el componente "ActivityStarter" lance Google Maps y busque la localización seleccionada.

En primer lugar, condiseremos la URL http://maps.google.com?q=Paris. Cuando escribimos esta URL en la barra de direcciones de un navegador de Internet, éste nos muestra un mapa de París. El carácter "?" es

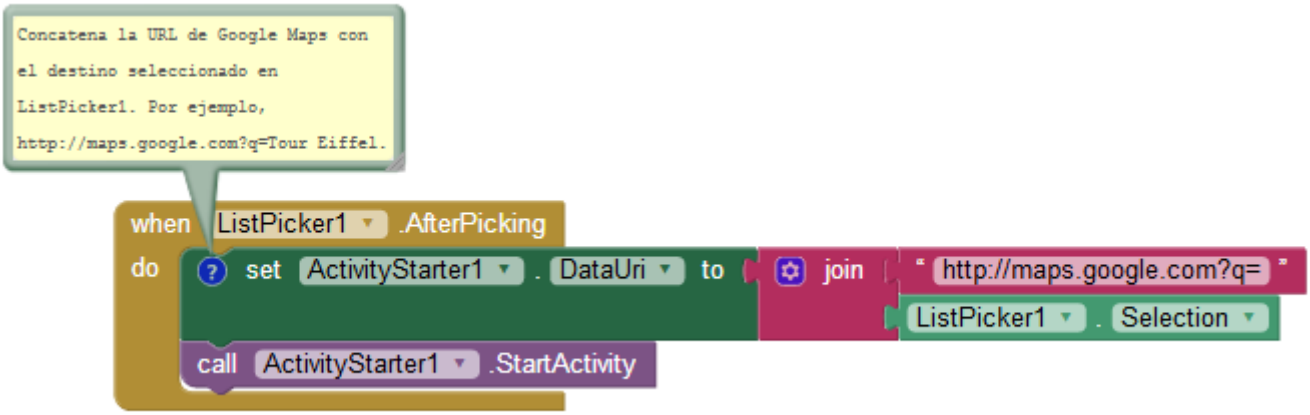
muy común en muchas URLs, y se usa para denotar el envío de un parámetro que la página web necesita para procesar la solicitud. En este caso, el nombre del parámetro es "q", abreviatura de "query" (consulta), y su valor es "Paris". Esto le indica a Google Maps qué mapa debe mostrar.

En esta app vamos a construir la URL dinámicamente, añadiendo el valor del parámetro según la localización elegida por el usuario. De esta forma podremos mostrar diferentes mapas en función de la elección del usuario.

Cuando el usuario elige un elemento del componente "ListPicker", se activa el manejador de eventos "when (ListPicker).AfterPicking". En este manejador fijaremos el valor de la propiedad "DataUri" del componente "ActivityStarter" para que sepa qué mapa debe abrir. A continuación lanzaremos la aplicación Google Maps llamando a la función "(ActivityStarter).StartActivity". Los bloques necesarios para programar estafuncionalidad están listados en la tabla:

Block type	Drawer	Purpose
`ListPicker1.AfterPicking`	ListPicker1	This event is triggered when the user chooses from `List Picker`.
`set ActivityStarter1.DataUri to`	ActivityStarter1	The `DataUri` instructs Maps which map to open on launch.
`join`	Text	Build the `DataUri` from two pieces of text.
`text` ("http://maps.google.com?q=")	Text	The first part of the `DataUri` expected by Maps.
`ListPicker1.Selection`	ListPicker1	The item the user chose.
`ActivityStarter1.StartActivity`	ActivityStarter1	Launch Maps.

El programa que usa estos bloques se muestra en la figura:



¿Cómo funciona este código? Cuando el usuario selecciona un destino en "ListPicker1", el elemento elegido se almacena en la propiedad "ListPicker1.Selection", e inmediatamente se desencadena el evento "(ListPicker1).AfterPicking". Como muestra la figura, la propiedad "DataUri" de "ActivityStarter1" se ajusta a un texto que combina "http://maps.google.com/?q=" con el destino elegido. Así pues, si el usuario selecciona el primer elemento de "ListPicker1", a saber, "Tour Eiffel", la propiedad "DataUri" toma el valor "http://maps.google.com/?q=Tour Eiffel".

Como en el Diseñador de Componentes ya fijamos las otras propiedades de "ActivityStarter1" para que pueda abrir Google Maps, el bloque "ActivityStarter1.StartActivity" lanza la aplicación Maps e invoca la búsqueda prescrita por "DataUri".

Vamos a probar nuestra app: Reiniciamos la app y presinamos el botón "Choose Paris destination". Al elegir uno de los destinos, debería aparecer un mapa donde aparezca ese destino. Ahora comporbamos que podemos volver a nuestra app presionando el botón "hacia atrás" de nuestro dispositivo.

9.8. VERSIÓN 2: TOUR POR PARIS CON UN WEB VIEWER.

"ActivityStarter" es un componente importante porque porporciona acceso a cualuqie otra aplicación desde nuestra app. Pero hay otra forma de construir nuestra app, usando el componente "WebViewer". Recordar que el componente "WebViewer" es un panel que podemos ubicar directamente dentro de nuestra app y que se comporta como un navegador integrado. Con él podemos abrir cualquier página web en el propio visor Web, y podemos cambiar mediante programación la página que muestra. Al contrario de los que ocurre con un "ActivityStarter", al usar un "WebViewer" el usuario no tienen que abandonar nuestra app, y no se ve obligado a volver a ella presionando el botón "hacia atrás" del dispositivo.

En esta segunda versión de la app usaremos el componente "WebViewer", y además mejoraremos la app para que abra algunas vistas ampliadas de las calles y de los monumentos de Paris. Definiremos una segunda lista y usaremos un esquema algo más complejo para decidir qué mapa vamos a mostrar.

Para empezar, vamos a explorar Google Maps para obtener las URLs de algunos mapas específicos. Seguiremos usando algunos lugares emblemáticos de Paris como destinos, pero cuando el usuario elija uno de ellos, usaremos el índice (esto es, suposición en la lista) de esta selección para buscar y abrir algunas vistas ampliadas específicas en el mapa.

Antes de empezar, es conveniente que guardemos nuestro proyecto (usando la opción "Projects" → "Save project as…") con el nombre "ParisMapTourV2", para abrir una copia de la aplicación tal y como está hasta ahora. De esta forma podremos modificar la app sin preocuparnos por machacar la versión 1, que ahora mismo es plenamente operativa.

AÑADIR EL COMPONENTE WEB VIEWER.

En el Diseñador, borramos el componente "ActivityStarter". A continuación, añadimos el componente `WebViewer` desde la bandeja "User Interface", y lo ubicamos debajo del resto de componentes. Desactivamos la propiedad "Scrollable" de "Screen1" para que el visor Web pueda mostrar las páginas correctamente.

HALLAR LAS URLS DE MAPAS ESPECÍFICOS.

El siguiente paso es abrir Google Maps en el ordenador y obtener los mapas específicos que queremos lanzar para cada destino:

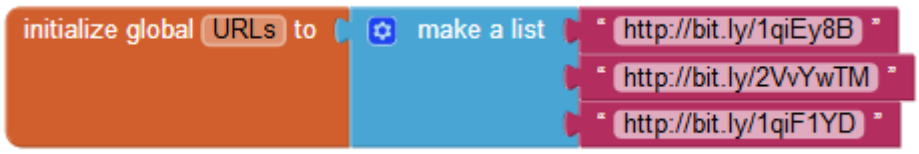
1) En nuestro ordenador acudimos al navegador, y en la bara de direcciones escribimos https://www.google.com/maps.
2) Buscamos un monumento, por ejemplo, la torre Eiffel.
3) Acercamos el mapa con la herramienta de zoom todo lo que necesitemos.
4) Elegimos el tipo de vista que queramos (por ejemplo, Street View).
5) Copiamos la URL del mapa que estamos visualizando. En las versiones actuales de google Maps basta copiar la URL para el mapa directamente de la barra de direcciones del navegador.

Usamos esta tácnica para crear algunos mapas interesantes de los moumentos de Paris y extraer sus URLs. La tabla 6.6. proporciona algunos ejemplos específicos. Aquí hemos usado el servivio bit.ly (https://bitly.com/) para acortar las URLs).

Monumento	URL del mapa
Tour Eiffel	http://bit.ly/1qiEy8B
Musée du Louvre	http://bit.ly/2VvYwTM
Cathédrale Notre Dame (street view)	http://bit.ly/1qiF1YD

DEFINIR UNA LISTA DE URLS.

Necesitaremos una lista llamada "URLs", que contenga una URL para cada uno de los destinos de la lista "destinations". Creamos la lista mostrada en la figura, de forma que cada URL de esta lista se corresponda con el destino corresppondiente de la lista de destinos (esto es, la primera URL debería corresponderse con el primer destino, la torre Eiffel).



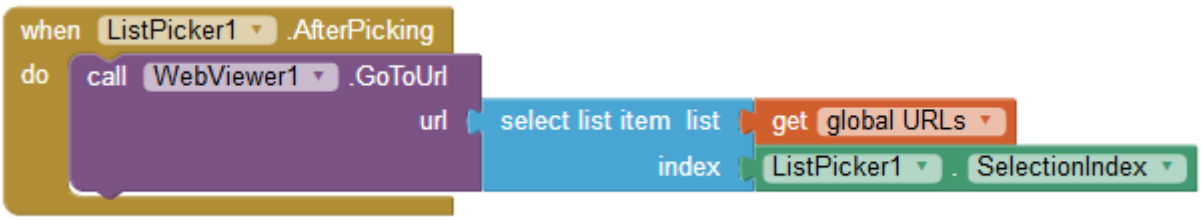
MODIFICAR EL COMPORTAMIENTO DEL MANEJADOR DE EVENTOS "WHEN LISTPICKER1.AFTERPICKING".

En la primera versión de esta app, el comportamiento del manejador "when ListPicker1.AfterPicking" fijaba el valor de "DataUri" a una combinación del texto "http://maps.google.com/?q=" y el destino que el usuario selecciona de la lista (por ejemplo, "Tour Eiffel"). En esta segunda versión el comportamiento de este manejador debe ser más sofisticado, porque el usuario selecciona un destino de la lista "destinations", pero la app elige la URL del mapa sociado de la lista "URLs". En concreto, cuando el usuario elija un destino del componente `ListPicker`, enecsitaremos conocer el índice de ese elemento de la lista para usarlo a la hora de seleccionar el elemento correspondiente de la lista de URLs.

Para implementar esta funcionalidad necesitaremos los bloues listados en la tabla:

Block type	Drawer	Purpose
`ListPicker1.AfterPicking`	ListPicker1	This event is triggered when the user chooses an item.
`ListPicker1.SelectionIndex`	ListPicker1	The index (position) of the chosen item.
`select list item`	Lists	Select an item from the URLs list.
`get global URLs`	Drag it from the variable initialization	The list of URLs.
`WebViewer.GoToURL`	WebViewer	Load the URL in the viewer to show the map.

El código que usa estos bloques e implementa esta funcionalidad se muestra en la figura:



¿Cómo funciona este programa? Cuando el usuario seleccina un destino del componente "ListPicker", se activa el evento "ListPicker.AfterPicking". El elemento elegido (por ejemplo, "Tour Eiffel") se guarda en la propiedad "ListPicker1.Selection", y en la versión 1 de la app nos bastó con ella. Pero "ListPicker" también posee una prpopiedad llamada "SelectionIndex", que se corresponde con la posición del elemento elegido en la lista. Así, si el usuario elige el destino "Tour Eiffel", la propiedad "SelectionIndex" toma el valor 1; si elige "Mosée du Louvre", "SelectionIndex" toma el valor 2; y si elige "Cathédrale Notre Dame", "SelectionIndex" toma el valor 3.

En esta segunda versión usamos la propiedad "ListPicker.SelectionIndex" para seleccionar un elemento de la lista "URLs". Esto funciona bien porque los elementos de las dos listas, "destinations" y "URLs", están en sintonía: El primer destino se corresponde con la primera URL, el segundo con la segunda, etc. Por lo tanto, aunque el usuario elija un elemento de una lista, podemos usar su selección (realmente, el índice de su selección) para tomar la URL apropiada de otra lista.

Ahora probamos nuestra app: En el dispositivo pinchamos en el botón "Choose Paris destination". Debería desplegarse una lista con tres elementos. Al elegir uno de ellos, la app deberóa mostrar un mapa integrado del destino seleccionado.

9.9. MODIFICACIONES A "PARISMAPTOUR".

Aquí proponenemos algunas variaciones que podemos probar:

- Crea una app para un tour virtual de tu escuela, de tu pueblo, ciudad, o barrio de residencia, o de tu lugar de vacaciones.
- Explora las distintas posibilidades del componente "ActivityStarter" y úsalo para enviar un correo electrónico o lanzar una app como YouTube. (Para una ayuda sobre este componente, consulta la web http://appinventor.mit.edu/explore/ai2/activity-starter.html)
- Crea una app para un tour virtual customizable, que le permita al usuario crear una guía para un lugar de su elección, introduciendo el nombre de cada destino de interés junto con la URL del mapa correspondiente. Necesitaremos guardar los datos en una base de datos "TinyWebDB" y crear una app que trabaje con los allí insertados.

9.10. COMENZAR CON LA APP "GEOGRAPHYQUIZ".

En las próximas secciones vamos a desarrollar un cuestionario sobre geografía, aunque podemos usarlo como modelo para construir un cuestionario similar sobre cualquier otro tema.

Para desarrollar esta app necesitaremos usar dos variables tipo lista para almacenar los datos (en este caso, las distintas preguntas y sus correspondientes respuestas), y utilizar una variable tipo índice para saber en qué pregunta del juego se encuentra el usuario. Para cuando terminemos esta app habremos aprendido cómo crear apps de preguntas y respuestas, o cualquier otra app que requiera un procesado de listas.

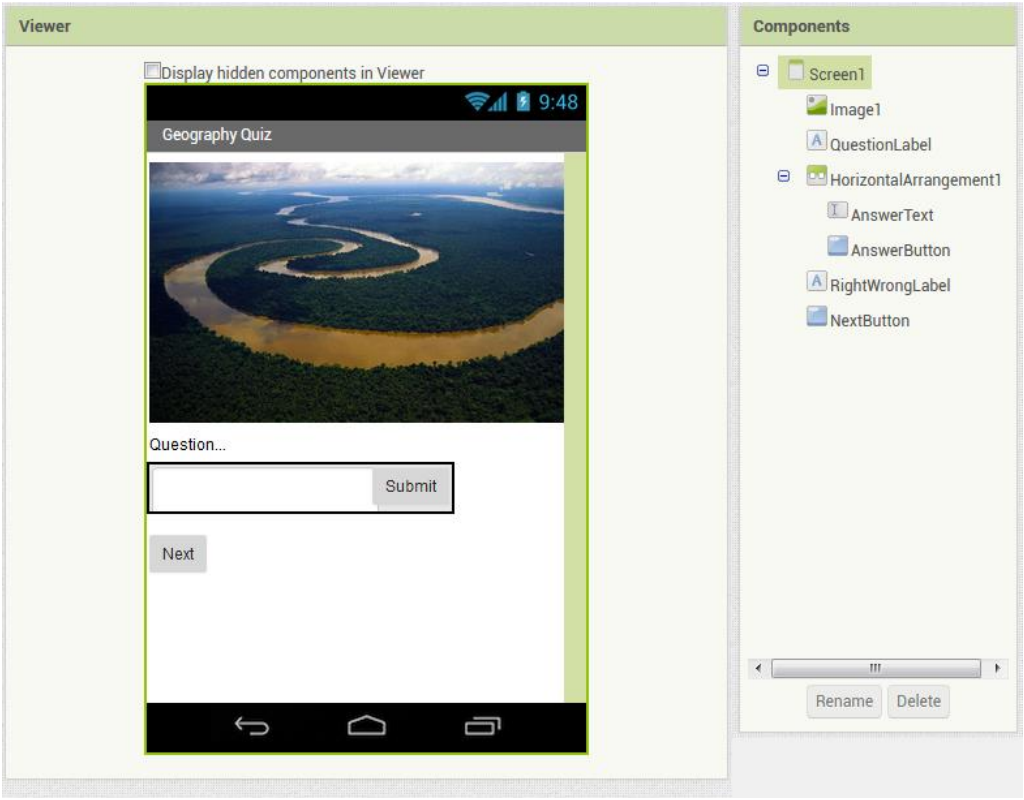
Nos conectamos a la web de App Inventor y comenzamos un nuevo proyecto al que llamaremos "GeographyQuiz". Fijamos el título de la pantalla a "Geography Quiz". Conectamos nuestro dispositivo o el emulador para hacer pruebas en vivo.

Para esta app necesitaremos algunas imágenes que encontraremos en los archivos de los alumnos. Estas imágenes son *Amazonas.jpg*, *EmpireState.jpg*, *London.jpg*, y *Pyramids.jpg*.

Cargaremos estas imágenes a nuestro proyecto en la próxima sección.

9.11. DISEÑAR LOS COMPONENTES DE "GEOGRAPHYQUIZ".

La app "GeographyQuiz" tiene una interfaz de usuario muy sencilla, que simplemente muestra las preguntas y le permite al usuario insertar su respuesta. La figura muestra la vista de Diseñador de esta app para ayudarnos a construir la interfaz de usuario.



Para crear esta interfaz, en primer lugar vamos a cargar en el proyecto las imágenes antes indicadas. En el área "Media" del Diseñador, pinchamos en "Upload File…" y subimos una a una las imágenes del proyecto. A continuación, agregamos los componentes listados en la tabla. Respecto a las propiedades de los componentes, usamos estos valores:

1) Fijamos la propiedad "Picture" de "Image1" a la imagen *Amazonas.jpeg*, la primera imagen que debería aparecer. Fija sus propiedades "Width" y "Height" a "Fill parent" y 200 píxeles, respectivamente.
2) Fijamos la propiedad "Text" de "QuestionLabel" a "Question…". (Pondremos el texto de la primera pregunta mediante el Editor de Bloques).
3) Fijamos la propiedad "Hint" de "AnswerText" a "Enter an answer". Deja su propiedad "Text" en blanco. Movemos este componente dentro de "HorizontalArrangement1".
4) Cambiamos la propiedad "Text" de "AnswerButton" a "Submit" y lo movemos dentro de "HorizontalArrangement1".
5) Cambiamos la propiedad "Text" de "NextButton" a "Next".
6) Dejamos la propiedad "Text" de "RightWrongLabel" en blanco.

Component type	Palette group	What you'll name it	Purpose
Image	User Interface	Image1	The picture displayed with the question.
Label	User Interface	QuestionLabel	Display the current question.
HorizontalArrangement	Layout	HorizontalArrangement1	Put the answer text and button in a row.
TextBox	User Interface	AnswerText	The user will enter his answer here.
Button	User Interface	AnswerButton	The user clicks this to submit an answer.
Label	User Interface	RightWrongLabel	Display "correct!" or "incorrect!"
Button	User Interface	NextButton	The user clicks this to proceed to the next question.

9.12. AÑADIR COMPORTAMIENTOS A LOS COMPONENTES DE "GEOGRAPHYQUIZ".

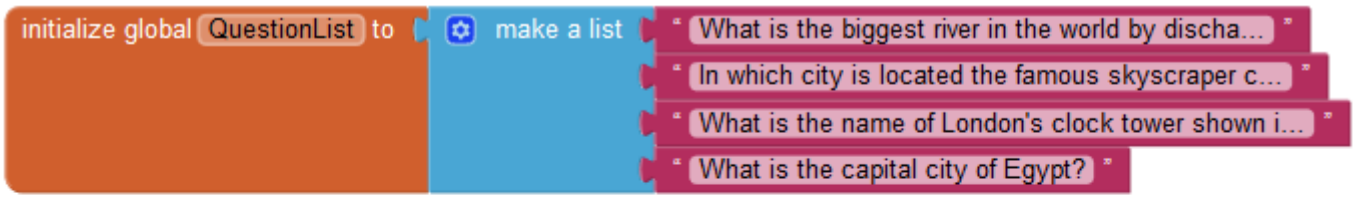
Tenemos que programar los siguientes comportamientos:

- Cuando la app comienza aparece la primera pregunta, incluyendo su imagen correspondiente.
- Cuando el usuario pulsa el botón "NextButton", aparece la segunda pregunta. Al volver a pulsarlo, aparece la tercera, y así sucesivamente.
- Cuando el usuario llega a la última pregunta y pulsa en "NextButton", vuelve a aparecer la primera pregunta.
- Cuando el usuario responde a una pregunta, la app indica si la respuesta es correcta o no.

A partir de ahora nuestra tarea será programar cada una de estas funcionalidades.

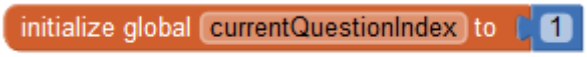
DEFINIR LAS LISTAS DE PREGUNTAS Y RESPUESTAS.

Para empezar, definimos dos variables tipo lista basadas en los elementos de la tabla. Estas variables son "QuestionList", que alojará la lista de preguntas, y "AnswerList", que almacenará la lista de las respuestas correspondientes. La figura muestra cómo crear las listas en el Editor de Bloques.



DEFINIR LA VARIABLE INDEX.

La app necesita controlar cuál es la pregunta actual conforme el usuario pulsa el botón "NextButton" para avanzar en el cuestionario. Para ello definiremos una variable llamada "currentQuestionIndex", y dicha variable servirá como índice tanto para la lista de preguntas como para la lista de respuestas. La figura muestra el código necesario:



MOSTRAR LA PRIMERA PREGUNTA.

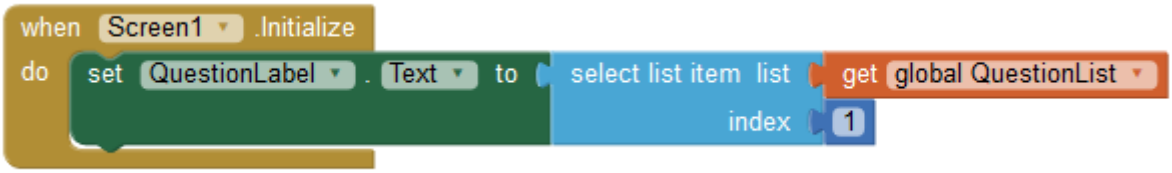
Ahora que ya hemos definido las variables que necesitamos, podemos especificar ell comportamiento interactivo de la app. Como con cualquier app, aquí será importante trabajar incrementalmente y definir un los comportamientos uno tras otro. Para empezar vamos a centrarnos en las preguntas, y específicamente, en mostrar la primera pregunta de la lista al arrancar la app. Más adelante nos ocuparemos de las imágenes asociadas y de las respuestas.

Queremos que nuestro código funcione independientemente de cuáles sean las preguntas que estén en la lista. De esa forma podremos crear un nuevo cuestionario simplemente copiando y modificando ligeramente esta app. Ello simplemente implicará cambiar las preguntas y las respuestas en las definiciones de las listas.

Así pues, para implementar este primer comportamiento es importante no referirse directamente a la primera pregunta, a saber, "What is the biggest river in the world by discharge volume of water?". En vez de ello, vamos a referirnos al primer conector de la lista "QuestionList", independientemente de cuál sea la pregunta que le hemos conectado. De esta forma el código seguirá funcionando incluso aunque modifiquemos la pregunta conectada a esa ranura.

Para seleccionar elementos concretos de una lista, usamos el bloque "select list item" de la bandeja "Lists". Este bloque recibe como argumentos un nombre de lista y un índice (una posición en la lista), y devuelve el elemento de la lista situado en ese índice. Si el elemento está en la primera posición, su índice es el 1, si está en la segunda posición, su índice es el 2, y así sucesivamente.

Ahora, cuando la app arranca queremos seleccionar el primer elemento de la lista "QuestionList", y ubicar este contenido en la etiqueta "QuestionLabel". Recordemos del capítulo previo que para hacer que algo ocurra nada más abrir una app, debemos programar el manejador de evento "Screen1.Initialize". La figura muestra el código necesario para implementar esta funcionalidad:

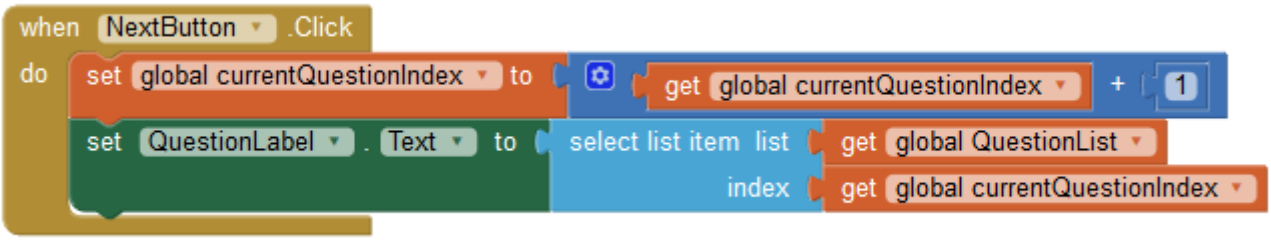


¿Cómo funciona este programa? El evento "Screen1.Initialize" se activa justo cuando se abre la app. En ese caso, seleccionamos el primer elemento de la lista "QuestionList" y lo ubicamos en la propiedad "Text" de la etiqueta "QuestionLabel". De esta forma, cuando el usuario abra la app, verá la primera pregunta.

Vamos a probar la app: En App Inventor seleccionamos "Connect" y conectamos nuestro dispositivo o el emulador para hacer pruebas en vivo. Cuando la app se cargue, veremos por pantalla el primer elemento de la lista de preguntas, "What is the biggest river in the world by discharge volume of water?".

RECORRER LA LISTA DE PREGUNTAS.

Ahora vamos a programar el comportamiento del botón "Nextbutton". Ya hemos definido la variable "currentQuestionIndex" para recordar cuál es la pregunta actual. Cuando el usuario pulse "Nextbutton", la app deberá incrementar el valor de "currentQuestionIndex" en una unidad (por ejemplo, cambiarlo de 1 a 2, o de 2 a 3, etc.). Después usaremos el valor resultante de "currentQuestionIndex" para elegir la nueva pregunta que debemos mostrar. A modo de ejercicio, intenta escribir el programa que implementa esta funcionalidad. Cuando hayas acabado, comprueba que tu código es como el de la figura:



¿Cómo funciona este programa? La primera línea de bloques dentro del manejador "when (Nextbutton).Click" incrementa el valor de la variable "currentQuestionIndex" en una unidad. Después de haber hecho este cambio, la app usa el valor actualizado de "currentQuestionIndex" para seleccionar la nueva pregunta a mostrar. Así, cuando el usuario pulse el botón "Nextbutton" por primera vez, los bloques de incremento cambiarán el valor de "currentQuestionIndex" de 1 a 2, y la app seleccionará el segundo elemento de la lista "QuestionList", a saber, "In which city is located the famous skyscraper called "Empire State Building"?". La segunda vez que el usuario pulse "Nextbutton", la variable "currentQuestionIndex" cambiará de 2 a 3, y la app mostrará la tercera pregunta de la lista, esto es, "What is the name of London's clock tower shown in the image?". Y así sucesivamente.

Vamos a probar la app. Al pulsar en el botón "Nextbutton", el teléfono debería mostrar la segunda pregunta de la lista. Al volver a pulsarlo, debería mostrar la tercera. Y si lo pulsamos una vez más, la cuarta pregunta debería aparecer por pantalla. Ahora bien, si pulsamos el botón una vez más, obtendremos un mensaje de error: "Attempting to get item 5 of a list of length 4." Esto significa que nuestra app tiene un malfuncionamiento (lo que en programación se denomina **bug**). ¿Te has dado cuenta ya de cuál es el error?

El problema de este código es que simplemente incrementa el valor de "currentQuestionIndex" para acceder a la siguiente pregunta, sin comprobar si hemos llegado al final de la lista de preguntas. Entonces, cuando el valor de "currentQuestionIndex" ya es 4 y el usuario vuelve a presionar el botón "Nextbutton", la app cambia su valor de 4 a 5. Después llama a la función "select list item" para obtener la pregunta en esa posición de la lista, pero en este caso, no hay una quinta pregunta. Como solo hay cuetro elementos en la lista de preguntas, el dispositivo Android no sabe qué hacer y lanza un mensaje de error que fuerza la parada de la app. ¿Cómo podemos hacerle saber a la app que hemos llegado al final de la lista de preguntas?

La app debe formular una pregunta cada vez que el usuario presione el botón "Nextbutton", y ejecutar un código diferente en función de la respuesta. Como sabemos que la app contiene 4 preguntas, una forma de comprobar si hemos llegado al final de la lista es preguntar si el valor de la variable "currentQuestionIndex" se ha hecho mayor que 4. Si la respuesta es afirmativa, deberíamos fijar su valor de vuelta a 1 para retornar al usuario a la primera pregunta.

Para programar este comportamiento, debemos modificar el código del manejador "when (NextButton).Click" añadiendo un bloque "if", como muestra la figura:

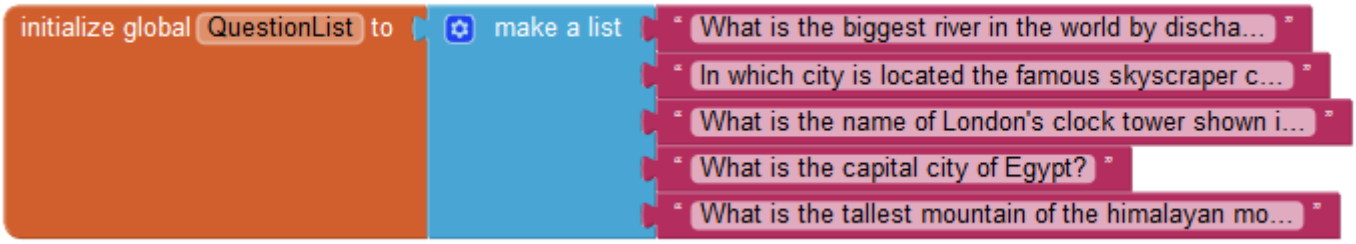
¿Cómo funciona este código? Cuando el usuario pulse "Nextbutton", la app incrementa el índice tal y como lo hacíamos antes. Pero ahora comprobamos si después de incrementar el índice, el valor de "currentQuestionIndex" se ha hecho mayor que 4. En ese caso, fijamos "currentQuestionIndex" otra vez a 1, para volver a mostrar la primera pregunta. Si es menor o igual a 4, los bloques dentro del bloque "if" no se ejecutan, y al igual que antes se muestra la pregunta actual.

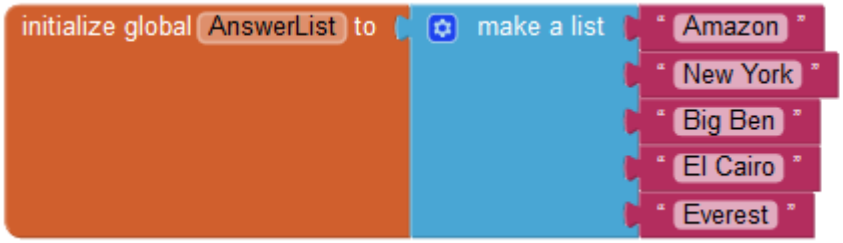
Probamos la app: al arrancar la app la pantalla muestra la primera pregunta. Ahora pulsamos "Nextbutton", con lo que el teléfono mostrará la segunda pregunta. Al pulsarlo otra vez, aparecerá la tercera pregunta. Y si lo pulsamos de nuevo veremos la cuarta pregunta. Ahora es cuando vamos a comprobar el funcionamiento que acabamos de programar: Si pulsamos una vez más, deberíamos ver que en la pantalla del teléfono vuelve a aparecer la primera pregunta.

HACER QUE EL CUESTIONARIO SEA FÁCIL DE MODIFICAR.

Parece que el botón "Nextbutton" ya funciona perfectamente, pero aún hay un problema: ¿Qué pasa si añadimos una nueva pregunta al cuestionario? ¿Seguirá funcionando el programa del botón?

Para comprobarlo, vamos a añadir una quitan pregunta a la lista de preguntas, y una quinta respuesta a la lista de respuestas:





(Además, vamos a cargar en el proyecto la imagen asociada a esta quinta pregunta, llamada *HimalayanRange.jpg*).

Si probamos la app, veremos que la quinta pregunta nunca aparece, independientemente del número de veces que pulsemos el botón "Nextbutton". ¿Te das cuenta de cuál es el problema?

El problema aquí es que la condición que hemos usado para determinar si el usuario está en la última pregunta es demasiado específica: Ahora mismo estamos comprobando si el valor de "currentQuestionIndex" es mayor que 4. Podríamos cambiar este número de 4 a 5 en el bloqiue "if", y la app volvería a funcionar correctamente, pero de nuevo, esta solución sería muy poco general, porque nos obligaría a volver a cambiar la condición del bloque "if" cada vez que modificásemos los tamaños de la listas de preguntas y respuestas.

Este tipo de dependencias no son buenas en programación, porque suelen producir malfuncionamientos, especialmente cuando aumenta la complejidad de la app. Una estrategia mucho mejor es diseñar los programas de forma que funcionen independientemente del número de preguntas que se tengan. Esta generalización permitirá que la adaptación del cuestionario a cualquier número de preguntas y a cualquier tema sea mucho más sencilla desde el punto de vista del programador. Además, también es esencial que si la lista con la que estamos trabajando cambia dinámicamente (por ejemplo, pensemos en una app de preguntas y respuestas que le permita al usuario añadir nuevas preguntas). En definitiva, para que un programa sea más general, no puede referirse a números concretos como 4, porque solo funcionaría para cuestionarios con cuatro preguntas.

Por lo tanto, en vez de preguntar si el valor de "currentQuestionIndex" es mayor que un número específico, vamos a preguntar si este valor es mayor que el número de elementos presentes en la lista "QuestionList". Si la app comprueba esta condición más general, funcionará incluso aunque añadamos o quitemos elementos de la lista. Con ello en mente, vamos a modificar el manejador de eventos "when (NextButton).Click" para reemplazar la comprobación del bloque "if" referida directamente al número 4.

El código necesario para esta funcionalidad se muestra en la figura:

¿Cómo funciona ahora este programa? La comprobación del "if" compara el nuevo valor de "currentQuestionIndex" con la longitud de la lista "QuestionList". Si el valor de "currentQuestionIndex" es 6, y la longitud de la lista de preguntas es 5, el valor de "currentQuestionIndex" se cambia a 1. Notar que como ahora los bloques ya no se refieren a 4 ni a ningún otro número específico, este programa funcionará independientemente del número de elementos que contenga la lista.

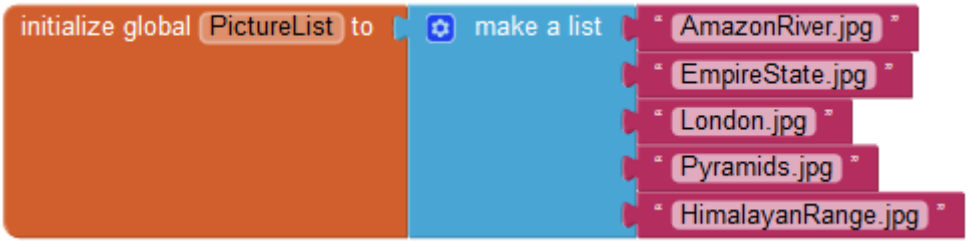
Vamos a probar la app: Al presionar repetidamente el botón "NextButton", la app debería cambiar cíclicamente entre las cinco preguntas, volviendo a la primera después de mostrar la quinta.

CAMBIAR LA IMAGEN DE CADA PREGUNTA.

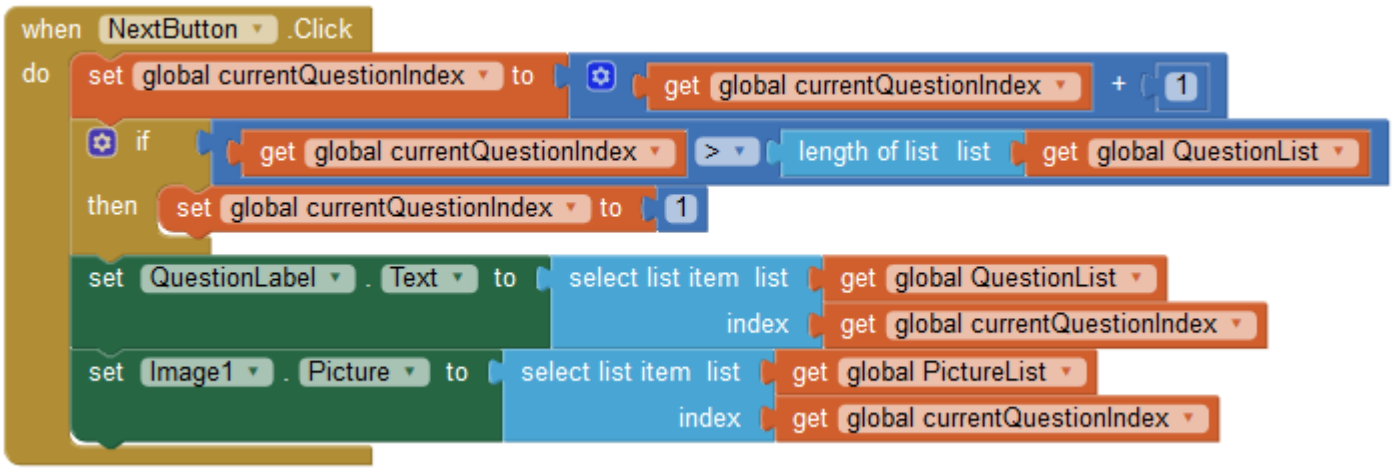
Ahora que ya hemos programado el cambio de las distintas preguntas del cuestionario (y después de haber hecho el código más flexible y general), nuestra siguiente tarea es hacer que la app muestre la imagen

correspondiente a cada pregunta. Ahora mismo, la app siempre muestra la misma imagen, independientemente de la pregunta que esté formulando. En secciones previas hemos cargado las cinco imágenes que necesitamos para este proyecto. Ahora vamos a crear una tercera lista, a la que llamaremos "PictureList", con los nombres de archivo de las imágenes como elementos constituyentes. También modificaremos el código del manejador "when (NextButton).Click" para que cambie la imagen cada vez que pasamos de una pregunta a la siguiente. (Como probablemente ya hayamos pensado, aquí también necesitaremos usar el índice "currentQuestionIndex" para cambiar las imágenes).

Para empezar, creamos la lista "PictureList" y la inicializamos a los nombres de archivo de las imágenes. Debemos asegurarnos que los nombres son exactamente los mismos que los nombres de los archivos que cargamos en la sección "Media" del proyecto. También es importante que el orden en el que escribimos los nombres de archivo de las imágenes en la lista de imágenes esté en sintonía con el orden de las respectivas preguntas asociadas en la lista de preguntas. La figura muestra el código necesario para definir correctamente esta lista:



Ahora vamos a modificar el código del manejador de evento "when (NextButton).Click" para que cambie la imagen que aparece en la pantalla asociada a cada pregunta. La propiedad "Picture" del componente `Image` nos permitirá determinar qué imagen vamos a mostrar como acompañante de la pregunta actual. La figura muestra el programa para implementar este comportamiento:

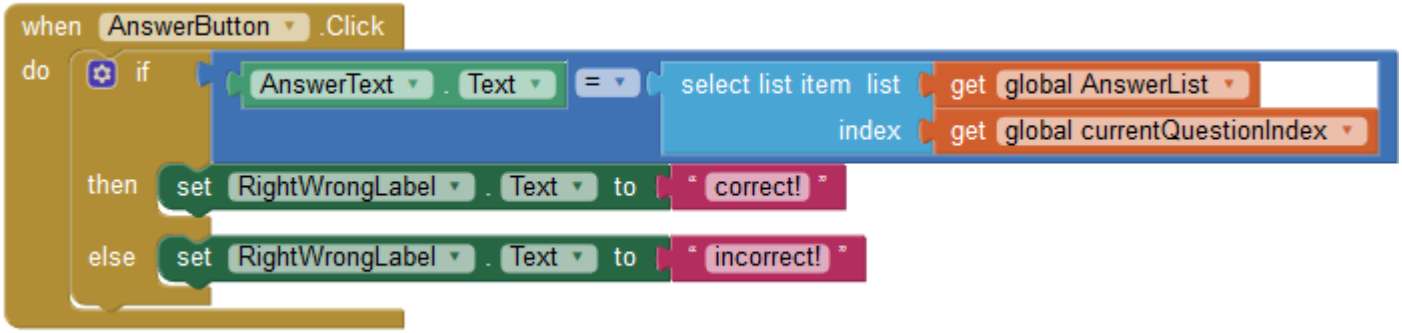


¿Cómo funciona este programa? La variable "currentQuestionIndex" actúa como índice tanto para la lista "QuestionList" como para la lista "PictureList". Si hemos definido nuestras listas adecuadamente, de forma que la primera pregunta se corresponda con la primera imagen, etc., este índice no servirá para trabajar con ambas listas. Por ejemplo, la primer imagen, *AmazonRiver.jpg*, es la figura asociada a la primera pregunta, "What is the biggest river in the world by discharge volume of water?", cuya respuesta es "Amazon".

Comprobamos el funcionamiento de la app: Clicamos en el botón "NextButton" unas cuantas veces, y comprobamos que al cambiar la pregunta, la imagen acompañante cambia de forma correspondiente a la pregunta mostrada.

COMPROBAR LAS RESPUESTAS DEL USUARIO.

Hasta ahora hemos creado una app que simplemente se desplaza a través de una serie de preguntas y respuestas (y de ciertas imágenes asociadas a las preguntas). Lo siguiente que debemos hacer es conseguir que la app indique si la respuesta proporcionada por el usuario es correcta. Vamos a escribir el programa que implementa este comportamiento. Ya hemos preparado la interfaz para que el usuario pueda escribir su respuesta en una caja de texto "AnswerText", y comprobar su validez al presionar un botón "AnswerButton". La app debe comparar la respuesta escrita por el usuario con la respuesta correcta a la pregunta actual, usando un bloque "if - else" para hacer esta comprobación. Entonces, la etiqueta "RightWrongLabel" debería cambiar para indicar si la respuesta del usuario es correcta o incorrecta. La figura muestra el código necesario para programar este comportamiento:

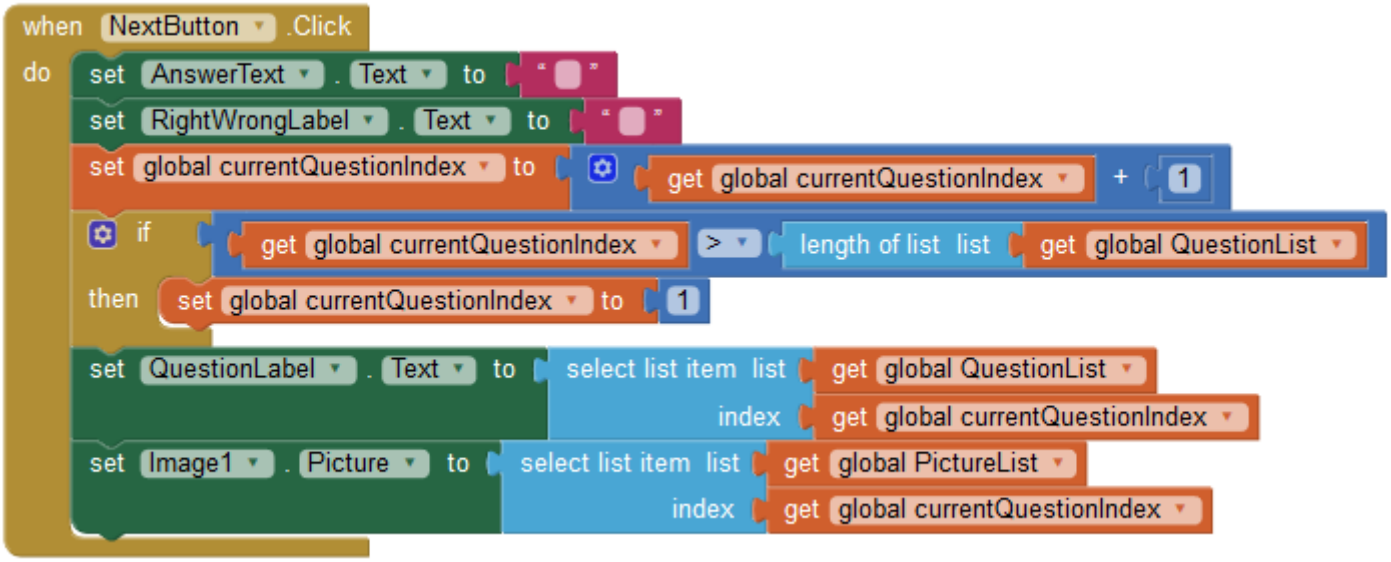


¿Cómo funciona este programa? En la figura vemos que el bloque "if - else" comprueba si la respuesta proporcionada por el usuario ("AnswerText.Text") es igual al elemento "currentQuestionIndex" de la lista "AnswerList". Por ejemplo, si "currentQuestionIndex" es 1, la app comparará la respuesta escrita por el usuario en la caja de texto "AnswerText" con el primer elemento de la lista "AnswerList", que es "Amazon". Si "currentQuestionIndex" es 2, la app comparará la respuesta del usuario con el segundo elemento de la lista "AnswerList", que es "New York". Y así sucesivamente. Si el resultado de esta comprobación es positivo, se ejecutan los bloques dentro de la rama "then" y fijamos la propiedad "Text" de la etiqueta "RightWrongLabel" a "correct!". Si el resultado de la comprobación es falso, se ejecutan los bloques dento de la rama "else", y fijamos la propiedad "Text" de "RightWrongLabel" a "incorrect!".

Ahora probamos la app: Intentamos responder a una de las preguntas. La app debería indicar si nuestra respuesta es correcta, en el sentido de que debe ser exactamente igual a la respuesta especificada en la lista "AnswerList". Probamos a responder de forma correcta e incorrecta. Notar que, para que una respuesta sea dada por buena, debe ser idéntica a la respuesta presente en la lista de respuestas. (Por ejemplo, si nuestra respuesta a la primera pregunta es "Amazonas", la app concluirá que es incorrecta, porque la respuesta correcta debe ser "Amazon", textualmente. La respuesta incluso debe escribirse respetando las letras mayúsculas y minúsculas: Por ejemplo, si la respuesta del usuario es "amazon", la app daría esa respuesta como incorrecta).

Tal y como está, la app debería funcionar. Pero puede que hayamos notado que al pasar de una pregunta a la siguiente, el texto "correct!" o "incorrect!" y la respuesta del usuario a la pregunta previa siguen estando allí, como vemos en la figura. Esto no es un problema importante, pero probablemente los usuarios de nuestra app se quejarán de este hecho.

Para programar la limpieza de la etiqueta "RightWrongLabel" y de la caja de texto "AnswerText", escribimos el código mostrado en la figura:



Como vemos en la figura, al clicar en el botón "NextButton", el usuario se mueve a la siguiente pregunta, y las dos primeras filas del manejador de eventos ponen en blanco los contenidos "RightWrongLabel" y de "AnswerText".

Vamos a probar la app. Respondemos a una pregunta y presionamos en el botón "AnswerButton". A continuación presionamos en el botón "NextButton". La respuesta del usuario a la pregunta previa y la calificación de la respuesta deben desaparecer.

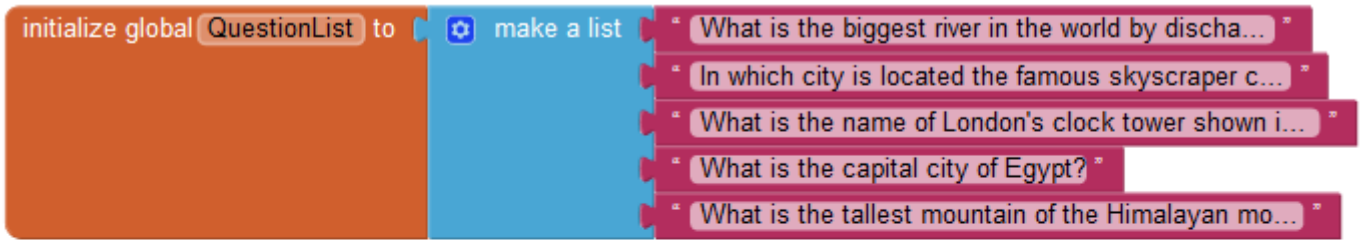
9.13. MODIFICACIONES A "GEOGRAPHYQUIZ".

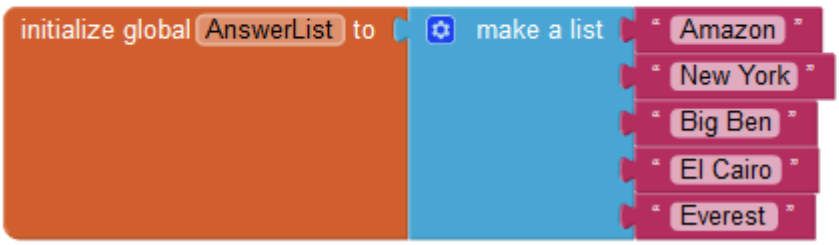
Cuando consigamos que esta app esté plenamente operativa, tal vez queramos probar algunas de las siguientes variaciones:

- En vez de mostrar una simple imagen para cada pregunta, podemos intentar reproducir un clip de audio o un video. Si decidimos reproducir sonidos, podríamos convertir nuestra app en un juego para intentar adivinar el nombre y/o autor de una pieza de música clásica, etc.
- Actualmente, nuestra app es muy rígida en cuanto a qué respuestas acepta como válidas. Hay varias formas de mejorar este funcionamiento, utilizando los bloques de procesado de texto disponibles en la bandeja "Text". Por ejemplo, el bloque "upcase" permite convertir a mayúsculas la respuesta del usuario y la respuesta oficial antes de compararlas. Otro ejemplo es usar el bloque "contains" para ver si la respuesta del usuario está contenida en la respuesta oficial. Una última opción es proporcionar varias opciones válidas para la respuesta correcta a cada pregunta, e iterar (bloque "foreach") a través de ellas para ver si la respuesta del usuario coincide con alguna de ellas.
- Otra posibilidad para mejorar la comprobación de la respuesta del usuario es transformar el cuestionario en un cuestionario de selección entre varias respuestas posibles, donde solo una de ellas es la correcta. Para ello necesitaremos una lista adicional, la lista de respuestas posibles. Esta lista será en realidad una lista de listas, donde cada sublista alojará las respuestas posibles para cada pregunta. Podemos usar un componente "ListPicker" para permitirle al usuario elegir una de las respuestas posibles a cada pregunta.

9.14. LISTAS DE LISTAS.

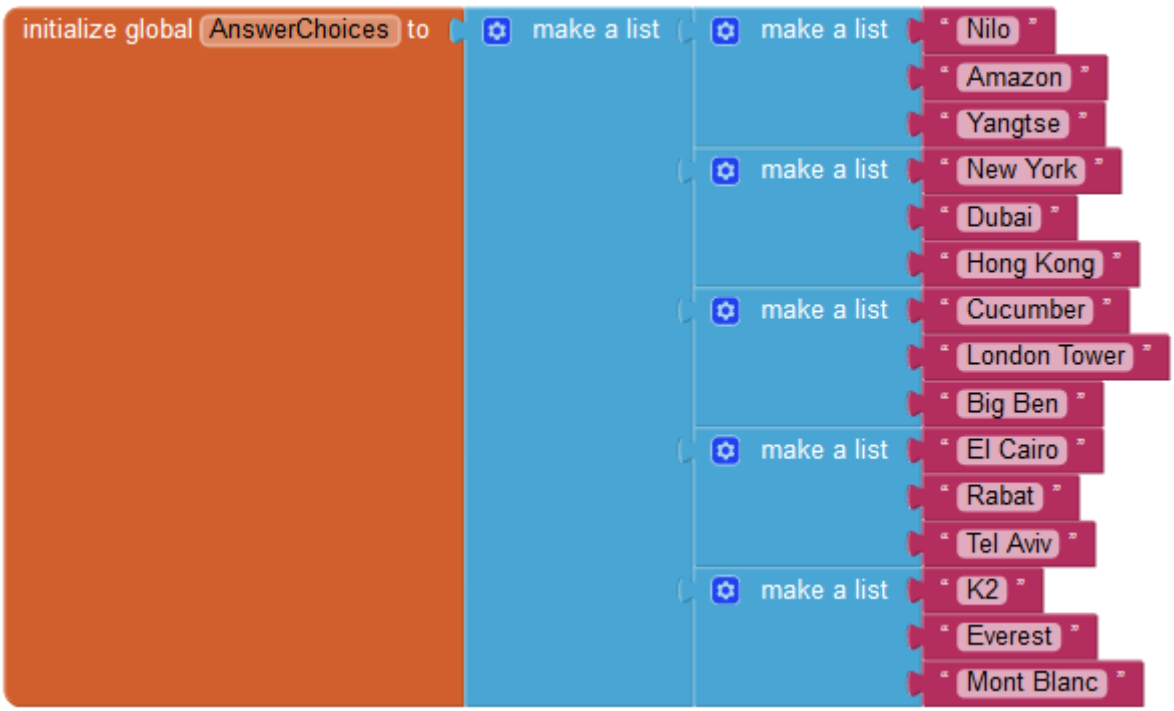
Los elementos contenidos en una lista pueden ser de cualquier tipo, incluyendo números, textos, o valores Booleanos (true/false). Pero los elementos de una lista también pueden ser otras listas. Este tipo de estructuras de datos complejas. Por ejemplo, podríamos usar una lista de listas para convertir la app "GeographyQuiz" en un cuestionario de respuestas múltiples. Vamos a echar un nuevo vistazo a la estructura básica de la app, que se basa en una lista de preguntas y en una lista de respuestas, como muestra la figura:





Cada vez que el usuario responde a una pregunta, la app comprueba si es correcta comparando la respuesta con el elemento actual de la lista "AnswerList".

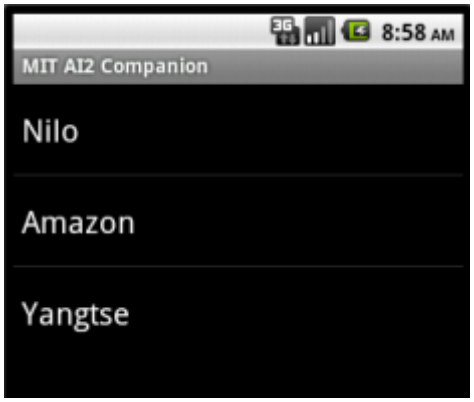
Para convertir la app en un cuestionario de respuesta múltiple necesitamos mantener una lista adicional, que almacene las distintas opciones para la respuesta a cada pregunta. Este tipo de estructura de datos se especifica poniendo cinco bloques "make a list" dentro de un bloque "make a list" interno, como muestra la figura:



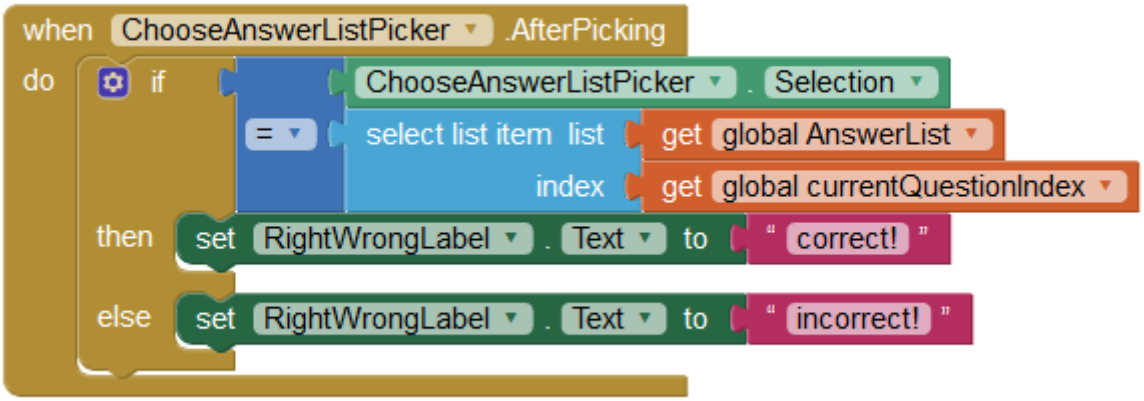
Cada elemento de la variable "AnswerChoices" es en sí mismo un alista que contiene tres elementos. Si seleccionamos un elemento de "AnswerChoices", el resultado es una lista. Ahora que hemos rellenado nuestra lista de respuestas múltiples, ¿cómo se la mostramos al usuario?

Como en la app "NotesTaker", podríamos usar un componente "ListPicker" para presentar las distintas opciones al usuario. Si el índice se llamase "currentQuestionIndex", el manejador de evento "when (ChooseAnswerListPicker).BeforePicking" se parecería al de la figura:

Estos bloques toman la sublista actual de "AnswerChoices" y le permiten al usuario elegir uno de sus elementos. Así, si el valor de "currentQuestionIndex" fuese 1, el componente "ListPicker" mostraría una lista como la de la figura:



Cuando el usuario elige, comprobamos si la respuesta elegida es la correcta con los bloques de la siguiente figura:



En estos bloques, la selección hecha por el usuario mediante el "ListPicker" se compara con la respuesta correcta, que se almacena en una lista independiente, "AnswerList" (porque "AnswerChoices" solo proporciona las opciones de respuesta, pero no la respuesta correcta) .

9.15. COMENZAR CON LA APP "XYLOPHONE".

Hasta ahora hemos estado trabajando con listas estáticas, en las que los elementos de las listas estaban definidos por el programador desde el principio, y no cambiaban. En las próximas secciones vamos a crear una app para un xilófono que requerirá usar listas dinámicas.

El xilófono que programaremos funcionará de forma que nos permitirá:

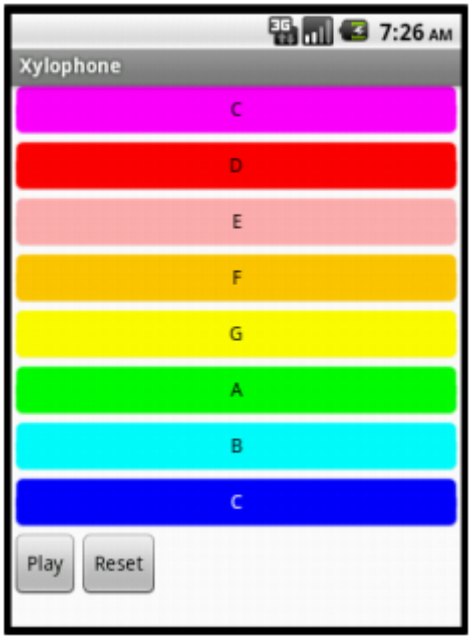
- Tocar ocho notas distintas presionando botones de distinto color en pantalla.
- Pinchar en un botón de "Play" para reproducir las notas que tocamos antes.

- Presionar en un botón de "Reset" para borrar las notas que hayamos tocado antes, de forma que podamos grabar una nueva canción.

Para implementar todos estos comportamientos necesitaremos aprender a:

- Usar un único componente "Sound" para reproducir diferentes archivos de audio.
- Usar el componente "Clock" para medir y aplicar retardos entre diferentes acciones.
- Crear un **procedimiento recursivo** que se llame a sí mismo.
- Usar listas dinámicas, incluyendo la inserción de nuevos elementos, el acceso a los elementos, y el borrado de listas.

Para empezar nos conectamos a la web de App Inventor y comenzamos un nuevo proyecto al que llamaremos "Xylophone". Fijamos el título de la pantalla a "Xylophone", y conectamos la app a nuestro dispositivo Android o al emulador.



9.16. DISEÑAR Y PROGRAMAR LOS COMPONENTES DEL TECLADO.

Esta app consta de los 13 componentes listados en la tabla (8 de los cuales comprenden el teclado). Como hay tantos componentes, sería un poco aburrido crearlos todos antes de empezar a programarlos, así que vamos a dividir la app en sus partes funcionales y a construirlas secuencialmente mientras cambiamos entre Diseñador y el Editor de Bloques, tal y como hicimos con la app "LadybugChase".

Nuestra interfaz de usuario incluirá un teclado de ocho notas para una escala pentatónica mayor de siete notas que vaya desde la nota C Baja hasta la nota C Alta. En esta sección comenzaremos creando este teclado musical.

CREAR LOS DOS PRIMEROS BOTONES DE NOTAS.

Comenzamos creando las dos primeras teclas del xilófono, que implementaremos como botones.

1) En la vista de Diseñador, y desde la bandeja "User Interface", añadimos un componente "Button" a la pantalla. Mantenemos el nombre de "Button1". Queremos que sea una barra larga de color magenta, igual que la de un xilófono de verdad, así que fijamos sus propiedades como indicamos:
   - Cambiamos su propiedad "BackgroundColor" a Magenta.
   - Cambiamos su propiedad "Text" a "C".
   - Fijamos su propiedad "Width" a "Fill parent", para que ocupe todo el ancho de la pantalla.

- Fijamos su propiedad "Height" a 40 píxeles.

2) Repetimos el proceso para un segundo botón, llamado "Button2", ubicándolo debajo de "Button1". Usamos los mismos valores que antes para sus propiedades "Width" y "Height", pero fijamos su propiedad "BackgroundColor" a Rojo, y su propiedad Text a "D".

Component type	Palette group	What you'll name it	Purpose
Button	User Interface	Button1	Play Low C key.
Button	User Interface	Button2	Play D key.
Button	User Interface	Button3	Play E key.
Button	User Interface	Button4	Play F key.
Button	User Interface	Button5	Play G key.
Button	User Interface	Button6	Play A key.
Button	User Interface	Button7	Play B key.
Button	User Interface	Button8	Play High C key.
Sound	Media	Sound1	Play the notes.
Button	User Interface	PlayButton	Play back the song.
Button	User Interface	ResetButton	Reset the song memory.
HorizontalArrangement	Layout	HorizontalArrangement1	Place the Play and Reset buttons next to each other.
Clock	User Interface	Clock1	Keep track of delays between notes.

Más adelante repetiremos el paso 2 seis veces más para crear los otros seis botones.

AÑADIR EL COMPONENTE SOUND.

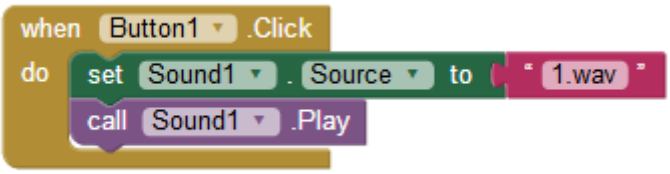
No se puede tener un xilófono sin sonidos, así que vamos a añadir un componente "Sound", manteniendo su nombre como "Sound1". Cambiamos su propiedad "MinimumInterval" de su valor por defecto de 500 milisegundos a 0. Esto nos permitirá reproducir sonidos con tanta frecuencia como queramos, en vez de tener que esperar medio segundo (500 miliseguntos) entre dos sonidos consecutivos. Aquí no vamos a fijar su propiedad "Source", porque lo haremos en el Editor de Bloques.

En la carpeta de archivos para los alumnos encontraremos los archivos de sonidos necesarios para las dos primeras teclas, a saber, *1.wav* y *2.wav*. Carga estos archivos en el proyecto, en la sección "Media" del Diseñador. Al contrario que en los proyectos previos, donde no había problema en cambiar los nombres de los archivos multimedia, en este caso es importante que usemos estos nombres exactos, por razones que quedarán claras en su momento. Más adelante cargaremos los otros seis archivos de sonido.

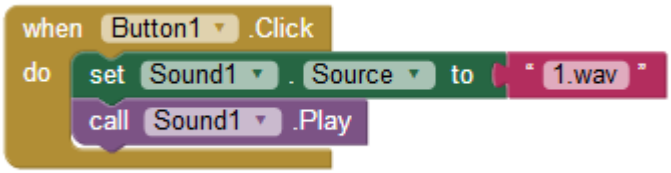
ASOCIAR LOS SONIDOS A LOS BOTONES.

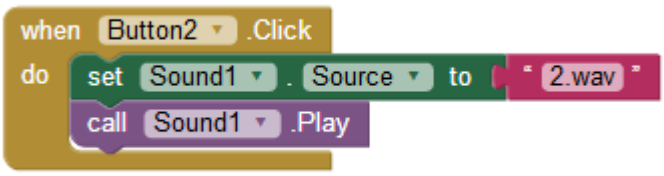
El comportamiento que debemos programar es que se reproduzca un sonido cuando se presione el botón correspondiente. Específicamente, si presionamos el botón "Button1", queremos que se reproduzca el archivo *1.wav*; si presionamos el botón "Button2", queremos reproducir el archivo *2.wav*, y así sucesivamente. Este comportamiento podemos especificarlo en el Editor de Bloques haciendo lo siguiente:

1) De la bandeja "Button1", sacamos un bloque "when (Button1).Click".
2) De la bandeja "Sound1", sacamos un bloque "set (Sound1).(Source) to", y lo ubicamos dentro de la sección "do" del manejador "when (Button1).Click".
3) Escribimos "text" para crear un bloque de texto. (Este método es más rápido que acudir a la bandeja "Text" de los bloques integrados, y funciona igual de bien). Dentro del bloque de texto, escribimos "1.wav", y conectamos este bloque a la ranura de parámetros del bloque "set (Sound1).(Source) to".
4) Añadimos un bloque "call (Sound1).Play"



Podríamos hacer exactamente lo mismo para el botón "Button2", pero el código sería terriblemente repetitivo:





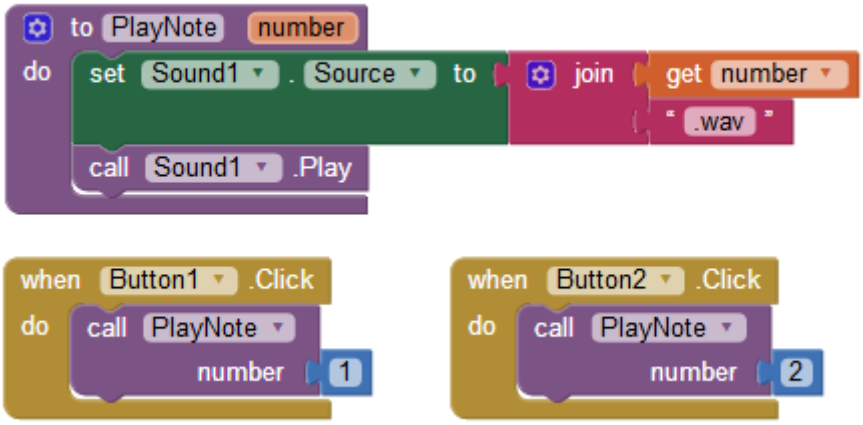
El código repetitivo es una señal inequívoca de que deberíamos crear un procedimiento para implementarlo, tal y como hicimos en las apps "MoleMash" y "LadybugChase" en capítulos previos. En particular, vamos a crear un procedimiento que reciba un número como parámetro, fije la propiedad "Source" de "Sound1" al archivo apropiado, y reproduzca ese sonido. Podemos usar el bloque "join" para combinar el número (por ejemplo, 1) y el texto ".wav" para crear el nombre de archivo apropiado (por ejemplo, "1.wav"). He aquí los pasos para crear el procedimiento que necesitamos:

1) En la parte de bloques integrados (Built-in), vamos a la bandeja "Procedures" y sacamos un bloque "to (procedure) do".
2) Añadimos un parámetro al procedimiento clicando en el icono azul del engranaje y arrastrando un bloque "input". Cambiamos el nombre del parámetro de "x" (su nombre por defecto) a "number".
3) Pinchamos en el nombre del procedimiento, que es "procedure" por defecto, y lo cambiamos a "PlayNote".
4) Tomamos un bloque "set (Sound1).(Source) to" y lo añadimos dentro del procedimiento "PlayNote", a la derecha de la palabra "do". Añadimos también un bloque "call (Sound1).Play" bajo el bloque "set (Sound1).(Source) to".
5) De la bandeja "Text", tomamos un bloque "join" y lo conectamos a la ranura del bloque "set (Sound1).(Source) to".
6) Pasamos el ratón sobre el parámetro "number" de "PlayNote" y tomamos un bloque "get (number)", que conectamos a la primera ranura del bloque "join".
7) De la bandeja "Text" tomamos un bloque de texto, lo rellenamos con el texto ".wav" (sin comillas), y conectamos este bloque a la segunda ranura del bloque "join".
8) Vaciamos de contenido el manejador de evento "when (Button1).Click".

9) En la ahora vacía sección "do" del manejador "when (Button1).Click", añadimos una llamada al procedimiento "PlayNote".
10) Escribimos un "1" para obtener un bloque numérico con ese valor, y lo conectamos a la ranura de la llamada al procedimiento "PlayNote".

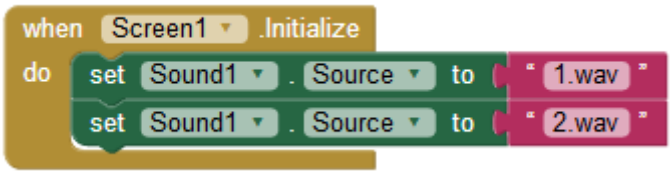
Ahora, al pulsar en "Button1", el manejador de este evento llamará al procedimiento "PlayNote", pasándole como argumento el valor 1. El parámetro "number" recibe este valor, y el procedimiento fija la propiedad "Source" de `Sound1` a "1.wav" y reproduce ese sonido.

Creamos un bloque "when (Button2).Click" con una llamada a "PlayNote" pasándole al parámetro "number" el valor 2. (Para ir más rápidos podemos copiar el código para el manejador "when (Button1).Click", y entonces cambiar "Button1" por "Button2", y el valor del parámetro "number" a 2). Nuestro programa debería parecerse al de la figura:



ORDENAR A ANDROID QUE CARGUE LOS SONIDOS.

Si hemos probado la app tal y como está nos habrá sorprendido no escuchar el sonido correspondiente, o escucharlo con un retardo inesperado, o incluso obtener un mensaje de error. Esto ocurre porque Android necesita cargar los sonidos que va a reproducir durante la ejecución de la app, lo que implica un retardo antes de que se reproduzcan. Este problema no lo hemos tenido antes porque los nombres de archivo ubicados en la propiedad "Source" de un componente "Sound" a través del *Diseñador* se cargan automáticamente al arrancar el programa. Pero en esta app no fijamos la propiedad "Source" de "Sound1" hasta después de que el programa ya haya empezado, este proceso de inicialización nunca tiene lugar. Por ello, debemos cargar explícitamente los sonidos nada más arrancar el programa. Ésa es precisamente la funcionalidad que implementa el código de la figura:



Probamos nuestra app: Tocamos los botones y comprobamos que las notas se reproducen sin retardos ni errores. (Si no escuchamos nada, nos aseguramos que el volumen de multimedia del teléfono no está silenciado).

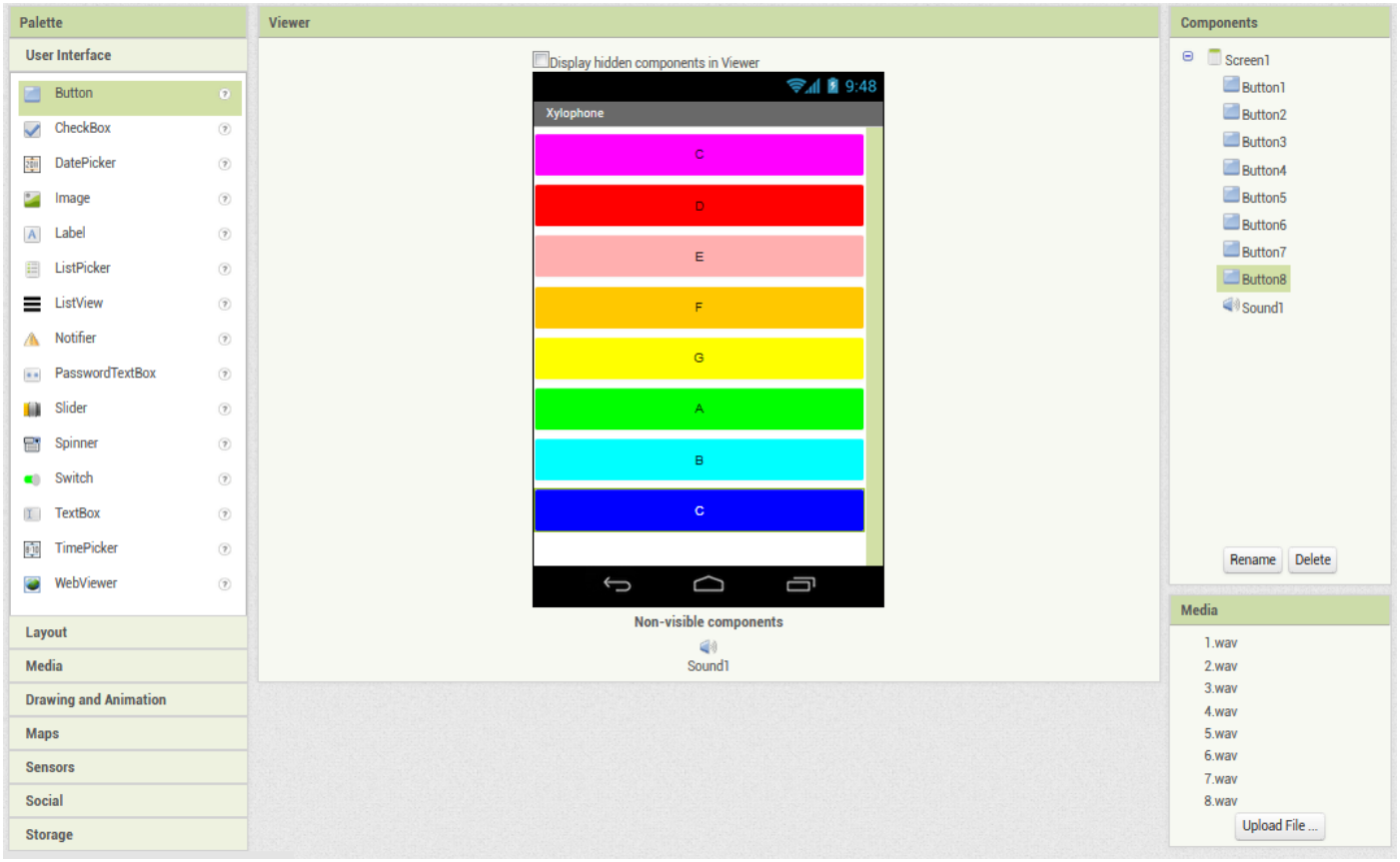
IMPLEMENTAR EL RESTO DE NOTAS MUSICALES.

Ahora que ya tenemos funcionando los dos primeros botones con sus correspondientes notas, vamos a añadir las otras seis notas restantes. Primero, acudimos al Diseñador y cargamos el resto de archivos de audio (desde *3.wav* hasta *8.wav*). A continuación creamos seis nuevos botones siguiendo los mismos pasos

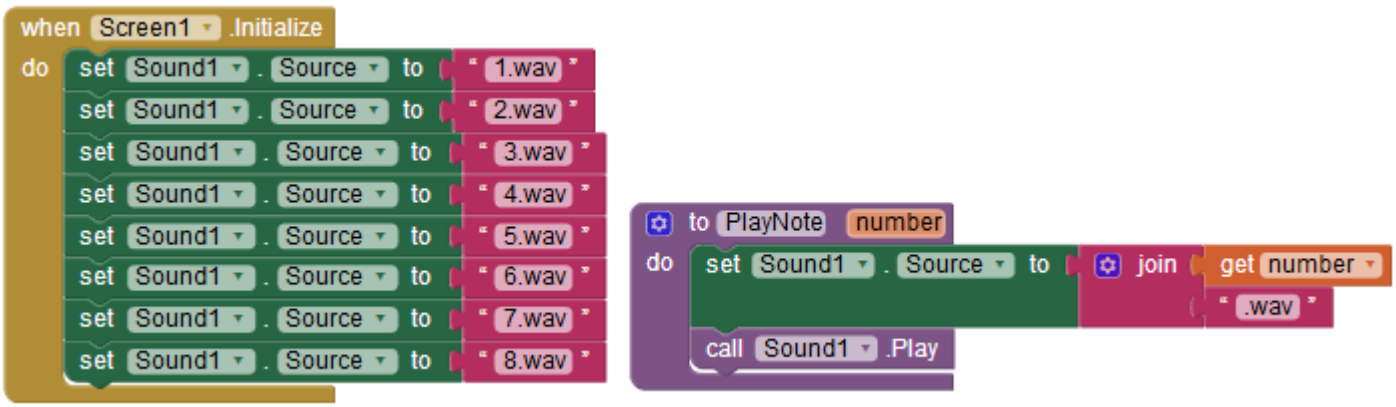
que para los dos botones previos, pero fijando sus propiedades "Text" y "BackgroundColor" a los valores aquí indicados:

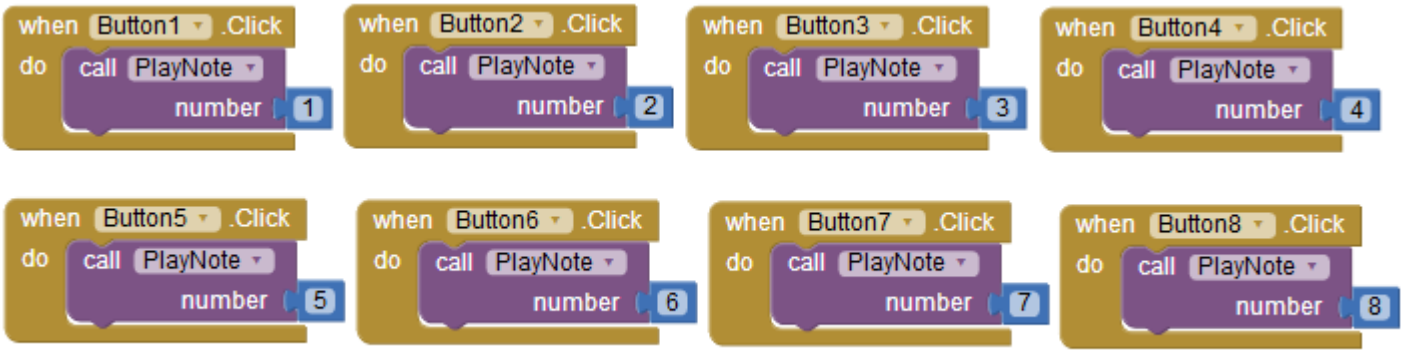
- "Button3": texto "E" y color Rosa.
- "Button4": texto "F" y color Naranja.
- "Button5": texto "G" y color Amarillo.
- "Button6": texto "A" y color Verde.
- "Button7": texto "B" y color Cian.
- "Button8": texto "C" y color Azul.

Puede que también nos convenga cambiar la propiedad "TextColor" de "Button8" a Blanco, para que el texto del botón sea más legible.



De vuelta al Editor de Bloques, creamos seis manejadores de evento "when (Button).Click" para cada uno de los nuevos botones, con la llamada apropiada al procedimiento "PlayNote". De forma similar, añadimos cada nuevo archivo de texto al manejador "when (Screen1).Initialize". El código debería quedarnos como muestra la figura:





Al probar la app, deberíamos tener todos los botones, y cada uno de ellos debería reproducir una nota distinta al pulsarlo.

9.17. GRABAR LAS NOTAS Y VOLVER A REPRODUCIRLAS.

Hacer que suenen notas musicales al presionar los botones está muy bien, pero es incluso mejor ser capaz de grabar canciones para volver a reproducirlas. Para implementar esta funcionalidad necesitamos mantener un registro de las notas musicales reproducidas. Además de recordar los tonos (esto es, los archivos de audio) que se tocaron, también debemos registrar la cantidad de tiempo transcurrido entre dos notas consecutivas. De otro modo, no seremos capaces de distinguir dos notas reproducidas en una sucesión rápida y dos notas reproducidas con un silencio de 10 segundos entre ambas.

Para ello, nuestra app mantendrá dos listas, cada una de las cuales tendrá una entrada para cada nota que se haya tocado. Estas listas serán:

- "notes", que contendrá los nombres de los archivos de sonido en el orden en el que se tocaron.
- "times", que registrará los instantes de tiempo en los que se tocaron esas notas.

Podemos obtener la información temporal de un componente "Clock". También usaremos este componente para temporizar adecuadamente las notas al reproducirlas tras ser grabadas.

AÑADIR LOS COMPONENTES.

En el diseñador necesitamos añadir un componente "Clock" y sendos botones de Play y de Reset, que organizaremos en un "HorizontalArrangement":

1) De la bandeja "Sensors", añadimos un componente "Clock", que aparecerá en la sección de componentes no visibles. Desactiva su propiedad "TimerEnabled" porque no queremos que su temporizador termine hasta que le digamos que lo haga tras la reproducción de una grabación.
2) De la bandeja "Layout" añadimos un componente "HorizontalArrangement" bajo el último botón de notas musicales. Fijamos su propiedad "Width" a "Fill parent".
3) De la bandeja "User Interface", añadimos un componente "Button". Lo renombramos como "PlayButton", y fijamos su propiedad "Text" a "Play".
4) Añadimos un segundo botón, que lo ubicaremos a la derecha de "PlayButton". Lo renombramos como "ResetButton", y fijamos su propiedad "Text" a "Reset".

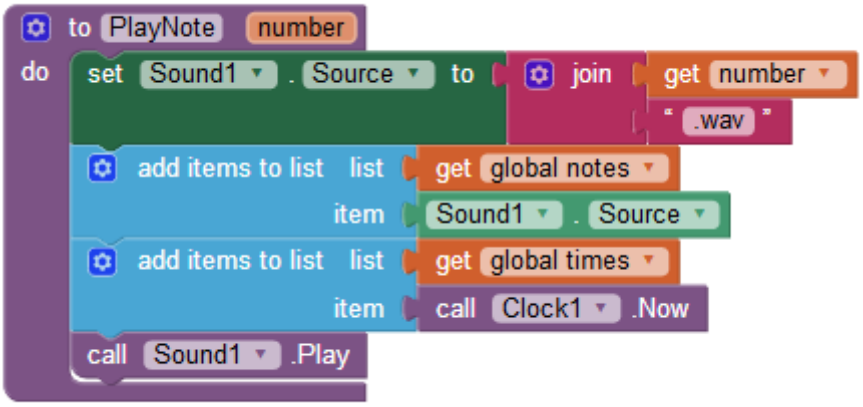
GRABAR NOTAS Y TIEMPOS.

Ahora vamos a definir el comportamiento de estos componentes en el Editor de Bloques. Queremos mantener sendas listas de notas y de tiempos, y añadir nuevos elementos a dichas listas cada vez que el usuario presiones un botón de una nota musical. Para ello:

1) Creamos una nueva variable yendo a la bandeja "Variables" y usando un bloque "initialize global (name) to".
2) Cambiamos el nombre de la variable de "name" a "notes".
3) Acudimos a la bandeja "Lists" para sacar un bloque "créate empty list", y lo conectamos a la ranura a la derecha del bloque "initialize global (notes) to"

Con esto definimos una nueva variable llamada "notes" que inicializamos a una lista vacía. Repetimos estos mismos pasos para otra variable tipo lista, a la que llamaremos "times". Estos nuevos bloques deberían ser iguales a los mostrados en la figura:

La siguiente figura muestra el código con el que debemos modificar la definición del procedimiento "PlayNote" para programar el comportamiento antes indicado: Al tocar un botón cualquiera de nota musical, debemos guardar tanto el nombre del archivo de sonido (en la lista "notes") y el instante de tiempo en el que se reprodujo ese sonido (en la lista "times"). Para registrar un instante de tiempo hacemos uso del bloque "call Clock1.Now", que devuelve el instante de tiempo actual (por ejemplo, May 20, 2019, 10:45:14 AM) con precisión de milisegundos. Estos dos valores (la nota reproducida y el instante de tiempo en el que se reprodujo), obtenidos mediante los bloques "Sound1.Source" y "call Clock1.Now", deben añadirse a las listas "notes" y "times", respectivamente.

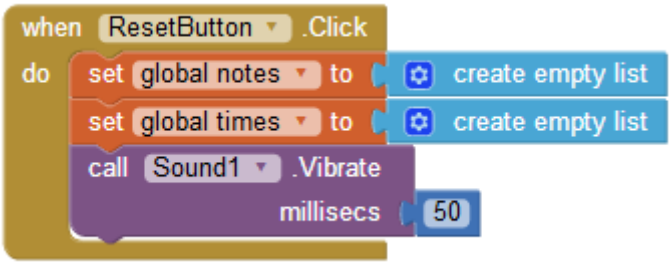


Por ejemplo, si tocamos la canción "Row, Row, Row Your Boat" (que en notas musicales es C C C D E), nuestras listas deberían terminar teniendo cinco entradas, que se parecerían a lo siguiente:

- "notes": 1.wav, 1.wav, 1.wav, 2.wav, 3.wav.
- "times" (fechas omitidas): 12:00:01, 12:00:02, 12:00:03, 12:00:03.5, 12:00:04.

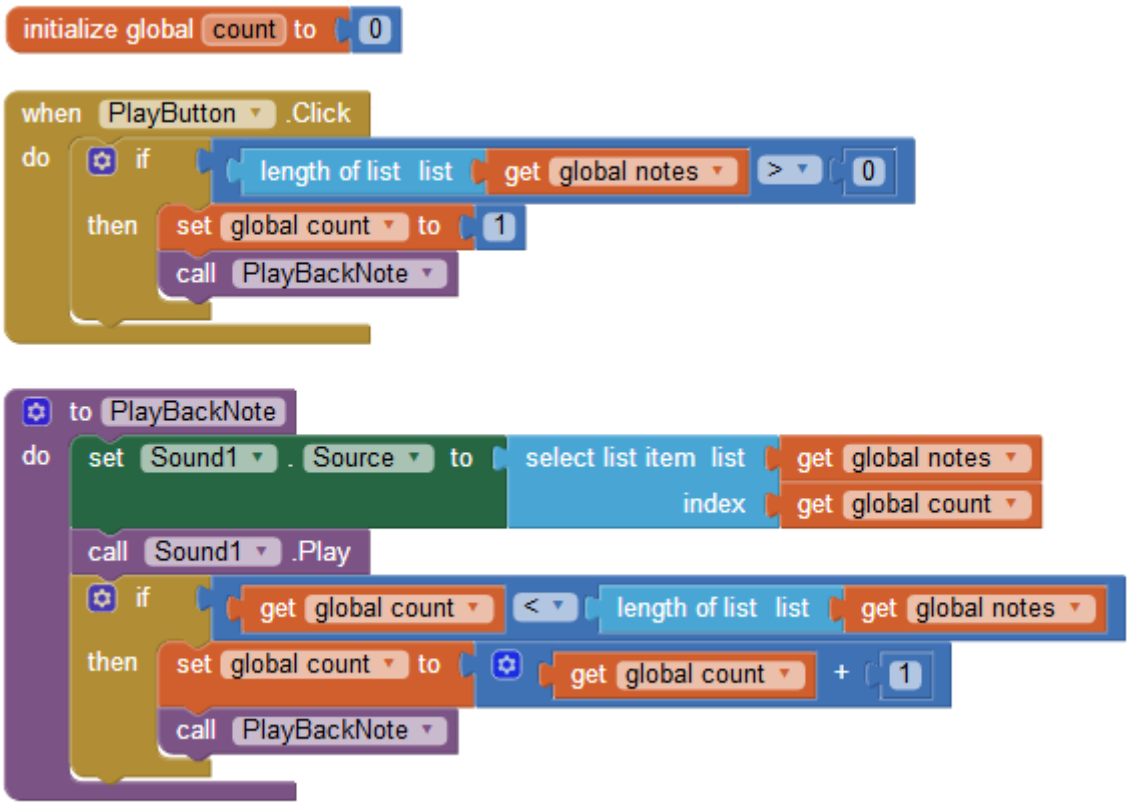
Ahora, cuando el usuario presione el botón de Reset, queremos que las dos listas retornen a sus estados originales, esto es, que vuelvan a quedar vacías. Como el usuario no percibirá cambio alguno, nos vendrá bien añadir una pequeña vibración (mediante el bloque "call Sound1.Vibrate") para indicar que se ha registrado la solicitud de reseteo.

La siguiente figura muestra el programa necesario para implementar este comportamiento:



REPRODUCIR NOTAS PREVIAMENTE GRABADAS.

A modo de experimento mental, pensemos en cómo implementar la reproducción de notas previamente grabadas sin prestar atención a los instantes de tiempo en que se reprodujeron. Esto podríamos hacerlo (aunque nosotros no lo haremos así) mediante el programa mostrado en la figura:



El método es el siguiente:

- Usamos una variable "count" para llevar la cuenta de la nota en la que estamos en la lista de notas.
- Creamos un nuevo procedimiento, llamado "PlayBackNote", que reproduzca esa nota y se mueva a la siguiente nota de la lista.
- Codificamos que al presionar el botón PlayButton la app fije el valor de "count" a 1 y llame al procedimiento "PlayBackNote", a menos que no haya notas previamente guardadas.

¿Cómo funciona el programa de la figura? Probablemente ésta sea la primera vez que vemos a un procedimiento llamándose a sí mismo. Aunque a primera vista esto pueda parecer extraño, se trata de una importante y poderosa técnica de programación llamada **recursión**.

Para hacernos una idea de cómo funciona la recursión, vamos a ver qué ocurre cuando un usuario reproduce y graba tres notas musicales (por ejemplo, 1.wav, 3.wav, y 6.wav), y a continuación, pulsa el botón "Play". En primer lugar, empieza a ejecutarse el manejador "when (PlayButton).Click". Como la longitud de la lista

"notes" es 3, lo cual es mayor que cero, el valor de la variable "count" se fija a 1, y se llama al procedimiento "PlayBackNote":

1) La primera vez que se llama a "PlayBackNote", el valor de "count" es 1, y entonces:
   - La propiedad "Sound1.Source" se fija al primer elemento de la lista "notes", que es 1.wav.
   - Se llama a la función "Sound1.Play" para reproducir esa nota.
   - Como el valor de "count" (que ahora es 1) es menor que la longitud de la lista "notes" (que es 3), "count" se incrementa a 2, y se vuelve a llamar al procedimiento "PlayBackNote".

2) La segunda vez que se llama a "PlayBackNote", "count" vale 2, y entonces:
   - La propiedad "Sound1.Source" se fija al segundo elemento de la lista "notes", que es 3.wav.
   - Se llama a "Sound1.Play" para reproducir esa nota.
   - Como el valor de "count" (que ahora es 2) es menor que la longitud de la lista "notes" (que es 3), "count" se incrementa a 3, y se vuelve a llamar al procedimiento "PlayBackNote".

3) La tercera vez que se llama a "PlayBackNote", "count" vale 3, y entonces:
   - La propiedad "Sound1.Source" se fija al tercer elemento de la lista "notes", que es 6.wav.
   - Se llama a "Sound1.Play" para reproducir esa nota.
   - Como el valor de "count" (que ahora es 3) ya no es menor que la longitud de la lista "notes" (que es 3), ya no se incrementa el valor de "count" no se vuelve a llamar a "PlayBackNote", y el proceso de reproducción de notas grabadas finaliza aquí.

NOTA: Aunque la recursión es muy útil, también es potencialmente peligrosa. A modo de experimento mental, preguntémonos qué ocurriría si el programador se olvidase de insertar en "PlayBackNote" los bloques que se encargan de incrementar el valor de "count".

Aunque la recursión que hemos usado aquí es correcta, el programa previo tiene un problema evidente: Como no hay tiempo de retardo entre una llamada a "Sound1.Play" y la siguiente, lo que implica que la reproducción de cada nota se verá interrumpida por la reproducción de la siguiente (excepto en el caso de la última nota). De esta forma no estamos permitiendo que la reproducción de una cierta nota se complete antes de cambiar el archivo de sonido de Sound1 y de reproducir la siguiente nota. Para conseguir el comportamiento correcto, debemos añadir un retardo entre dos llamadas sucesivas al procedimiento "PlayBackNote".

REPRODUCIR NOTAS GRABADAS CON LOS RETARDOS ADECUADOS.

Para implementar el retardo necesario vamos a ajustar el temporizador del reloj "Clock1" a una cantidad igual a la diferencia de tiempos entre la nota actual y la siguiente. Por ejemplo, si nota siguiente se reprodujo 3000 milisegundos (3 segundos) después de que se tocase la nota actual, fijaremos la propiedad "Clock1.TimerInterval" a 3000 antes de volver a llamar al procedimiento "PlayBackNote".

Para ello, cambiamos el bloque "if" del procedimiento "PlayBackNote" como indica la figura, y creamos y completamos el manejador de evento "when Clock1.Timer" (el cual especifica qué debería ocurrir cuando termina el temporizador).

¿Cómo funciona este código? Digamos que tenemos los siguientes contenidos en las dos listas:

- "notes": 1.wav, 3.wav, 6.wav.
- "times": 12:00:00, 12:00:01, 12:00:03.

Como muestra la figura, el manejador "when (PlayButton).Click" fija el valor de "count" a 1 y llama a "PlayBackNote".

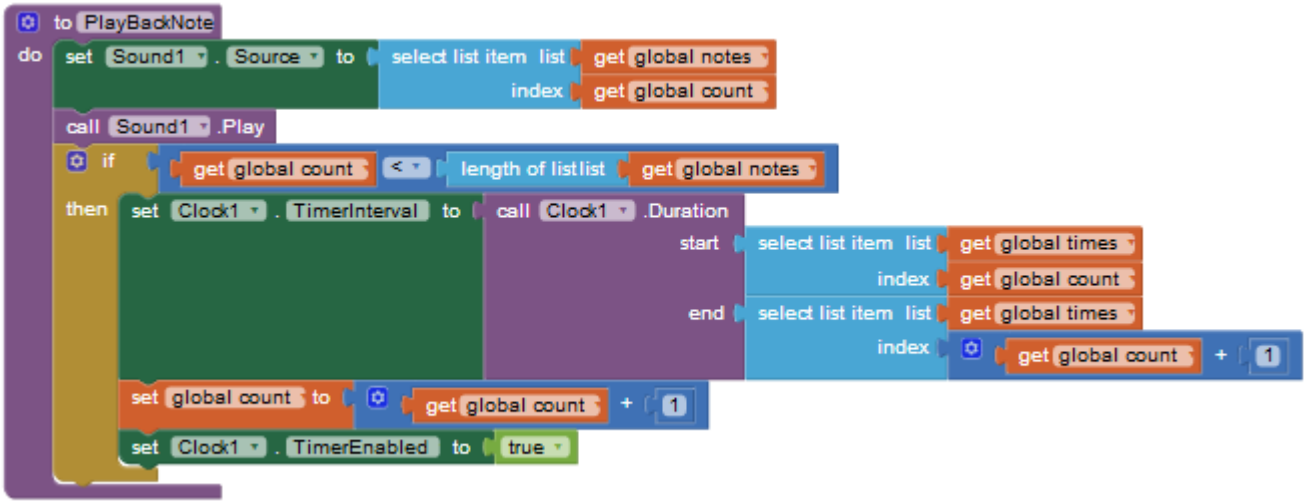
1) La primera vez que se llama a "PlayBackNote", "count" es igual a 1, y entonces:
   - La propiedad "Sound1.Source" se fija al primer elemento de la lista "notes", que es 1.wav.
   - Se llama a la función "Sound1.Play" para reproducir esa nota.
   - Como el valor de "count" (que ahora es 1) es menor que la longitud de la lista "notes" (que es 3), la propiedad "Clock1.TimerInterval" se fija a la cantidad de tiempo entre el primer elemento (12:00:00) y el segundo elemento (12:00:01) de la lista "times", lo que es igual a 1 segundo. Además, "count" se incrementa a 2, y se habilita "Clock1.Timer" para iniciar una cuenta atrás de 1 segundo.

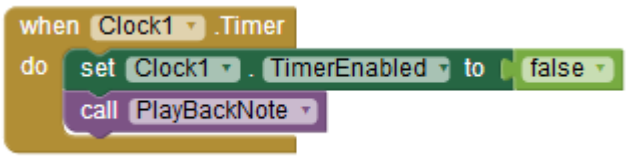
     Durante esta cuenta atrás de 1 segundo no ocurre nada más, hasta que finaliza la temporización y se ejecuta el manejador "when Clock1.Timer", el cual deshabilita temporalmente el temporizador y llama al procedimiento "PlayBackNote".

2) La segunda vez que se llama a "PlayBackNote", "count" es igual a 2, y entonces:
   - La propiedad "Sound1.Source" se fija al segundo elemento de la lista "notes", que es 3.wav.
   - Se llama a la función "Sound1.Play" para reproducir esa nota.
   - Como el valor de "count" (que ahora es 2) es menor que la longitud de la lista "notes" (que es 3), la propiedad "Clock1.TimerInterval" se fija a la cantidad de tiempo entre el segundo elemento (12:00:01) y el tercer elemento (12:00:04) de la lista "times", lo que es igual a 3 segundos. Además, "count" se incrementa a 3, y se habilita "Clock1.Timer" para iniciar una cuenta atrás de 3 segundos.

     Durante esta cuenta atrás de 3 segundos no ocurre nada más, hasta que finaliza la temporización y se ejecuta el manejador "when Clock1.Timer", el cual deshabilita temporalmente el temporizador y llama al procedimiento "PlayBackNote".

3) La tercera vez que se llama a "PlayBackNote", "count" es igual a 3, y entonces:
   - La propiedad "Sound1.Source" se fija al tercer elemento de la lista "notes", que es 6.wav.
   - Se llama a la función "Sound1.Play" para reproducir esa nota.
   - Como el valor de "count" (que ahora es 3) ya no es menor que la longitud de la lista "notes" (que es 3), no ocurre nada más, y la reproducción de las notas grabadas ha finalizado.





9.18. MODIFICACIONES A "XYLOPHONE".

- Ahora mismo no hay nada que le impida al usuario presionar el botón de Reset durante la reproducción de unas notas grabadas, lo que hace que el programa falle. (¿Puedes ver por qué?). Modifica el

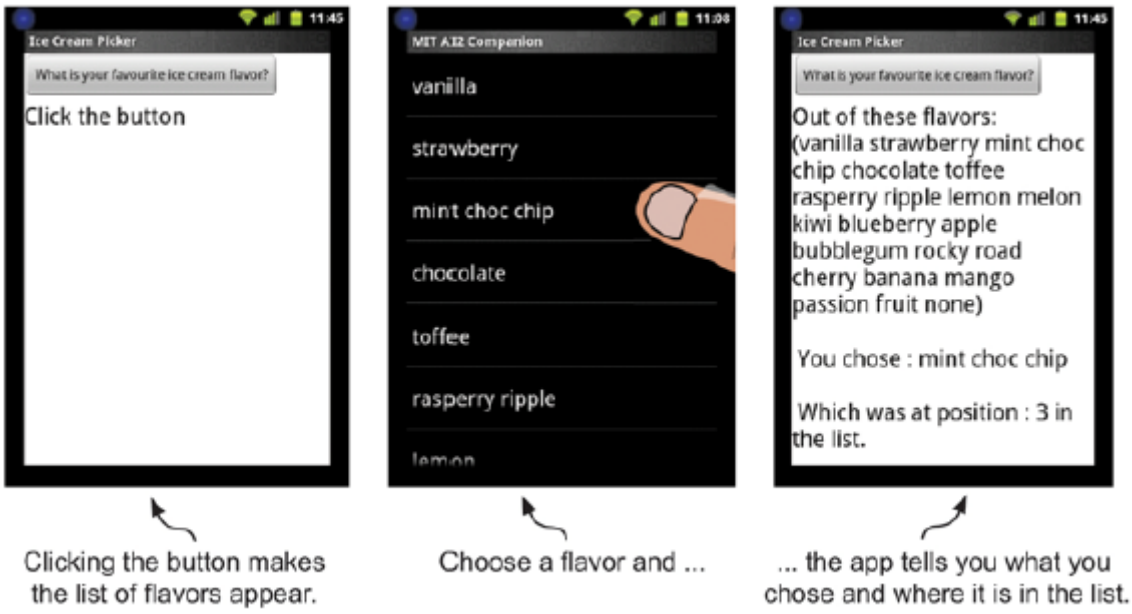
manejador "when (PlayButton).Click" para deshabilitar el botón "ResetButton". Para rehabilitar este botón cuando la canción grabada se haya reproducido completamente, cambia el bloque "if" del manejador "when (PlayButton).Click" para convertirlo en un bloque "if - else", y activa el botón de Reset en la sección "else".

- De forma similar, ahora mismo el usuario puede pulsar el botón "PlayButton" cuando ya se está reproduciendo una canción. (¿Puedes ver qué ocurriría en ese caso?). Haz que el manejador "when (PlayButton).Click" deshabilite el botón "PlayButton" y cambie su texto a "Playing…". Puedes volver a activar este botón y resetear su texto con un bloque "if - else", como en la actividad previa.
- Añade un botón con el nombre de una canción, como por ejemplo, "Für Elise". Si el usuario lo presiona, la app debe rellenar la lista de notas y de tiempos con los valores correspondientes, fijar el valor de "count" a 1, y llamar a "PlayBackNote". Para fijar los tiempos apropiados nos resultará útil el bloque "call Clock1.MakeInstantFromMillis".
- Si el usuario presiona una nota, se va a hacer algo, y vuelve horas más tarde y presiona una tecla adicional, las notas formarán parte de la misma canción. Pero es probable que no fuera esa la intención del usuario. Mejora el programa implementando una de las siguientes funcionalidades:
  a) Para la grabación después de que haya pasado un intervalo de tiempo razonable (digamos, 1 minuto), o bien…
  b) Pon un límite a la cantidad de tiempo usada por "Clock1.TimerInterval", usando el bloque "max" de la bandeja "Math".
- Indica visualmente qué nota se está tocando actualmente cambiando la apariencia del botón (por ejemplo, cambiando sus propiedades "Text", "BackgroundColor", o "ForegroundColor").

9.19. EJERCICIOS DEL CAPÍTULO 9.

Ejercicio 9.1. Helados.
Crea una app donde hayas definido una lista de sabores de helado. Cuando el usuario selecciona uno de estos sabores a través de un componente "ListPicker", la app muestra un mensaje como el siguiente:

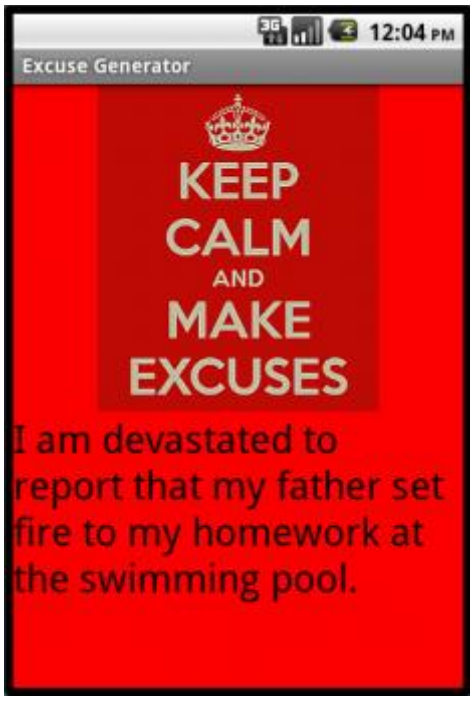
"De la siguiente lista de sabores [aquí va la lista de todos los sabores], tú has elegido [aquí va el sabor elegido], el cual estaba en la posición [aquí va la posición del sabor elegido] en la lista".



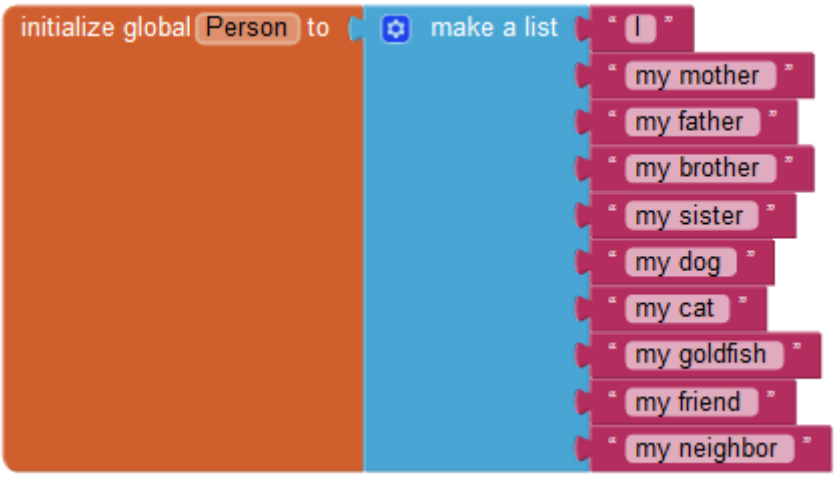
Ejercicio 9.2. Generador de excusas.
Vamos a escribir una app que nos permita generar de forma aleatoria las excusas que podemos dar cuando no hemos hecho los deberes. Para construir una excusa la app utiliza 4 listas ocultas, cada una de las cuales con 10 elementos. Hay un total de 10000 posibles combinaciones de excusas, algunas tienen sentido y otras no.

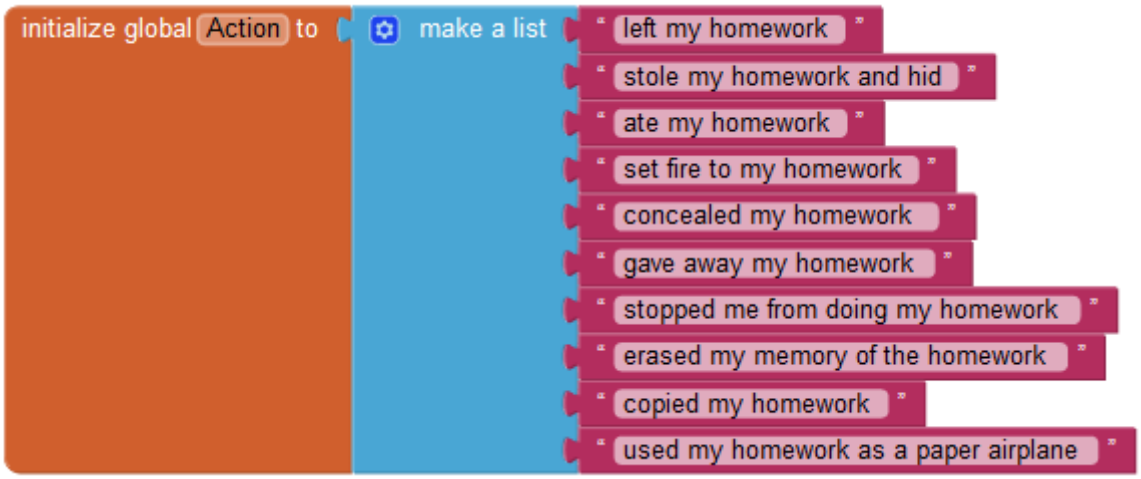
a) Usando los componentes listados en la tabla, diseña la interfaz de usuario de la app para que se parezca a la de la figura. Necesitarás la imagen *keep-calm-and-make-excuses.png*, disponible en los archivos de los alumnos.

Excuse Generator			
Screen1 properties:	`Title`: Excuse Generator `AlignHorizontal`: Center `BackgroundColor`: Red `Screen Orientation`: Portrait		
Components	**What do I rename it?**	**What does it do?**	**What properties do I set?**
`Button`	`ExcuseButton`	Produces a new excuse each time it's clicked.	`Image`: keep-calm-and-make-excuses.png `Text`: None
`Label`	`ExcuseLabel`	This is where the excuse will appear.	`BackgroundColor`: Red `FontSize`: 28 `Text`: "Your Excuse:"



b) En el Editor de Bloques, crea las siguientes listas:





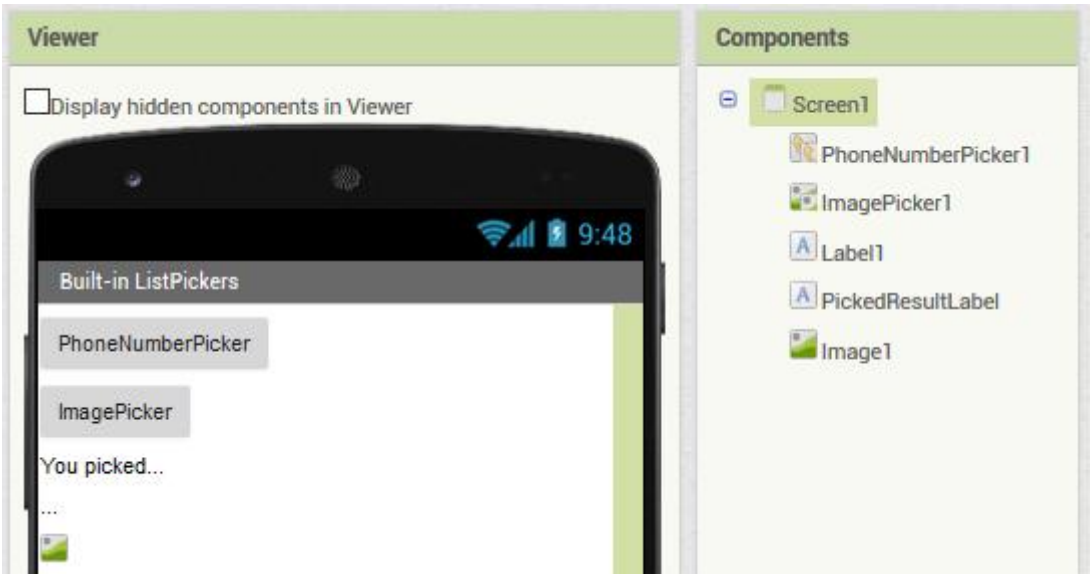
c) ¿Cómo debe funcionar esta app? Cuando el usuario pulse en el botón "ExcuseButton", el generador de excusas tomará un elemento al azar de la lista "Apology", le concatenará un elemento al azar de la lista "Person", después un elemento a azar de la lista "Action", y finalmente un elemento al azar de la lista "Place", para mostrar en la etiqueta "ExcuseLabel" una excusa del tipo "A thousand apologies, my sister ate my homework under the bridge".

Ejercicio 9.3. Listas preconstruidas: "ImagePicker" y "PhoneNumberPicker".
En este ejercicio vamos a construir una app para probar los componentes "ImagePicker" y "PhoneNumberPicker".

- El componente "ImagePicker" (paleta "Media") sirve para abrir la galería de imágenes del teléfono para elegir una imagen. Al seleccionar una imagen, se guarda una copia en la tarjeta SD. Además, la ruta de la imagen se almacena automáticamente en la propiedad "(ImagePicker).Selection".

- El componente "PhoneNumberPicker" (Paleta "Social") nos permite elegir uno de los contactos de nuestro teléfono. App Inventor puede accder y usar el nombre del contacto, su número de teléfono, y su imagen.

a) Crea un nuevo proyecto llamado "BuiltInListPickers", y diseña la siguiente interfaz de usuario:



Notar que la interfaz de usuario dispone de dos etiquetas: "Label1" es una etiqueta estática que simplemente muestra el texto "You picked…". La etiqueta "PickedResultLabel" (con el texto inicial "…") sirve para mostrar la información de la imagen o del contacto seleccionado. Por su parte, el componente "Image1" servirá para mostrar la imagen del contacto elegido.

b) Programa el componente "PhoneNumberPicker1":
Después de que el usuario pulse en el botón asociado a "PhoneNumberPicker1" y seleccione un contacto de la agenda de su teléfono, queremos mostrar en la etiqueta "PickedResultLabel" el nombre del contacto y su número de teléfono. Estos datos residen en las propiedades "ContactName" y "PhoneNumber" del componente "PhoneNumberPicker1". Además, queremos mostrar en el componente "Image1" la imagen de este contacto, la cual reside en la propiedad "Picture" de "PhoneNumberPicker1".

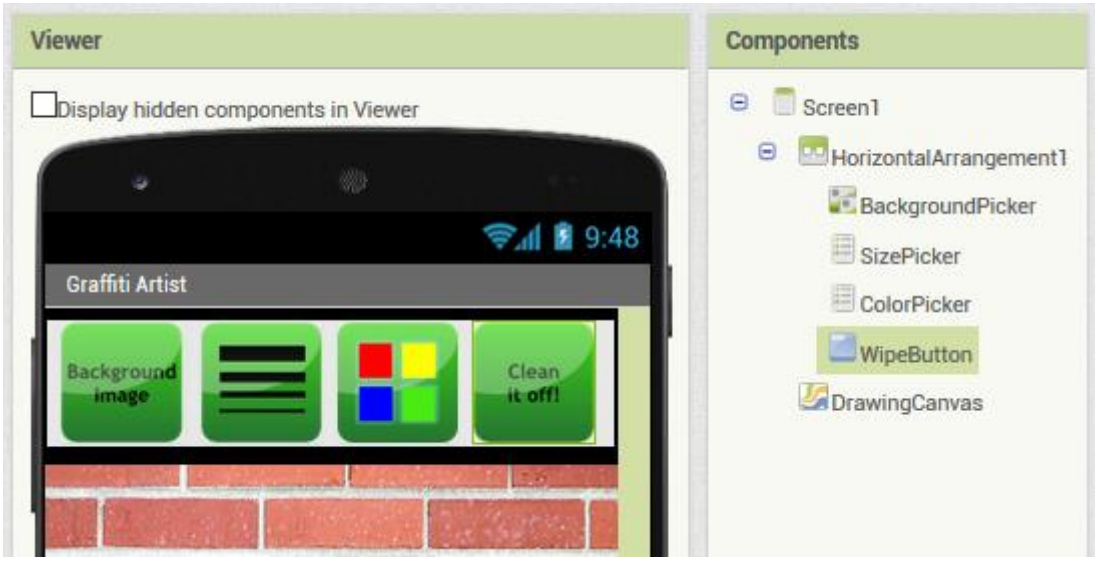
c) Programa el componente "ImagePicker1":
Después de que el usuario pulse en el botón asociado a "ImagePicker1" y seleccione una imagen de la galería del teléfono, queremos cargar el nombre de archivo de esta imagen en la etiqueta "PickedResultLabel", y la propia imagen en el componente "Image1". Ambas informaciones están disponibles en la propiedad "Selection" de "ImagePicker1".

Ejercicio 9.4. PaintPot con listas.
En este ejercicio vamos a utilizar el componente "ListPicker" para mejorar la app "PaintPot" que desarrollamos en los capítulos 3 y 4 mediante las siguientes modificaciones:
- Los susuarios podrán elegir una imagen de fondo de una galería de imágenes, en vez de tener siempre la imagen del muro.
- Los usuarios podrán cambiar el tamaño de las líneas y los puntos que dibujan.
- Los usuarios podrán seleccionar entre un abanico más amplio de colores.
- La interfaz de usuario se verá muy mejorada, usando botones con iconos en la parte superior de la pantalla.

Para esta app necesitaremos las imágenes *clear_icon.png*, *color_icon.png*, *size_icon.png*, *background_img_icon.png*, y *Wall.png*, disponibles en la carpeta de archivos para los alumnos.

a) Diseña la interfaz de usuario de la app: Para empezar, carga en App Inventor la app "PaintPot1", y crea una copia a la que llamarás "PaintPotLists", y haz lo siguiente:

- Comienza borrando los cinco botones de los colores en el organizador horizontal en la parte superior de la pantalla. (Ignora los avisos que lanza App Inventor, y simplemente pulsa "OK"). A continuación, mueve el botón "WipeButton" desde el organizador horizontal inferior al organizador horizontal superior. Borra el organizador horizontal inferior (junto con el botón para tomar una fotografía), porque ya no lo necesitaremos. (De nuevo, ignora los avisos que aparecen y solo pulsa "OK").
- Ahora añade otros tres botones (antes de hacerlo, lee un poco más abajo), y actualiza "WipeButton" como ilustra la figura:



El botón "BackgroundPicker" es realmente un componente "ImagePicker" (disponible en la paleta "Media"), que le permitrá al usuario ir a la galería de imágenes de su teléfono para ponerla como imagen de fondo del lienzo "DrawingCanvas".

El botón "SizePicker" es un componente "ListPicker" que le permitirá al usuario elegir el tamaño de las líneas y los puntos que dibujará. El rango de tamaños posibles que le ofrecerá la app es $1, 3, 5, 7$, y 9 (píxeles). Inserta estos valores en la propiedad "Elements" de "SizePicker". (También podemos experimentar con otros valores posibles añadiendo o cambiando os elementos de esta lista).

El botón "ColorPicker" es otro componente "ListPicker" que le proporciona al usuario una lista de colores para elegir. Añade la siguiente lista de colores a la propiedad "Elements" de este "ListPicker": Black, Blue, Cyan, Dark Gray, Gray, Green, Light Gray, Magenta, Orange, Pink, Red, White, y Yellow.

El botón "WipeButton" funciona exactamente igual que antes. Solo hay que cambiar su posición y la imagen del botón.

- Ubica estos cuatro botones en las posiciones mostradas en la figura, y cambia las propiedades "Image" de los respectivos botones a los archivos *clear_icon.png, color_icon.png, size_icon.png* y *background_img_icon.png*.

b) Codifica el comportamiento del botón "BackgroundPicker".

Aquí vamos a usar el mismo código que usamos en el ejercicio 9.3 con el componente "ImagePicker": Después de que el usuario haya seleccionado una imagen de la galería a través del componente "BackgroundImage" (realmente, un componente "ImagePicker"), la app la pondrá como imagen de fondo del lienzo "DrawingCanvas".

c) Cambiar los tamaños de los puntos y las líneas.

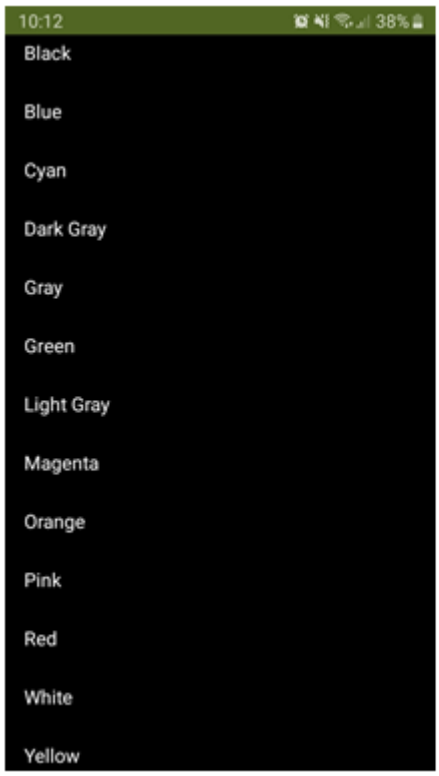
Recordar que se puede fijar el grosor de las líneas dibujadas sobre un lienzo ajustando el valor de la propiedad "LineWidth" de ese lienzo. Así mismo, podemos cambiar el tamaño de los puntos dibujados sobre un lienzo modificando el parámetro "radius" de la función "(Canvas).DrawCircle".

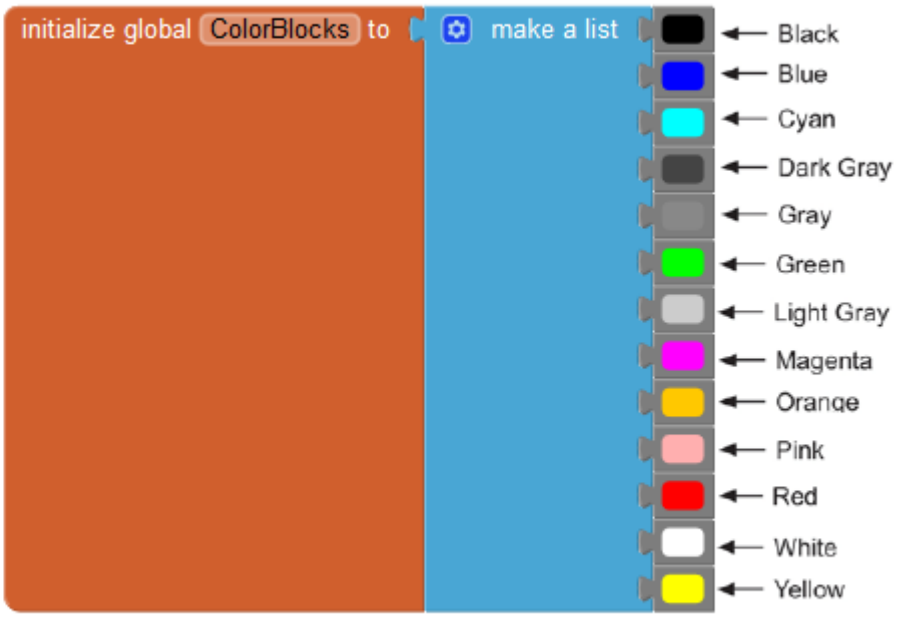
Por consiguiente, lo que aquí debemos programar el siguiente compoetamiento: Desués de que el usuario haya seleccionado un tamaño en el botón "SizePicker", la app debe ajustar el valor elegido en la propiedad "LineWidth" del lienzo "DrawingCanvas", y copiar este valor en una variable "DotRadius", con la que después podremos ajustar el valor del parámetro "radius" del bloque "call (DrawingCanvas).DrawCircle". (Por supuesto, primero deberemos haber creado una variable global llamada "DotRadius", e inicializarla a algún valor por defecto, por ejemplo, 5 píxeles).

d) Programar el botón "ColorPicker".
En la app "PaintPot" original le mostrábamos al usuario un teclado de 5 botones para cambiar el color con el que dibujaba en el lienzo. Aquí vamos a ofrecer al usuario un listado mucho más amplio de colores con los que dibujar.

Cuando el usuario pulse en el botón "ColorPicker" (realmente, un componente "ListPicker") y seleccione uno de los colores de la lista, la app fijará la propiedad "(DrawingCanvas).PaintColor" al color seleccionado.

Recordar que en el Diseñador de Componentes completamos la propiedad "Elements" de "ColorPicker" con la lista de colores que la app ofrecerá al usuario. Pero, ¿por qué no hemos creado una lista de colores con los bloques de los colores conectados a sus ranuras? ¿Por qué hemos escrito los nombres de todos esos colores en la propiedad "Elements" de "ColorPicker"? Los bloques de colores que incluye App Inventor en la bandeja "Colors" son en realidad números. Y claro, pedirle al usuario que elija un color entre -65536 y -1677691 es mucho menos transparente que decirle que elija entre Rojo o Azul. Así pues, el problema aaquí es cómo asociar el texto Azul con el bloque de color (-1677691). La forma más sencilla de hacerlo es crear una nueva lista de bloques de colores que se ajuste al orden de los elementos de "ColorPicker". Después, podremos usar la propiedad "SelectionIndex" del color elegido en "ColorPicker" para buscar el valor numérico de ese color en la lista (ver figura).





Así, cuando el usuario elija un color en el componente "ColorPicker", podemos usar su índice para seleccionar el bloque de color correspondiente en la variable tipo lista "ColorBlocks", y con ese color fijar la propiedad "PaintColor" de "DrawingCanvas".
Un ejemplo puede ayudarnos a entender esto:

- El usuario pulsa en el botón "ColorPicker" y elige el color Azul (Blue), que es el segundo de la lista (index= 2).
- Se dispara el evento "when (ColorPicker).AfterPicking"-
- La app fija la propiedad "(DrawingCanvas).PaintColor" al bloque de color que esté en la posición 2 de la lista "ColorBlocks", que es el bloque Azul.

Ejercicio 9.5. Araña (1).

En este ejercicio vamos a usar una lista de imágenes para animar un sprite en la pantalla del móvil. Para crear la animación, combinaremos cambios en la posición de un sprite con cambios en su imagen. Pero para conseguir que la animación sea suave, necesitaremos muchas imágenes diferentes del mismo sprite. La app que construiremos aquí usa 10 imágenes de una araña para simular su movimiento por pantalla:

a) Crea un nuevo proyecto, llamado "CreepySpider", en el que usarás un reloj y una lista para animar el sprite de una araña. La araña se moverá sobre un fondo de hojas secas. Además, cuando el usuario toque a la araña, la araña emitirá un sonido de mosdisco, el teléfono vibrará, la araña dejará de moverse. Cuando el usuario levante su dedo de la araña, esta empezará a moverse nuevamente.

Para este proyecto necesitaremos las 10 fotogramas para animar a la araña, llamados *spider1frame.gif*, *spider2frame.gif*, …, *spider9frame.gif*. (Estas imágenes se han obtenido de la web http://heathersanimations.com/, una fantástica página de imágenes gratuitas).

También necesitaremos la imagen de fondo (*leavesbackground.jpg*), el sonido de la araña (*spiderbitesound.wav*), y una imagen estática de la araña para el icono de la app (*creepyspider.png*). Todos estos archivos están disponibles en la carpeta de alumnos.

NOTA: **Reventar archivos GIF**.
Un archivo animado GIF (Graphics Interchange Format) es un aimagen gráfica que suele mostrarse en las páginas web y que parece estar en movimiento. Para crear esta ilusión de movimiento, el archivo GIF contiene un cierto número de imágenes estáticas que van sucediéndose ininterrumpidamente. Si queremos obtener las imágenes individuales que constituyen un archivo GIF animado, debemos "reventar" (explode) el archivo. Una página web que nos permite hacer esto de forma rápida y sencilla es http://www.gif-explode.com/.

b) Diseña la interfaz de usuario de la app, con los componentes listados en la tabla:

Creepy Spider			
Screen1 properties	`AlignHorizontal`: Center `AlignVertical`: Center `ScreenOrientation`: Portrait `Scrollable`: No `Icon`: creepyspider.png `Title`: Creepy Spider		
Components	**What do I rename it?**	**What does it do?**	**What properties do I set?**
`Canvas`	`Canvas1`	Allows you to position sprites	`BackgroundImage`: leavesbackground.jpg `Width`: Fill Parent `Height`: Fill Parent
`ImageSprite`	`SpiderImagesprite`	Crawls diagonally down the screen	`Heading`: 225 `Interval`: 100 `Picture`: Spider1frame.gif `Rotates`: No (unselected) `Speed`: 5 `X`: 190 `Y`: 10
`Sound`	`Spiderbitesound`	Provides a biting sound that is activated when the user touches the spider	`Source`: spiderbitesound.wav
`Clock`	`Clock1`	Reminds the spider sprite to change its image each time it moves	`Interval`: 100
Media files			
13 files downloaded from our website: spiderbitesound.wav leavesbackground.jpg creepyspider.png Spider1frame.gif (and nine more files labelled Spider2frame.gif, and so on, up to Spider10frame.gif)			

b) Crea la lista de imágenes para animar a la araña.
Notar que a "SpiderImageSprite" le hemos asociado en el diseñador la imagen *Spider1frame.gif*, pero necesitaremos construir la lista de imágenes que le iremos asociando a la araña secuencialmente para animarla. Para ello:

A continuación, define una variable "index" para recorrer la lista de imágenes de la araña. Inicialízala al valor adecuado.

c) Programa a la araña para cambier de disfraz cada décima de segundo.
Mediante el manejador de evento "when (Clock1).Timer", vamos a hacer que el sprite de la araña cambie a la siguiente imagen de su lista cada décima de segundo (esto es, cada $100\ ms$). En el Diseñador hemos indicado que el sprite de la araña comience mostrando la imagen *Spider1frame.gif* . Ahora queremos que, tras cada cuenta de $100\ ms$ (evento "(Clock1).Timer"), la araña muestre la siguiente imagen de su lista, a saber, *Spider2frame.gif*, *Spider3frame.gif*, y así sucesivamente, hasta *Spider10frame.gif*. Si el sprite llega a la última imagen de la lista, *Spider10frame.gif*, queremos que la siguiente imagen que muestre sea la primera, *Spider1frame.gif*. Utiliza la variable "index" para escribir el programa que nos permita recorrer la lista de imágenes de la araña, e ir cambiando la imagen que muestra cada $100\ ms$. Recuerda que la imagen que muestra el sprite de la araña puede cambiarse mediante código usando el bloque "set (SpiderImageSprite).Picture"

d) Anima a la araña.
Los ajustes que has fijado para el sprite de la araña en el Diseñador de Componentes (ver tabla) hacen que la araña se mueva diagonalmente hacia abajo desde la esquina superior derecha:

Setting	In English!
`Heading`: 225	Moves toward the bottom-left corner of the screen
`Interval`: 100	Moves every 1/10 of a second
`Speed`: 5	Moves 5 pixels each time
`X`: 190 `Y`: 10	Starts in the top-right corner of the screen

Pero si no hacemos nada más, la araña terminará llegando al borde izquierdo de la pantalla,y allí se quedará inmóvil. Pero aquí queremos que, cuando la araña llegue el borde de la pantalla, vuelva a aparecer en su posición inicial $(x, y) = (190, 10)$. Programa el sprite de la araña para lograr este comportamiento.

e) Haz que la araña muerda y se quede inmóvil.
En las secciones (e) y (f) de este ejercicio vamos a hacer uso de los eventos "TouchDown" y "TouchUp"del sprite de la araña. En este caso, cuando el usuario toque a la araña y mantenga su dedo sobre ella (evento "TouchDown"), la araña responderá reproduciendo un sonido de mordedura, vibrando, y quedándose inmóvil (fijando su velocidad a cero). Además de todo ello, el manejador deberá detener el reloj para que deje de dispararse su temporizador. Escribe el programa que codifica este comportamiento.

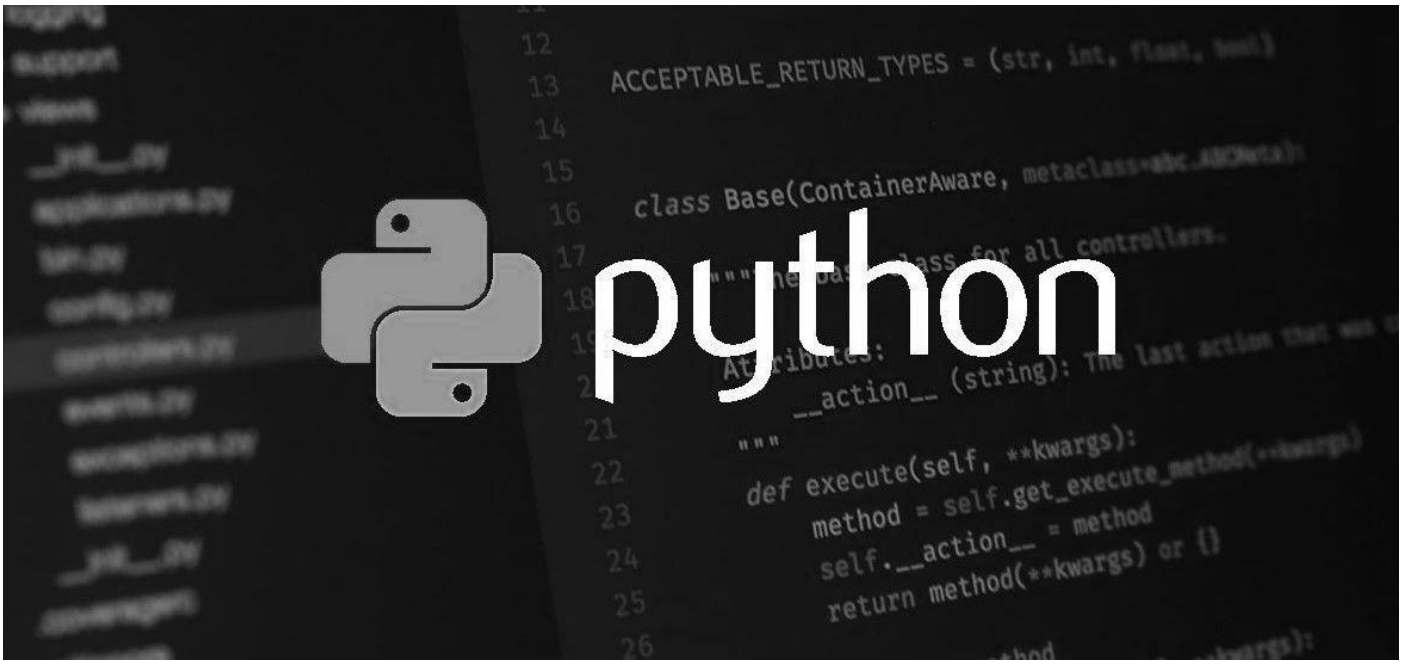
f) Consigue que la araña vuelva a moverse de nuevo.
Cuando el usuario levante su dedo de la araña (evento "TouchUp"), la araña volverá a moverse. Para ello, el manejador del evento hará que la araña deje de estar inmóvil (estaurando el valor inicial de su velocidad), y activará de nuevo el reloj para que su temporizador vuelva a seguir disparándose.

AMPLIACIÓN 1: Para mejorar la animación de la araña, podríamos hacer que ésta rebote al llegar al borde de la pantalla. Para ello, deberíamos cambiar la dirección en la que pasa a moverse tras rebotar, dependiendo del borde contra el que rebote. (Por ejemplo, si la araña se mueve en la dirección $225°$ (hacia abajo y a la izquierda), e impacta contra el borde izquierdo, su nueva dirección debería ser hacia abajo y a la derecha, dirección $225° + 90° = 315°$. Si la araña se mueve hacia abajo y a la derecha ($315°$) y golpea el borde inferior, debería pasar a moverse hacia arriba y a la derecha, dirección $315° + 90° = 405° \equiv 45°$, etc.). Además de cambiar de dirección, deberíamos rotar al sprite para que pase a mirar en la nueva dirección en la que está caminando: Para ello, activa la propiedad "Rotates" del sprite.

AMPLIACIÓN 2: También podemos hacer que el fondo cambie cada vez que el usuario toque a la araña. Esto implicaría añadir una acción extra al manejador del evento "TouchDown" para cambiar la imagen de fondo del lienzo al siguiente elemento de una hipotética lista de fondos.

AMPLIACIÓN 3: Otra modificación que podríamos probar sería incrementar el nivel de dificultad fijando fondos que hagan cada vez más difícil encontrar y tocar a la araña. (Para ello, también podríamos hacer a la araña cada vez más pequeña). La puntuación del usuario podría depender de cuánto tarde en encontrar y en tocar a la araña. Incluso podríamos crear un fondo lleno de imágenes de arañas entre las cuales el usuario debería encontrar a la araña móvil.

PARTE II: PROGRAMACIÓN CON PYTHON.



TEMA 1. PRIMEROS PASOS EN PYTHON.

1.1. INTRODUCCIÓN.

Python es uno de los diez lenguajes de programación más utilizados en el mundo. Algunos ejemplos de organizaciones que usan Python son Google, NASA, DropBox, Industrial Light and Magic, etc. El software necesario para editar y ejecutar los programas escritos en Python está disponible de forma gratuita en la web www.python.org.

1.2. EL SHELL INTERACTIVO.

El **intérprete** de Python es el software que permite ejecutar los programas escritos en Python. Pero para escribir los programas, necesitamos usar el entorno de desarrollo interactivo o **IDLE** (Interactive DeveLopment Environment) de Python.

Para arrancar el IDLE pinchamos en el botón de inicio de Windows, escribimos IDLE en la casilla de búsqueda, y seleccionamos "IDLE (Python GUI)". Al arrancar el IDLE aparece una pantalla prácticamente en blanco como la mostrada en la figura:

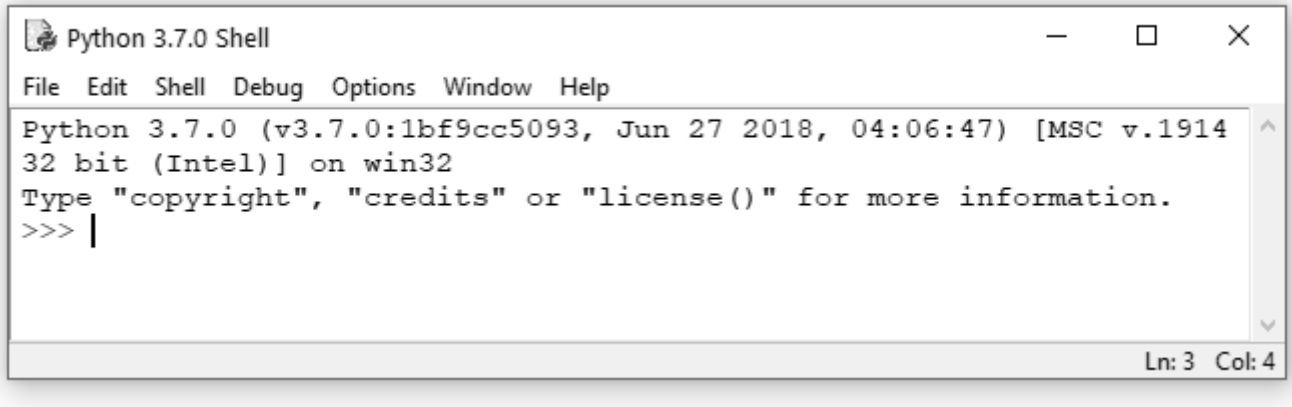


Figura 1.1. El shell interactivo.

Esta ventana se denomina **shell interactivo**. Toda instrucción que escribamos en el shell interactivo se ejecuta de forma inmediata. Para probar el shell interactivo y ejecutar nuestra primera instrucción en Python, escribimos lo siguiente tras el "prompt" (>>>):

```
>>> print('Hola, mundo.')
```

Después de escribir este comando y presionar la tecla ENTER, el shell interactivo debería mostrar esta respuesta:

```
Hola, mundo.
```

1.3. EL EDITOR DE ARCHIVOS.

Aunque el shell interactivo es muy útil para ejecutar instrucciones una a una, a la hora escribir programas utilizaremos **el editor de archivos** de Python. Para abrir el editor de archivos desde el IDLE, seleccionamos "File" → "New File". El editor de archivos permite escribir muchas instrucciones, guardar el archivo, y ejecutar el programa completo. Es fácil diferenciar entre el shell interactivo y el editor de archivos porque la ventana del shell siempre es la del prompt (>>>). El editor de archivos no tiene prompt.

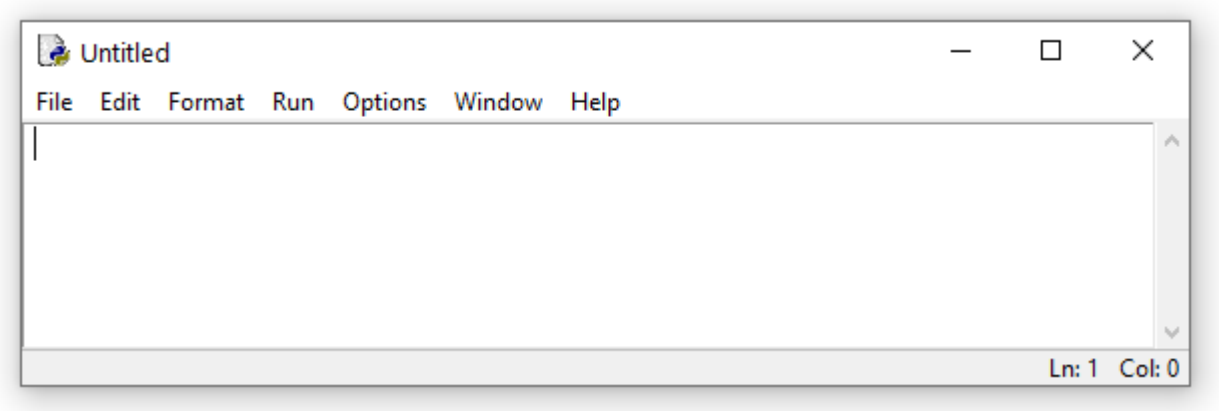


Figura 1.2. El editor de archivos.

1.4. EXPRESIONES.

En Python decimos que `2+3` es una **expresión**, y constituye el tipo de instrucción más básica que podemos encontrar en este lenguaje. Las expresiones consisten en **valores** (como el `2` y el `3`) y **operadores** (como el `+`), y siempre pueden **evaluarse** (esto es, reducirse) a un valor único. En el ejemplo previo, `2+3` se evalúa al valor único `5`.

Python proporciona múltiples operadores que podemos usar en las expresiones. La tabla 3.1 muestra todos los operadores *matemáticos* disponibles en Python:

OPERADOR	OPERACIÓN	EJEMPLO	SE EVALÚA A…
`**`	Exponente	`2 ** 3`	`8`
`%`	Módulo (resto)	`22 % 8`	`6`
`//`	División entera	`22 // 8`	`2`
`/`	División	`22 / 8`	`2.75`
`*`	Multiplicación	`3 * 5`	`15`
`-`	Resta	`5 - 2`	`3`
`+`	Suma	`2 + 3`	`5`

Tabla 1.1. Operadores matemáticos básicos.

El orden en el que se efectúan las operaciones combinadas en una expresión (la llamada **precedencia**) es similar a la de las matemáticas. El operador `**` se evalúa en primer lugar; a continuación se evalúan los operadores `*`, `/`, `//`, y `%` (de izquierda a derecha); en último lugar, los operadores `+` y `-` (también de izquierda a derecha). Podemos usar paréntesis para saltarnos la precedencia si así lo necesitamos. A modo de ejemplo, observemos cómo evalúa Python la siguiente expresión combinada:

```
(5 - 1) * ((7 + 1) / (3 - 1))
    ↓
4 * ((7 + 1) / (3 - 1))
    ↓
4 * (( 8  ) / (3 - 1))
    ↓
4 * (( 8  ) / (  2  ))
    ↓
4 * 4.0
    ↓
 16.0
```

1.5. TIPOS DE DATOS.

Un **tipo de datos** es la categoría a la que pertenece un cierto valor. Los tipos de datos más comunes en Python están listados en la tabla 3.2.

TIPO DE DATOS	EJEMPLOS
Enteros	`-2, -1, 0, 1, 2, 3, 4, 5,` etc.
Números de punto flotante	`-1.25, -0.76, 0.12, 1.127, 2.561,` etc.
Cadenas	`'a','aaa','Hola, mundo!','Tengo 5 gatos.',` etc.

Tabla 1.2. Los tipos de datos entero, punto flotante, y cadena.

Los valores como `-2`, `4`, `20`, etc. son ejemplos de valores **enteros** (integer). Los números con un punto decimal, como `3.14` y `-5.05`, se denominan números de **punto flotante** (floating - point). Notar que, aunque el valor `42` es un entero, el valor `42.0` es un número de punto flotante. Los números de punto – flotante muy grandes o muy pequeños pueden expresarse usando notación científica, en la forma `1.2e34` (lo que significa $1{,}2 \times 10^{34}$) o `0.46e-19` (esto es, $0{,}46 \times 10^{-19}$). Python también usa valores de tipo texto llamados **cadenas** (strings). Un valor de tipo cadena siempre va rodeado por comillas simples (por ejemplo, `'Hola'`, `'Adiós, mundo cruel!'`, etc.) para que Python sepa dónde empieza y dónde termina la cadena. Mediante la instrucción `''` también podemos tener una cadena sin caracteres, a la que denominamos **cadena vacía**.

1.6. CONCATENACIÓN DE CADENAS.

El significado de un operador puede cambiar dependiendo del tipo de datos de los valores que opera. Por ejemplo, `+` es un operador de suma cuando opera sobre dos valores enteros o sobre dos valores de tipo punto flotante. Pero cuando lo usamos para operar dos valores de tipo cadena, el operador `+` actúa como un operador de **concatenación** que une las dos cadenas:

```
>>> 'Alex' + 'Susana'
'AlexSusana'
```

Esta expresión se reduce a un valor único de tipo cadena que combina el texto de las dos cadenas. Si intentamos usar el operador `+` sobre una cadena y un entero, Python mostrará un mensaje de error:

```
>>> 'Alex' + 5
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    'Alex' + 5
TypeError: can only concatenate str (not "int") to str
```

1.7. VARIABLES.

Una **variable** es como una caja etiquetada con un nombre en la que podemos guardar un cierto valor, que posteriormente podemos recuperar usando el nombre de la caja que lo contiene.

Para guardar un valor en una variable usamos una **sentencia de asignación**, por ejemplo, `edad = 35`. Esta sentencia indica que a la variable llamada `edad` le asignamos el valor entero `35`. A partir de ahora, la variable `edad` almacenará el entero `35` dentro de ella.

A modo de ejemplo, inserta el siguiente código en el shell interactivo:

```
>>> manzanas = 10
>>> manzanas
10
>>> peras = 2
>>> frutas = manzanas + peras
>>> frutas
12
>>> manzanas = manzanas + 4
>>> manzanas
14
```

Una variable se **inicializa** (o se crea) la primera vez que almacenamos un valor dentro de ella (como en `manzanas = 10`, o en `peras = 2`). Después de eso, podemos usarla en expresiones junto con otras variables o valores, como en `frutas = manzanas + peras`. Cuando a una variable se le asigna un nuevo valor, como en `manzanas = manzanas + 4`, el valor antiguo se borra. Ésta es la razón por la que la variable `manzanas` se evalúa a `14` en vez de a `10` al final del ejemplo. A esto se le llama **sobrescribir** la variable.

1.8. NUESTRO PRIMER PROGRAMA EN PYTHON.

Abre el editor de archivos y escribe el siguiente programa:

```
# Este programa dice hola y me pregunta mi nombre y mi edad.

print('Hola, mundo.')
print('¿Cómo te llamas?')
miNombre = input()
print('Encantado de conocerte, ' + miNombre)
print('La longitud de tu nombre es:')
print(len(miNombre))
print('¿Cuántos años tienes?')
miEdad = input()
print('El año que viene tendrás ' + str(int(miEdad)+1) + ' años.')
```

Después de haber escrito todo el código, lo guardamos. Para ello, seleccionamos "File" → "Save As" en el menú del editor de archivos. En la ventana emergente escribimos `hola.py` como nombre del archivo, y lo guardamos en nuestra carpeta de trabajo.

Tras haber guardado el programa, podemos ejecutarlo seleccionando "Run" → "Run Module" (o simplemente pulsando la tecla F5). El programa se ejecuta en la ventana del shell interactivo. Escribimos nuestro nombre y edad cuando el programa nos lo pida. La salida del programa debería ser similar a ésta:

```
Hola, mundo.
¿Cómo te llamas?
Alejandro
Encantado de conocerte, Alejandro.
La longitud de tu nombre es:
9
¿Cuántos años tienes?
15
El año que viene tendrás 16 años.
```

Cuando no hay más líneas de código que ejecutar, el programa de Python **finaliza**, esto es, deja de ejecutarse.

A continuación, vamos a analizar el programa instrucción a instrucción:

COMENTARIOS.

La línea (1) del programa es un **comentario**. En Python, todo texto precedido de una almohadilla (`#`) forma parte de un comentario. Python ignora los comentarios, por lo que podemos usarlos para escribir notas o recordatorios de lo que hace una cierta línea de código.

LA FUNCIÓN PRINT().

La función `print()` muestra por pantalla *cualquier* valor (entero, flotante, o cadena) que hayamos puesto dentro de sus paréntesis. En la línea (2) del programa, la instrucción `print('Hola, mundo.')` significa "Imprime por pantalla el texto de la cadena `'Hola, mundo.'`". Al ejecutar esta instrucción decimos que Python está **llamando** a la función `print()`, y que a dicha función le estamos **pasando** la cadena `'Hola, mundo.'` como **argumento**. (En programación, al valor que le pasamos a una función se le denomina *argumento*). Notar que las comillas no se imprimen por pantalla (las comillas solo indican dónde empieza y dónde termina la cadena, pero no forman parte de la propia cadena). Al escribir el nombre de una función, los paréntesis que la acompañan nos permiten identificar la instrucción como una llamada a esa función. Ésta es la razón por la que se escribe `print()` y no `print`.

LA FUNCIÓN INPUT().

La función `input()` pausa el programa, y espera a que el usuario escriba un texto *con el teclado* y presione la tecla ENTER. La instrucción `miNombre = input()` es una llamada a la función `input()` que obtiene una *cadena* igual al texto escrito por el usuario, y que guarda dicho valor en la variable `miNombre`.

IMPRIMIR EL NOMBRE DEL USUARIO.

En la línea (5), la función `print()` recibe la expresión `'Encantado de conocerte, ' + miNombre`. Recordar que las expresiones siempre pueden evaluarse a un valor único. Si el valor almacenado en la variable `myNombre` es `'Alejandro'`, esta expresión se evalúa a `'Encantado de conocerte, Alejandro'`. El programa pasa esta cadena a la función `print()` para que la imprima por pantalla.

LA FUNCIÓN LEN().

La función `len()` recibe un valor de tipo cadena (o una variable que contenga una cadena), y devuelve un valor entero igual a número de caracteres de la cadena. Por ejemplo:

```
>>> len('Hola')
4
>>> len('Encantado de conocerte.')
23
```

Como en los ejemplos previos, la expresión `len(miNombre)` en la línea (7) se evalúa a un entero. Después, el programa le pasa este valor a la función `print()` para mostrarlo por pantalla. Notar que la función `print()` permite mostrar por pantalla tanto enteros como cadenas (o cualquier otro valor que le pasemos).

LAS FUNCIONES STR(), INT(), Y FLOAT().

Observar el error que obtenemos cuando escribimos la siguiente instrucción en el shell interactivo:

```
>>> print('Tengo ' + 18 + ' años.')
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    print('Tengo ' + 18 + ' años.')
TypeError: can only concatenate str (not "int") to str
```

No es la función `print()` la que produce el error, sino la expresión que le hemos pasado. Python produce este error porque el operador `+` sirve para sumar dos números o para concatenar dos cadenas, pero no para concatenar una cadena con un entero. Podemos arreglar este error convirtiendo el entero `18` en una cadena (`'18'`) usando la función `str()`:

```
>>> str(18)
'18'
>>> print('Tengo ' + str(18) + ' años.')
Tengo 18 años.
```

Las funciones `str()`, `int()`, y `float()` convierten el valor que les pasamos como argumento en una cadena, un entero, o un número de punto flotante, respectivamente. Vamos a usar estas funciones en el shell interactivo para convertir algunos valores:

```
>>> str(5)
'5'
>>> str(3.14)
'3.14'
>>> int('42')
42
>>> int(3.25)
3
>>> int(3.99)
3
>>> float('3.14')
3.14
>>> float(10)
10.0
```

La función `str()` es útil cuando tenemos un entero o un flotante que queremos concatenar a una cadena. La función `int()` también es útil si tenemos un número en forma de cadena y lo queremos usar para hacer algunas operaciones matemáticas. Por ejemplo, la función `input()` siempre devuelve una cadena, incluso cuando el usuario inserta un número. Escribe `edad = input()` en el shell interactivo, y responde `15` cuando quede a la espera de tu respuesta:

```
>>> edad = input()
15
>>> edad
'15'
```

El valor almacenado en la variable `edad` no es el entero `15`, sino la cadena `'15'`. Si queremos realizar operaciones matemáticas usando el valor almacenado en `edad`, primero hemos de usar la función `int()` para convertir la cadena en entero, y después guardar este nuevo valor en la propia variable `edad`:

```
>>> edad = int(edad)
>>> edad
15
```

A partir de ahora ya podemos usar la variable `edad` como un entero, y no como una cadena:

```
>>> (edad + 5)/4
5.0
```

En la línea (10) del programa hemos usado las funciones `int()` y `str()` para obtener un valor con el tipo de datos apropiado. La variable `miEdad` contiene el valor devuelto por la función `input()`. Como esta función siempre devuelve una cadena (incluso cuando el usuario escribe un número), podemos usar el código `int(miEdad)` para obtener un valor entero a partir de la cadena `miEdad`. A continuación, el programa le suma 1 a este valor entero, mediante la expresión `int(miEdad)+1`. Después, pasamos esta expresión a la función `str()` para convertir el resultado en una cadena: `str(int(myAge)+1)`. Por último, concatenamos ese valor con las cadenas `'El año que viene tendrás '` y `' años.'` para crear el mensaje final e imprimirlo por pantalla con la función `print()`.

Por ejemplo, digamos que el usuario proporciona un `15` como respuesta al `input()` de la línea (9). En ese caso, la variable `miEdad` almacenará el valor `'15'`. Los pasos mediante los que se evaluará la expresión de la línea (10) son los siguientes:

```
print('El año que viene tendrás ' + str(int(miEdad)+1) + ' años.')

print('El año que viene tendrás ' + str(int('15')+1) + ' años.')

print('El año que viene tendrás ' + str(15+1) + ' años.')

print('El año que viene tendrás ' + str(16) + ' años.')

print('El año que viene tendrás ' + '16' + ' años.')

print('El año que viene tendrás 16 años.')
```

1.9. EL MÓDULO MATH.

Cualquier programa de Python puede utilizar un conjunto básico de funciones a las que denominamos **funciones integradas** (*built-in* functions), entre las que figuran las funciones `print()`, `input()`, y `len()` que ya hemos usado antes. Pero Python también incluye una serie de módulos llamados **biblioteca estándar**. Un **módulo** es un programa de Python que contiene un grupo de funciones relacionadas que podemos incorporar en nuestros programas. Por ejemplo, el módulo `math` contiene algunas funciones matemáticas avanzadas. (No confundir estas funciones matemáticas con los operadores matemáticos básicos que ya estudiamos en secciones previas). Otro ejemplo es el módulo `random` , que incluye funciones para trabajar con números aleatorios.

Sin embargo, antes de poder usar las funciones de un módulo, debemos importarlo mediante una sentencia `import`. (Por ejemplo, `import math`). Después de haber importado un módulo, podremos usar en nuestro programa todas las funciones que incluye.

A modo de ejemplo, vamos a probar el módulo `math`, el cual nos da acceso a la función `math.sqrt()` que obtiene la raíz cuadrada de un número. Para ver esta función en acción, vamos a escribir un programa que obtenga las raíces reales de una **ecuación cuadrática** de la forma $ax^2 + bx + c = 0$. Como sabemos, las soluciones vienen dadas por la **fórmula cuadrática**:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Escribe el siguiente código en el editor de archivos de Python y guárdalo como `formulaCuadratica.py`:

```
# Este programa usa la fórmula cuadrática para hallar
# las raíces reales de la ecuación cuadrática ax^2+bx+c=0.

import math

# Le pedimos al usuario los coeficientes a, b, y c.

print('Este programa halla las raíces reales de ax^2+bx+c=0.')
print('Por favor, inserta el coeficiente a:')
a = float(input())
print('Por favor, inserta el coeficiente b:')
b = float(input())
print('Por favor, inserta el coeficiente c:')
c = float(input())

# Hallamos las raíces reales de la fórmula cuadrática.
d = (b*b)-4*a*c  #Primero obtenemos el determinante.
if d < 0:
    print('Esta ecuación no tiene raíces reales.')
else:
    x1 = (-b+math.sqrt(d))/(2*a)
    print('Primera raíz real: x1 = ' + str(x1))
    x2 = (-b-math.sqrt(d))/(2*a)
    print('Segunda raíz real: x2 = ' + str(x2))
```

Si usamos este programa para resolver la ecuación $x^2 - 5x + 4 = 0$, la salida debería ser la siguiente:

```
Este programa halla las raíces reales de ax^2+bx+c=0.
Por favor, inserta el coeficiente a:
1
Por favor, inserta el coeficiente b:
-5
Por favor, inserta el coeficiente c:
4
Primera raíz real: x1 = 4.0
Segunda raíz real: x2 = 1.0
```

Por supuesto, el módulo `math` incorpora muchas más funciones matemáticas aparte de la función `math.sqrt()`. Para acceder al resto de funciones disponibles, notar que al escribir `math.` en el shell interactivo, aparece una ventana contextual que muestra una lista con todas las funciones incluidas en el módulo `math`:

Figura 1.3. Funciones disponible en el módulo `math`.

Por último, debemos saber que con una sola sentencia `import` podemos importar simultáneamente varios módulos distintos. A modo de ejemplo, podemos escribir la siguiente instrucción:

```
import math, random, sys, os
```

Después de ejecutar esta sentencia, podremos usar cualquiera de las funciones contenidas dentro de estos cuatro módulos.

1.10. EJERCICIOS.

EJERCICIO 1. Suma. Escribe un programa que le pida al usuario que introduzca por el teclado dos números, que sume estos números, y que muestre el resultado por pantalla. Guarda el programa como `Ejer1.py`. CUIDADO: Los números proporcionados por el usuario podrían ser decimales, y el programa debería ser capaz de sumarlos sin problemas.

EJERCICIO 2. Nombres. Escribe un programa que te pida tu nombre y el del compañero que tengas más cerca. Este programa debe (1 y 2) calcular las longitudes los dos nombres, (3) concatenar ambos nombres, y (4) calcular la longitud de esta cadena combinada. El programa debe mostrar estos cuatro resultados por pantalla. Guarda el programa como `Ejer2.py`.

EJERCICIO 3. Banco. Imagina que trabajas en la ventanilla de un banco. Vas a recibir a tres clientes, que harán tres ingresos, y otros dos clientes que realizarán sendas retiradas de efectivo. Para hacer cada ingreso y cada retirada, el programa pregunta cuánto dinero quiere ingresar o retirar el cliente, y el cliente responde a través del teclado. En el momento que el usuario responde, se considera que la operación se ha realizado. Asume que primero se hacen los tres ingresos, y luego las dos retiradas de efectivo. La tarea del programa es hacer el balance diario del banco, esto es $(ingresos - retiradas)$, y mostrar el resultado por pantalla. Utiliza un mensaje similar a éste:

```
Hoy se han ingresado 1500 € en total, y se han retirado 950 € en total. El
                        balance final es de 550 €.
```

(Notar que el balance final puede ser una cantidad negativa, si las retiradas exceden a los ingresos). Guarda el programa como `Ejer3.py`.

EJERCICIO 4. Nota media. Escribe un programa en el que insertes las notas de los tres exámenes que has hecho de matemáticas durante el trimestre, y que calcule la nota media de los exámenes. El primer examen cuenta el 20% de la nota media, el segundo el 30%, y el último el 50%. Cuando tengas la nota media ponderada de los tres exámenes, inserta el número de negativos que tienes por no haber traído los deberes. Cada negativo resta 0,1 puntos a la nota media de los exámenes. Al final, el programa muestra por pantalla la nota final trimestral. Guarda el programa como `Ejer4.py`.

EJERCICIO 4b. Caída libre.

(a) Dejamos caer una pelota desde una torre de altura h. Su velocidad inicial es cero, y al ser liberada, la pelota se acelera hacia abajo debido a la gravedad a razón de $g = 9{,}81\ m/s^2$. El objetivo es escribir un programa que le pida al usuario la altura h de la torre en metros y un intervalo temporal t en segundos, y que imprima por pantalla la altura a la que se encuentra la pelota en relación al nivel del suelo un tiempo t después de haber sido soltada. Ignorando la resistencia del aire, la respuesta viene dada por la fórmula $y(t) = -gt^2/2 + h$ (válida para un sistema de referencia con origen en el suelo, donde $y = 0$, tal y como nos pide el ejercicio). Guarda el programa como `Ejer4b_1.py`.

(b) Dejamos caer de nuevo una pelota desde una torre de altura h con una velocidad inicial cero. Escribe un programa que le pida al usuario la altura en metros de la torre, y que calcule e imprima el tiempo que tarda la pelota en llegar al suelo ($y = 0$), ignorando la resistencia del aire. Usa tu programa para calcular el tiempo que tardará una pelota liberada desde una altura de $100\ m$ en llegar al suelo. Guarda el programa como `Ejer4b_2.py`.

EJERCICIO 4c. Tiro parabólico. De física sabrás que el movimiento de proyectil es una combinación de un movimiento a velocidad constante en la dirección horizontal, y de un movimiento de caída libre (esto es, un movimiento bajo la única influencia de la gravedad) en la dirección vertical. (En este caso, estamos ignorando la resistencia del aire). Cuando se combinan los dos movimientos, la trayectoria resultante es una parábola. Imagina que un cañón dispara un obús con una cierta velocidad inicial $\vec{v}_0$ y con un cierto ángulo θ desde la horizontal. La componente horizontal de la velocidad inicial es $v_{0x} = v_0 \cos\theta$, y la componente vertical es $v_{0y} = v_0 \sin\theta$. Situando el origen de nuestro sistema de coordenadas en el punto en el que el proyectil comienza su vuelo, el valor de la coordenada x del proyectil en cualquier instante de tiempo posterior es $x(t) = v_{0x}t$, y el de la coordenada y es $y(t) = v_{0y}t - \frac{1}{2}gt^2$, donde $g = 9{,}8\ m/s^2$ es la aceleración gravitacional. Usando estas ecuaciones, podemos calcular el tiempo total de vuelo t_{total}, la altura máxima h alcanzada, y el alcance horizontal d del proyectil:

$$t_{total} = \frac{2v_0 \sin\theta}{g} \qquad h = \frac{(v_0 \sin\theta)^2}{2g} \qquad d = \frac{v_0^2 \sin 2\theta}{g}$$

Escribe un programa que, dados los valores de la rapidez inicial v_0 (en metros por segundo) y el ángulo θ (en grados) con el que se dispara el proyectil, calcule el tiempo total de vuelo (en segundos), la altura máxima alcanzada (en metros), y el alcance máximo horizontal (en metros). Guarda el programa como `Ejer4b.py`.

NOTA: Las funciones trigonométricas seno y coseno están contenidas el módulo `math`, y se escriben como `math.sin()` y `math.cos()`. Estas funciones necesitan como argumento el ángulo en *radianes*. Aquí tienes un ejemplo de utilización de estas funciones, donde la función `math.pi` (sin argumentos) simplemente proporciona el número π.

```
# Este programa calcula el coseno y el seno de un ángulo dado.

# Para usar las funciones trigonométricas, debemos importar el módulo math.
import math

print('Por favor, proporcione un ángulo (en grados).')
angulo = input()

# Convertimos el ángulo de grados a radianes, porque las funciones math.cos() y
# math.sin() requieren radianes.
angulo = (float(angulo)*math.pi)/180

print('El coseno del ángulo es:')
print(math.cos(angulo))
print('El seno del ángulo es:')
print(math.sin(angulo))
```

EJERCICIO AMPLIACIÓN. Órbitas planetarias.
En general, la órbita que sigue un cuerpo en interacción gravitacional con otro (como en el caso de un planeta y el Sol) es elíptica, de forma que el planeta a veces está más cerca y a veces está más lejos del Sol. Si se nos da la distancia l_1 de máxima aproximación del planeta al sol (el perihelio) y su velocidad lineal v_1 en el perihelio, podemos calcular cualquier propiedad de la órbita a partir de ellas.

(a) La segunda ley de Kepler dice que la distancia l_2 y la velocidad lineal v_2 en el punto más alejado (el afelio) satisfacen $l_2 v_2 = l_1 v_1$. Al mismo tiempo, la energía total (cinética más potencial) de un planeta con una velocidad v a una distancia r del Sol es:

$$E = \frac{1}{2}mv^2 - G\frac{mM}{r}$$

, donde m es la masa del planeta, $M = 1{,}9891 \times 10^{30}\ kg$ es la masa del Sol, y $G = 6{,}6738 \times 10^{-11}\ m^3 kg^{-1} s^2$ es la constante de gravitación de Newton. Aplicando la conservación de la energía, se puede demostrar que v_2 viene dada por la raíz menor de la ecuación cuadrática:

$$v_2^2 - G\frac{mM}{l_1 v_1} v_2 - \left[v_1^2 - \frac{2GM}{l_1} \right] = 0$$

Una vez tenemos v_2, podemos calcular l_2 usando $l_2 v_2 = l_1 v_1$.

(b) Dados los valores de v_1, l_1, y l_2 podemos obtener otros parámetros de la órbita, que derivan de las leyes de Kepler y de las propiedades de las elipses:
Semieje mayor:

$$a = \frac{1}{2}(l_1 + l_2)$$

Semieje menor:

$$b = \sqrt{l_1 l_2}$$

Periodo orbital:

$$T = \frac{2\pi ab}{l_1 v_1}$$

Excentricidad orbital:

$$e = \frac{l_2 - l_1}{l_2 + l_1}$$

Escribe un programa que le pida al usuario la distancia al Sol y la velocidad en el perihelio, y que calcule e imprima las cantidades v_2, l_2, T, y e.

(c) Comprueba tu programa calculando las propiedades de las órbitas de la Tierra ($l_1 = 1{,}4710 \times 10^{11}\ m$ y $v_1 = 3{,}0287 \times 10^4\ m\, s^{-1}$) y del cometa Halley ($l_2 = 8{,}7830 \times 10^{10}\ m$ y $v_1 = 5{,}4529 \times 10^4\ m\, s^{-1}$). Busca en libros o Internet el resto de datos que necesites. Deberías hallar que el periodo orbital de la Tierra es, por supuesto, 1 año, y que el del cometa Halley es de aproximadamente 76 años.

Guarda el programa como `AmpOrbitas.py`.

2. CONTROL DE FLUJO.

2.1. INTRODUCCIÓN.

Los programas que hemos escrito hasta ahora son secuenciales: El flujo de ejecución comienza en la primera instrucción, y continúa hacia abajo hasta llegar a la última, tras la cual termina el programa. Sin embargo, la verdadera programación no consiste en ejecutar una instrucción tras otra. En vez de eso, un programa podría necesitar saltarse algunas instrucciones, repetirlas, o elegir que se ejecute una de entre varias instrucciones posibles. De hecho, casi nunca querremos que un programa ejecute de principio a fin cada una de las líneas de su código. Las **sentencias de control de flujo** nos permiten decidir qué instrucciones se ejecutarán bajo qué condiciones. Un ejemplo es el programa `formulaCuadratica.py` que escribimos en el capítulo previo. Al calcular el determinante comprobamos si es negativo o no, y según el caso, hacemos una cosa u otra: Si es negativo informamos de que la ecuación no tiene soluciones reales, y en caso contrario (si no es negativo) calculamos las dos raíces reales. Como ya hemos mencionado, las sentencias de control de flujo vienen acompañadas de una condición. En las siguientes secciones vamos a estudiar cómo se construyen estas condiciones. Para ello necesitaremos presentar algunos conceptos un poco abstractos, como los valores booleanos, los operadores de comparación, y los operadores booleanos.

2.2. VALORES BOOLEANOS.

Las condiciones que gobiernan las sentencias de control de flujo se asemejan a preguntas que se formula el programa para decidir qué instrucciones debe ejecutar. Las respuestas a estas preguntas serán siempre "Sí" o "No". La representación de este tipo de respuestas es la función fundamental de los valores de tipo Booleano.

Mientras que los tipos de datos entero, punto – flotante, y cadena pueden tener un número infinito de valores posibles, el tipo de datos **Booleano** solo puede tomar dos valores: `True` (verdadero) y `False` (falso). Cuando los escribimos en un programa de Python, los valores `True` y `False` no llevan las comillas que solemos poner delante y detrás de las cadenas, y además, siempre se escriben con T y F mayúsculas (y el resto de la palabra en minúsculas). Además, y como cualquier otro valor, los valores Booleanos pueden usarse en expresiones y almacenarse en variables. Si no escribimos el valor con la primera letra en mayúscula, Python nos dará un mensaje de error.

```
>>> par = True
>>> par
True
>>> mayorEdad = false
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    mayorEdad = false
NameError: name 'false' is not defined
```

2.3. OPERADORES DE COMPARACIÓN.

Como veremos, las condiciones de las entencias de control de flujo siempre formulan pregunta del tipo "¿El valor de esta variable es mayor que 5?", "¿La cadena almacenada en esta variable coincide con esta otra cadena?", etc. Para construir estas preguntas, las condiciones utilizan operadores de comparación.

Los **operadores de comparación** nos permiten comparar dos valores y evaluar una expresión de comparación a un valor Booleano único (`True` o `False`). La tabla 3.3 muestra todos los operadores de comparación. Estos operadores evalúan una expresión a verdadero (`True`) o falso (`False`) dependiendo de los valores que comparemos.

OPERADOR	SIGNIFICADO
==	Igual a
!=	No es igual a
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que

Tabla 3.3. Operadores de comparación.

Vamos a probar algunos de ellos:

```
>>> 42 == 42
True
>>> 2 != 3
True
>>> 2 == 3
False
>>>
>>> 'hola' == 'hola'
True
>>> 'hola' == 'Hola'
False
>>> 42 == 42.0
True
>>> 42 == '42'
False
```

Como cabía esperar, == (igual a) se evalúa a `True` cuando los valores a ambos lados son iguales, y `!=` (no es igual a) se evalúa a `True` cuando los dos valores son distintos. Los operadores `==` y `!=` pueden usarse con valores de cualquier tipo de datos. Por ejemplo, la expresión `42 == '42'` se evalúa a `False` porque Python considera que el entero `42` es diferente de la cadena `'42'`.

Por su parte, los operadores <, >, <=, y >= solo funcionan correctamente con valores enteros y flotantes:

```
>>> 42 < 100
True
>>> 42 > 100
False
>>> 42 < 42
False
>>> miEdad = 28
>>> miEdad <= 28
True
>>> puntos = 20
>>> puntos >= 10
True
```

A menudo usaremos operadores de comparación para comparar el valor de una variable con algún otro valor, como en las expresiones `miEdad <= 28` y `puntos >= 10`.

2.4. OPERADORES BOOLEANOS.

Los valores Booleanos `True` y `False` no se pueden sumar, ni multiplicar, ni concatenar (eso son peraciones propias de valores ennteros, flotantes, y cadenas). Las únicas operaciones que podemos efectuar sobre los

valores Booleanos son las operaciones `and`, `or`, y `not`. Al igual que los operadores de comparación, estos operadores Booleanos evalúan las expresiones que los contienen a un valor Booleano único.

LOS OPERADORES AND Y OR.

Los operadores `and` (y) y `or` (o) siempre reciben dos valores Booleanos (o dos expresiones que se reduzcan a un valor Booleano).

El operador `and` evalúa una expresión a `True` si los dos valores Booleanos que opera son `True`; en otro caso, se evalúa a `False`:

```
>>> True and True
True
>>> True and False
False
>>> False and True
False
```

Por su parte, el operador `or` evalúa una expresión a `True` si alguno de los dos valores Booleanos que opera es `True`. Si los dos son `False`, se evalúa a `False`:

```
>>> False or True
True
>>> True or True
True
>>> False or False
False
```

Algunos ejemplos prácticos de uso de los operadores Booleanos serían los siguientes:

```
>>> necesitoCamisa = True
>>> tengoDinero = True
>>> comproCamisa = necesitoCamisa and tengoDinero
>>> comproCamisa
True

>>> piesPlanos = True
>>> cortoTalla = False
>>> eximeServicioMilitar = piesPlanos or cortoTalla
>>> eximeServicioMilitar
True
```

EL OPERADOR NOT.

Al contrario que los operadores `and` y `or`, el operador `not` solo actúa sobre un valor Booleano (o una única expresión Booleana). El operador `not` simplemente se evalúa al opuesto del valor Booleano proporcionado.

```
>>> not True
False
>>> not False
True
```

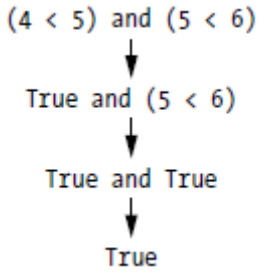
2.5. COMBINAR OPERADORES BOOLEANOS Y DE COMPARACIÓN.

Como los operadores de comparación se evalúan a valores Booleanos, podemos usarlos en expresiones que incluyan operadores Booleanos. Recordemos que los operadores Booleanos `and`, `or`, y `not` siempre operan

sobre los valores Booleanos `True` y `False`. Aunque expresiones como `4 < 5` no son valores Booleanos propiamente dichos, sí que son expresiones que se reducen a valores Booleanos. Escribe en el shell interactivo las siguientes expresiones Booleanas con operadores de comparación:

```
>>> (4 < 5) and (5 < 6)
True
>>> (4 < 5) and (9 < 6)
False
>>> (1 == 2) or (2 == 2)
True
```

A modo de ejemplo, la figura muestra el proceso de evaluación de la expresión `(4 < 5) and (5 < 6)`.



También podemos usar múltiples operadores Booleanos en una expresión, junto con los operadores de comparación:

```
>>> (2 + 2 == 4) and not (2 + 2 == 5) and (2 * 2 == 2 + 2)
True
```

Como en los operadores matemáticos, los operadores Booleanos se rigen por una precedencia: Después de que se evalúen todos los operadores matemáticos y de comparación, Python evalúa los operadores `not` en primer lugar, luego los operadores `and`, y finalmente los operadores `or`.

La combinación de los operadores de comparación con los operadores Booleanos nos pwrmitirán construir condiciones más complejas que nos otroguen un mayor grado de control de las sentencias de control de flujo. Algunos ejemplos prácticos de uso son los siguientes:

```
>>> notaMedia = 9.4
>>> negativos = 0
>>> matriculaHonor = (notaMedia >= 9) and (negativos == 0)

>>> edad = 25
>>> haVotado = False
>>> puedeVotar = (edad >= 18) and (haVotado == False)

>>> precio = 35
>>> color = 'Rojo'
>>> comproCamisa = (precio <= 30) and not(color == 'Blanco')

>>> destino = 'Roma'
>>> voyViaje = (destino == 'Paris') or (destino == 'Londres')
```

2.6. ESTRUCTURA DE LAS SENTENCIAS DE CONTROL DE FLUJO.

Las **sentencias de control de flujo** suelen comenzar con una parte llamada **condición**, y siguen con un **bloque de código**. Antes de estudiar las sentencias de control de flujo, vamos a explicar qué son las condiciones y los bloques de código.

CONDICIONES.

Todas las expresiones Booleanas que hemos visto hasta ahora pueden considerarse **condiciones**. Las condiciones no son más que expresiones que siempre se evalúan a un valor Booleano, `True` o `False`. Una sentencia de control de flujo decide qué hacer basándose en si su condición se evalúa a `True` o a `False`.

BLOQUES DE CÓDIGO.

Las líneas de código Python pueden agruparse en *bloques*. Podemos saber dónde empieza y termina un bloque mediante la tabulación de sus líneas de código.

Hay tres reglas que debemos seguir a la hora de construir bloques:
1) Los bloques comienzan allí donde la tabulación aumenta.
2) Los bloques pueden contener otros bloques.
3) Los bloques terminan allí donde la tabulación disminuye a cero, o a la tabulación del bloque que los contiene.

A modo de ejemplo, veamos si podemos identificar los bloques en el siguiente programa:

```
print('Escribe tu nombre:')
nombre = input()
print('Inserta tu contraseña:')
contraseña = input()

if nombre == 'Alex':
    print('Hola Alex.')
    if contraseña == '12345':
        print('Acceso concedido.')
    else:
        print('Contraseña errónea.')
else:
    print('Hola desconocido.')
```

Tras haber visto su estructura general, vamos a estudiar las distintas sentencias de control de flujo que incluye Python.

2.7. SENTENCIAS DE CONTROL DE FLUJO.

SENTENCIAS IF.

Uno de los tipos más comunes de sentencias de control de flujo son las sentencias `if`. El bloque de código que sigue a la sentencia `if` se ejecutará si la condición se evalúa a `True`. Por el contrario, el bloque no se ejecutará si la condición se evalúa a `False`. En palabras, como el término inglés "if" es el equivalente al "si" condicional en castellano, una sentencia `if` puede leerse como: "Si esta condición es cierta, ejecuta las instrucciones del siguiente bloque de código". A modo de ejemplo, escribe el siguiente programa en el shell interactivo:

```
print('Hola. Dime tu nombre.')
nombre = input()

if (nombre == 'Alex'):
    print('Hola Alex.')

print('Adiós.')
```

Todas las sentencias de control de flujo incluyen dos puntos (:) tras la condición, que sirven para indicar dónde empieza el bloque de código que contienen. El bloque de código de esta sentencia `if` es la instrucción `print('Hola, Alex.')`.

SENTENCIAS ELSE.

El bloque de código de una sentencia `if` puede venir seguido (opcionalmente) por una sentencia `else`. El bloque de código de la sentencia `else` solo se ejecuta cuando la condición de la sentencia `if` se evalúa a `False`. En palabras, como "else" significa "si no", una estructura `if - else` puede leerse como: "Si esta condición es cierta, ejecuta este código. Si no, ejecuta este otro código". Una sentencia `else` no tiene una condición propia (depende de la condición de su sentencia `if` asociada). Siguiendo con el ejemplo previo:

```
print('Hola. Dime tu nombre.')
nombre = input()

if (nombre == 'Alex'):
    print('Hola Alex.')
else:
    print('No eres Alex.')
    print('Hola, desconocido.')

print('Adiós.')
```

SENTENCIAS ELIF.

Mientras que en las sentencias `if` y `else` solo se ejecuta uno de los bloques de código (el del `if` o el del `else`), en ocasiones tal vez necesitemos que se ejecute un bloque de entre *muchos* posibles. La sentencia `elif` es una sentencia "else if" que siempre sigue a una sentencia `if` o a otra sentencia `elif`. La sentencia `elif` proporciona otra condición que solo se comprueba si alguna de las condiciones previas fue `False`. Vamos a cambiar el ejemplo previo añadiendo un `elif` para ver esta sentencia en acción:

```
print('Hola. Dime tu nombre.')
nombre = input()
print('¿Cuántos años tienes?')
edad = int(input())

if (nombre == 'Alex'):
    print('Hola Alex.')
elif (edad < 12):
    print('Tú no eres Alex, jovencito.')

print('Adiós.')
```

Esta vez, comprobamos la edad del usuario, y el programa le dirá algo diferente si tiene menos de 12 años y no es Alex. Esto es, el bloque de código del `elif` se ejecuta si `edad < 12` es `True` y `nombre == 'Alex'` es `False`. Sin embargo, si ambas condiciones son `False`, no se ejecuta ninguno de los bloques. No hay garantías de que se vaya a ejecutar al menos uno de los bloques. Cuando tenemos una cadena de sentencias `elif`, solo se ejecutará uno o ninguno de los bloques. En cuanto una de las condiciones de las distintas sentencias se evalúa a `True`, automáticamente se ignoran el resto de sentencias `elif`. A modo de segundo ejemplo, abre una nueva ventana en el editor de archivos y escribe el siguiente código. Guárdalo como `vampiro.py`:

```
print('Hola. Dime tu nombre.')
nombre = input()
print('¿Cuántos años tienes?')
edad = int(input())
```

```
if nombre == 'Alex':
    print('Hola, Alex.')
elif edad < 12:
    print('Tú no eres Alex, jovencito.')
elif edad > 1000:
    print('Tú no eres Alex, vampiro inmortal.')
elif edad > 100:
    print('Tú no eres Alex, abuelito.')
```

El orden de las sentencias `elif` es *crucial*. Vamos a reescribir el programa previo para verificar este hecho. Recordar que cuando el programa encuentra una condición que se evalúa a `True`, el resto de sentencias `elif` se omiten automáticamente. Por consiguiente, si intercambiamos algunas de los bloques de código del programa `vampiro.py`, nos encontraremos con problemas. (Guarda este programa modificado como `vampiro2.py`).

```
print('Hola. Dime tu nombre.')
nombre = input()
print('¿Cuántos años tienes?')
edad = int(input())

if nombre == 'Alex':
    print('Hola, Alex.')
elif edad < 12:
    print('Tú no eres Alex, jovencito.')
elif edad > 100:
    print('Tú no eres Alex, abuelito.')
elif edad > 1000:
    print('Tú no eres Alex, vampiro inmortal.')
```

Digamos que a la variable `edad` se le asigna el valor `3000` antes de que este código se ejecute. Puede que pensemos que el programa responderá `'Tú no eres Alex, vampiro inmortal.'`. Pero como la condición `edad > 100` se evalúa a `True` (porque `3000` es mayor que `100`), la cadena que se imprime por pantalla es `'Tú no eres Alex, abuelito.'`, y el resto de sentencias `elif` se omiten automáticamente. Es importante recordar que, como mucho, solo se ejecutará uno de los bloques de código, y que en las sentencias `elif` encadenadas el orden es muy importante.

Opcionalmente, podríamos añadir una sentencia `else` después de la última sentencia `elif`. En este caso queda garantizado que al menos uno (y solo uno) de los bloques de código se ejecutará. Si las condiciones en todos las sentencias `if` y `elif` son todas `False`, entonces se ejecutará el bloque del `else`. Por ejemplo:

```
print('Hola. Dime tu nombre.')
nombre = input()
print('¿Cuántos años tienes?')
edad = int(input())

if nombre == 'Alex':
    print('Hola, Alex.')
elif edad < 12:
    print('Tú no eres Alex, jovencito.')
elif edad > 1000:
    print('Tú no eres Alex, vampiro inmortal.')
elif edad > 100:
    print('Tú no eres Alex, abuelito.')
else:
    print('Tú no eres Alex, ni un jovencito, ni un abuelito, ni un vampiro.')
```

Recordar que en este tipo de estructuras el orden es importante. En primer lugar, siempre hay exactamente una sentencia `if`. En segundo lugar, todas las sentencias `elif` que necesitemos deben de

seguir al `if`. Por último y opcional, si queremos asegurarnos de que se ejecutará al menos un bloque de código, debemos cerrar la estructura con una sentencia `else`.

SENTENCIAS WHILE (BUCLES).

Con una sentencia `while` podemos hacer que un bloque de código se ejecute una y otra vez. El código dentro del `while` se estará ejecutando repetidamente mientras que la condición de ese `while` siga siendo `True`. Un ejemplo es el siguiente:

```
cuenta = 0
while cuenta < 5:
    print('Hola, mundo.')
    cuenta = cuenta + 1
print ('Adiós')
```

La estructura de un `while` es muy parecida a la de un `if`. La diferencia está en cómo se comporta. Al final del bloque de código de un `if`, la ejecución del programa continua con la siguiente instrucción tras la sentencia `if`. Pero al final del bloque de un `while`, la ejecución del programa salta de vuelta al principio de la sentencia `while`. Es por ello que a las sentencias `while` se las denomina **bucles**. Vamos a comparar el funcionamiento de la sentencia `while` previa con el de una sentencia `if` con la misma condición y el mismo bloque de código:

```
cuenta = 0
if cuenta < 5:
    print('Hola, mundo.')
    cuenta = cuenta + 1
print ('Adiós')
```

Ambos códigos son casi idénticos, pero cuando los ejecutamos ocurren cosas muy diferentes. Para el código con la sentencia `if`, la salida es simplemente la impresión de la cadena `'Hola, mundo.'` una única vez. Pero el código con la sentencia `while` imprime esa cadena cinco veces. Notar que el código con la sentencia `if` comprueba su condición, e imprime la cadena `'Hola, mundo.'` una sola vez si esa condición se evalúa a `True`. Por otro lado, el bucle `while` comprueba su condición y repite la ejecución de su bloque de código mientras la condición siga siendo `True`. Como el código dentro del `while` incrementa en una unidad el valor de `cuenta` en cada repetición del bucle, se imprimirá `'Hola, mundo.'` en cinco ocasiones hasta que la condición `cuenta < 5` se haga `False`, momento en el que el programa sale del bucle.

En un bucle `while` siempre se comprueba la condición al principio de cada **iteración** (esto es, cada vez que el bucle se ejecuta). Si la condición es `True`, el bloque de código se ejecuta, y a continuación, se vuelve a comprobar la condición. La primera vez que la condición se hace `False`, el bloque del `while` se omite, y el programa sale del bucle.

Como segundo ejemplo, vamos a escribir un programa que pide constantemente que escribas, literalmente, "tu nombre". Abre una nueva ventana del editor de archivos, y escribe el siguiente código. Guárdalo como `tuNombre.py`:

```
nombre = ''
while nombre != 'tu nombre':
    print('Por favor, escribe tu nombre.')
    nombre = input()
print('Gracias.')
```

Ejecuta el programa, y cuando te pregunte, escribe algo distinto a `tu nombre` unas cuantas veces antes de proporcionarle al programa lo que quiere:

```
Por favor, escribe tu nombre.
Alex
Por favor, escribe tu nombre.
Alejandro
Por favor, escribe tu nombre.
@#%&#!!
Por favor, escribe tu nombre.
tu nombre
Gracias.
```

Si nunca escribes `tu nombre`, la condición del bucle `while` nunca es `False`, y el programa seguirá preguntando para siempre. En este caso, la llamada a la función `input()` le permite al usuario insertar la respuesta apropiada para hacer que el programa salga del bucle. Pero en otros programas, la condición podría no cambiar nunca, y eso sería un problema. Vamos a ver cómo podemos forzar la salida de un bucle `while`.

SENTENCIAS BREAK.

En Python podemos hacer que la ejecución de un programa salga de un bucle `while` antes de tiempo: Cuando la ejecución del programa llega a una sentencia `break`, sale inmediatamente del bloque de código del bucle en el que se encuentra. A modo de ejemplo, aquí tenemos un programa que hace exactamente lo mismo que el programa previo, pero usando una sentencia `break` para escapar del bucle. Escribe este programa y guárdalo como `tuNombre2.py`.

```
while True:
    print('Por favor, escribe tu nombre.')
    nombre = input()
    if nombre == 'tu nombre':
        break
print('Gracias.')
```

La línea (1) crea un **bucle infinito**. Se trata de un bucle `while` cuya condición siempre es `True`. El programa siempre entrará en el bucle y solo saldrá cuando se ejecute la sentencia `break`. (CUIDADO: Un bucle infinito del que *nunca* se sale es un error muy común en programación). En la línea (2), el programa te pide que escribas, literalmente, `tu nombre`. Pero en este caso, mientras la ejecución del programa todavía está dentro del bucle `while`, se ejecuta una sentencia `if` para comprobar si la variable `nombre` contiene la cadena `tu nombre`. Si esta condición es `True`, se ejecuta la sentencia `break`, y la ejecución sale del bucle a la instrucción `print('Gracias.')`. En caso contrario, se omite el bloque del `if` que contiene el `break`. En este punto, la ejecución del programa vuelve al principio de la sentencia `while`, línea (1), para comprobar de nuevo la condición. Como esta condición siempre es `True`, la ejecución vuelve a entrar en el bucle para pedirte otra vez que escribas `tu nombre`.

SENTENCIAS CONTINUE.

Como las sentencias `break`, las sentencias `continue` siempre se usan dentro de bucles. Cuando se llega a una sentencia `continue`, la ejecución del programa vuelve inmediatamente al principio del bucle, y reevalúa la condición del bucle. (Esto es lo mismo que pasa cuando la ejecución llega al final del bucle). Vamos a usar la sentencia `continue` para escribir un programa que le pida al usuario un nombre y una contraseña.

Escribe el siguiente programa en el editor de archivos y guárdalo como `contraseña.py`.

```
while True:
    print('Hola. ¿Quién eres?')
    nombre = input()
    if nombre != 'Alex':
        continue
    print('Hola Alex. Por favor, escribe tu contraseña.')
    contraseña = input()
    if contraseña == '12345':
        break
print('Acceso concedido.')
```

Si el usuario escribe un nombre que no sea `Alex`, la sentencia `continue` hace que la ejecución del programa salte de vuelta al principio del bucle. Como la condición del bucle es simplemente el valor `True`, la ejecución siempre entra en el bucle. Cuando el usuario responde `Alex`, se le pide una contraseña. Si la contraseña proporcionada es `12345`, se ejecuta la sentencia `break`, y la ejecución sale del bucle `while` para imprimir por pantalla `Acceso concedido`. En otro caso, la ejecución continúa hasta el final del bucle `while`, desde donde salta atrás al principio del bucle. Vamos a ejecutar el programa y a darle algunas respuestas. Hasta que el usuario no diga que es Alex, no le pedirá una contraseña, y hasta que no introduzca la contraseña correcta, no le dará acceso:

```
Hola. ¿Quién eres?
Ana
Hola. ¿Quién eres?
Susana
Hola. ¿Quién eres?
Alex
Hola Alex. Por favor, escribe tu contraseña.
abcde
Hola. ¿Quién eres?
Alex
Hola Alex. Por favor, escribe tu contraseña.
12345
Acceso concedido.
```

BUCLES FOR Y LA FUNCIÓN RANGE().

El bucle `while` continúa iterando mientras su condición se evalúe a `True`. Pero, ¿y si queremos ejecutar un bloque de código únicamente un número de veces determinado? Esto podemos hacerlo mediante un bucle `for` y la función `range()`. Vamos a escribir un nuevo programa llamado `cinco.py` para ver al bucle `for` en acción:

```
for i in range(5):
    print('Repetición número ' + str(i))
```

El `print()` del bloque dentro del bucle `for` se ejecuta cinco veces. La primera vez que se ejecuta, la variable `i` toma el valor `0`, y la llamada `print()` imprimirá `Repetición número 0`. Después de que Python termine una iteración a través de todo el bloque de código de la sentencia `for`, la ejecución retorna al principio del bucle, y la sentencia `for` incrementa el valor de `i` en una unidad. Ésta es la razón por la que la llamada `range(5)` produce cinco iteraciones a través del bloque de código, ajustando `i` inicialmente a 0, después a 1, luego a 2, a 3, y finalmente a 4. La variable `i` llegará, pero no incluirá, el entero que se le pasa a la función `range()`.

La salida del programa es la siguiente:

```
Repetición número 0
Repetición número 1
Repetición número 2
Repetición número 3
Repetición número 4
```

También podemos usar las sentencias `break` y `continue` dentro de los bucles `for`. La sentencia `continue` continuará hasta el siguiente valor del contador del bucle. CUIDADO: Solo podemos usar las sentencias `continue` y `break` dentro de los bucles `while` y `for`. Si intentamos usar estas sentencias en cualquier otro lado, Python nos dará un mensaje de error. A modo de segundo ejemplo, aquí tenemos un programa que hace la suma de los números naturales desde el 0 hasta el 100 usando un bucle `for`:

```
total = 0
for num in range(101):
    total = total + num
print(total)
```

El resultado debería ser 5050. Cuando el programa comienza, el valor de la variable `total` se fija inicialmente a 0 (línea 1). Entonces el bucle `for` (línea 2) ejecuta la instrucción `total =total+num` (línea 3) un total de 100 veces. Cuando el bucle ha terminado las 100 iteraciones, cada número entero desde 0 hasta 100 se habrá sumado a `total`. Llegados a este punto, el programa imprime `total` por pantalla (línea 4).

Ahora vamos a modificar este programa para permitirle al usuario elegir hasta que número quiere sumar. Esto es, si el usuario elige el 250, el programa sumará todos los números naturales desde el 0 hasta el 250. El programa es el siguiente:

```
total = 0
print('¿Hasta qué número quieres sumar?')
limite = int(input())

for num in range(limite+1):
    total = total + num

print('El resultado final es:')
print(total)
```

Argumentos de la función range().

Algunas funciones pueden llamarse con múltiples argumentos separados mediante comas, y la función `range()` es una de ellas. Esta opción nos permite personalizar la secuencia de enteros que genera esta función. A modo de ejemplo, escribe el siguiente programa:

```
for i in range(12, 16):
    print(i)
```

El primer argumento indica el primer valor que toma la variable del bucle `for`, y el segundo indica el número al que llega, sin incluirlo. La salida de este programa es:

```
12
13
14
15
```

A modo de ejercicio, podemos intentar modificar el programa de la suma de enteros para permitirle al usuario elegir el entero inicial y el entero final entre los que se realizará la suma.

También podemos llamar a la `range()` con tres argumentos. Los dos primeros argumentos son los valores de inicio y parada, y el tercero es el **valor de incremento**. El incremento es la cantidad en la que la variable se incrementará tras cada iteración. Escribe este programa para ver cómo funciona:

```
for i in range(0, 10, 2):
    print(i)
```

La llamada `range(0,10,2)` contará desde cero hasta ocho en pasos de dos:

```
0
2
4
6
8
```

La función `range()` es flexible en la secuencia de números que produce para los bucles `for`. Por ejemplo, incluso podemos usar un número negativo para el argumento de incremento para hacer que el bucle cuente hacia atrás en vez de hacerlo hacia adelante:

```
for i in range(5, -1, -1):
    print(i)
```

Este programa debería obtener una cuenta atrás desde cinco hasta cero:

```
5
4
3
2
1
0
```

2.8. EL MÓDULO RANDOM.

En la sección 3.1 aprendimos a importar funciones integradas de Python con la sentencia `import`, e importamos el módulo `math` para acceder a algunas funciones matemáticas, como la raíz cuadrada. Otro módulo muy útil es el módulo `random`, el cual nos da acceso a la función `random.randint()`. Escribe este programa en el editor de archivos, y guárdalo como `random.py`:

```
import random
for i in range(5):
    print(random.randint(1, 10))
```

Cuando ejecutamos este programa, la salida se parecerá a lo siguiente:

```
3
1
2
10
7
```

La llamada a la función `random.randint()` se evalúa a un valor entero aleatorio entre los dos enteros que le pasemos. Como `randint()` está contenida en el módulo `random`, debemos escribir `random.randint()` para indicarle a Python que busque esta función dentro del módulo `random`.

2.9. EL MÓDULO SYS Y LA FUNCIÓN SYS.EXIT().

El último concepto de control de flujo que nos queda por ver es cómo terminar un programa. Esto siempre ocurre cuando la ejecución del programa llega a la última instrucción. Sin embargo, podemos forzar la finalización del programa llamando a la función `sys.exit()`. Como esta función está en el módulo `sys`, primero debemos importarlo. A modo de ejemplo, escribe el siguiente programa en el editor de archivos y guárdalo como `ejemploExit.py`:

```
import sys
while True:
    print('Escribe "fin" para salir.')
    respuesta = input()
    if respuesta == 'fin':
        print('Programa terminado.')
        sys.exit()
    print('Has escrito ' + respuesta + '.')
```

Este programa usa un bucle infinito que no incluye una sentencia `break` dentro de él. La única forma de terminar el programa es que el usuario escriba `fin`, lo que implicará una llamada a la función `sys.exit()`. Cuando el usuario escriba `fin` y la variable `respuesta` tome este valor, el programa termina:

```
Escribe "fin" para salir.
hola
Has escrito hola.
Escribe "fin" para salir.
adiós
Has escrito adiós.
Escribe "fin" para salir.
termina
Has escrito termina.
Escribe "fin" para salir.
fin
Programa terminado.
```

2.10. PROGRAMA DE EJEMPLO: ADIVINA EL NÚMERO.

Para terminar con el control de flujo, vamos a escribir un programa que combine varios de los conceptos que hemos aprendido a lo largo de esta sección. Este programa será un juego de adivinar un número secreto. La salida del programa será similar a la siguiente:

```
Estoy pensando en un número entre el 1 y el 20.
Adivina el número.
5
Tu número es demasiado pequeño.
Adivina el número.
15
Tu número es demasiado grande.
Adivina el número.
10
Tu número es demasiado pequeño.
Adivina el número.
13
Tu número es demasiado grande.
Adivina el número.
11
Acertaste! Has necesitado 5 intentos.
```

Escribe el siguiente código en el editor de archivos y guárdalo como `adivinaNumero.py`:

```
# Este programa es un juego de adivinar un número secreto.

import random
numeroSecreto = random.randint(1, 20)
print('Estoy pensando en un número entre el 1 y el 20.')

# Le damos al usuario 6 oportunidades para adivinar el número.
for numIntentos in range(1, 7):
    print('Adivina el número.')
    apuesta = int(input())

    if apuesta < numeroSecreto:
        print('Tu número es demasiado pequeño.')
    elif apuesta > numeroSecreto:
        print('Tu número es demasiado grande.')
    else:
        break # El usuario ha acertado.

# Salimos del for porque hemos acertado o porque hemos agotado los 6 intentos.
if apuesta == numeroSecreto:
    print('Acertaste! Has necesitado ' + str(numIntentos) + ' intentos.')
else:
    print('Agotaste tus 6 oportunidades!. El número era ' + str(numeroSecreto))
```

Vamos a analizar este código desde el principio. En primer lugar tenemos un comentario que explica para qué sirve el programa (algo que siempre deberíamos hacer). A continuación, el programa importa el módulo `random` para poder usar la función `random.randint()` y generar el número que el usuario ha de adivinar. El valor de retorno, un número entero aleatorio entre 1 y 20, se almacena en la variable `numeroSecreto`.

A continuación el programa le dice al usuario que ha pensado un número secreto, y le da seis oportunidades para adivinarlo. El código que permite al usuario insertar un número y que comprueba si es correcto está dentro de un bucle `for` que iterará un máximo de 6 veces. Lo primero que ocurre en el bucle es que el jugador escribe su apuesta. Como la función `input()` siempre devuelve una cadena, convertimos la respuesta del usuario a un entero con la función `int()`. Este valor se almacena en una variable llamada `apuesta`.

Las siguientes líneas de código sirven para comprobar si el número propuesto por el usuario es mayor o menor que el número secreto. En ambos caso, el programa imprime una pista por pantalla. Si el número propuesto por el usuario no es ni mayor ni menor que el número secreto, es porque debe ser igual al número secreto, en cuyo caso queremos que la ejecución del programa salga del bucle `for` con la sentencia `break`.

Finalmente, y al salir del bucle `for`, la sentencia `if` comprueba si el programa ha salido del bucle porque el usuario ha adivinado correctamente el número secreto, o porque ha agotado las 6 oportunidades de las que disponía. Dependiendo del caso, el programa muestra un mensaje que indica cuántos intentos necesitó el usuario para adivinar el número secreto, o un mensaje que le dice cuál era el número que no consiguió adivinar. (Estos valores se guardan en las variables `numIntentos` y `numeroSecreto`, respectivamente). Como en ambos casos debemos concatenar estos valores enteros con unas cadenas para poder imprimirlos, el programa los convierte en cadenas con la función `str()`.

2.11. EJERCICIOS.

EJERCICIO 5. Mayor de edad. Escribe un programa que le pida al usuario su nombre y su edad. En base a la edad, el programa le indica personalmente al usuario (dirigiéndose a él por su nombre) si es mayor de edad o menor de edad. Guarda el programa como `Ejer5.py`.

EJERCICIO 6. Par o impar. Escribe un programa que le pida al usuario un número entero cualquiera. El programa deberá determinar si ese número es par o impar, e indicarlo por pantalla. (Pista: Para saber si un número es par o impar, debes saber si es divisible entre dos. ¿Cómo puedes comprobar si un número es divisible por otro?). Guarda el programa como `Ejer6.py`.

EJERCICIO 7. Saludos. Escribe un programa que le pida al usuario su nombre y un número entero. El programa saludará al usuario tantas veces como haya indicado. Guarda el programa como `Ejer7.py`. Un ejemplo de ejecución del programa es el siguiente:

```
Dime tu nombre.
Alejandro
¿Cuántas veces quieres que te salude?
8

Hola Alejandro por 1ª vez.
Hola Alejandro por 2ª vez.
Hola Alejandro por 3ª vez.
Hola Alejandro por 4ª vez.
Hola Alejandro por 5ª vez.
Hola Alejandro por 6ª vez.
Hola Alejandro por 7ª vez.
Hola Alejandro por 8ª vez.

Adiós!
```

EJERCICIO 8. Tabla de multiplicar. Escribe un programa que le pida al usuario un número del 1 al 10, y que imprima por pantalla la tabla de multiplicar del número elegido. Guarda el programa como `Ejer8.py`. Por ejemplo, si el usuario elige el 6, el programa imprimirá por pantalla algo parecido a lo siguiente:

```
Elige un número del 1 al 10 para imprimir su tabla de multiplicar:
6

La tabla del 6 es:

6 x 1 = 6
6 x 2 = 12
6 x 3 = 18
6 x 4 = 24
6 x 5 = 30
6 x 6 = 36
6 x 7 = 42
6 x 8 = 48
6 x 9 = 54
6 x 10 = 60
```

EJERCICIO 9. Suma impares. Escribe un programa que muestre por pantalla todos los impares desde 1 hasta un cierto número entero introducido por el usuario. (Por ejemplo, si el usuario inserta el 8, el programa debe mostrar los impares 1, 3, 5, y 7). Además, el programa debe hacer y mostrar la suma de todos estos impares. Guarda el programa como `Ejer9.py`.

EJERCICIO 9b. Serie de sumas. En física y matemáticas es habitual tener que evaluar sumas con grandes cantidades de términos. En ciertas situaciones, es conveniente usar un bucle para evaluar dichas sumas. Por ejemplo, suponer que queremos conocer el valor de la suma:

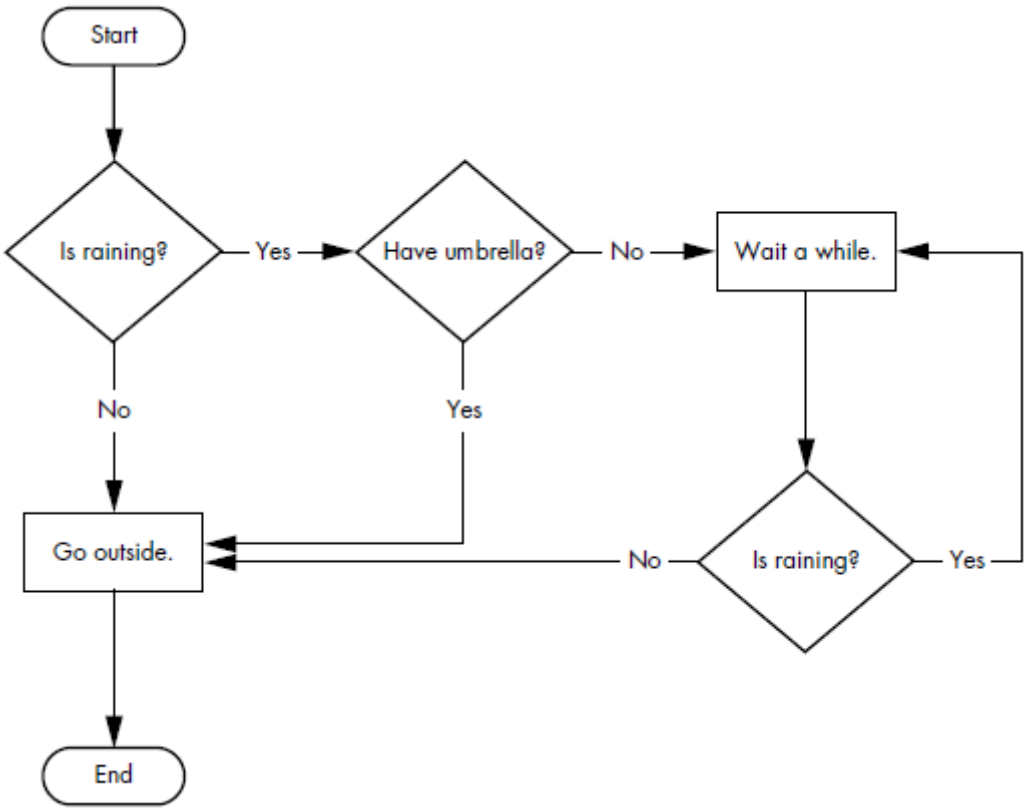
$$s = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{100} \equiv \sum_{k=1}^{100} \frac{1}{k}$$

, donde hemos aprovechado para presentar la notación basada en sumatorios (∑), que nos permite escribir sumas de muchos términos (incluso infinitos) de forma ágil y compacta. Escribe un programa que evalúe la suma de los n primeros términos de la serie $\sum_{k=1}^{n} 1/k$, donde n es un número configurable por el usuario. Guarda el programa como `Ejer9b.py`.

EJERCICIO 10. Quiniela. Escribe un programa que rellene automáticamente una quiniela de 10 partidos. Para ello, el programa debe sacar un total de 10 números aleatorios entre el 0 y el 2, donde un 0 significa empate (X), un 1 representa una victoria local, y un 2 indica una victoria visitante. Guarda el programa como `Ejer10.py`.

EJERCICIO 11. Conversor de notas. Escribe un programa en el que el usuario pueda escribir una nota del 0 al 10. En función de la nota, el programa indicará MUY DEFICIENTE si la nota es inferior o igual a 3, INSUFICIENTE si la nota es menor que 5, SUFICIENTE si la nota es mayor o igual a cinco y menor que seis, BIEN si la nota es mayor o igual a seis y menor que siete, NOTABLE si la nota es mayor o igual que siete y menor que nueve, y SOBRESALIENTE si la nota está entre nueve y diez, ambos incluidos. Guarda el programa como `Ejer11.py`.

EJERCICIO 12. Lloviendo. Escribe un programa que implemente el diagrama de flujo que permite decidir qué hacer si está lloviendo (ver figura 2.1). Guarda el programa como `Ejer12.py`.



EJERCICIO AMPLIACIÓN. Series de Taylor.
Las sumas infinitas suelen aparecer de forma muy frecuente en matemáticas. Por ejemplo, las funciones seno, coseno, y exponencial pueden calcularse como una suma infinita de potencias (las llamadas **series de Taylor**):

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

, donde $3!$ ("3 factorial") es $3! = 3 \times 2 \times 1$; $5!$ ("5 factorial") es $5! = 5 \times 4 \times 3 \times 2 \times 1$; etc. El cálculo del factorial de un número arbitrario n, esto es, $n! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1$, puede hacerse usando la función `math.factorial(n)` del módulo `math`.

Para igualar el desarrollo en serie de potencias con la función a la que representa, debemos usar un número infinito de términos (eso es lo que significan los puntos suspensivos en las fórmulas). Pero usando solo un número finito de términos, podemos obtener aproximaciones a dichas funciones. Por supuesto, cuantos más términos consideremos, mejor será la aproximación de la serie de potencias a la función real. Por ejemplo, si consideramos hasta el orden 3 (la tercera potencia), la función exponencial puede aproximarse como:

$$e^x \approx 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!}$$

(a) Escribe un programa que aproxime la función exponencial para un valor de x dado hasta la potencia que indique el usuario. Por ejemplo, si el usuario indica que quiere aproximar la función exponencial en $x = 1.75$ hasta el orden 3, el programa debería calcular:

$$1 + (1.75) + \frac{(1.75)^2}{2!} + \frac{(1.75)^3}{3!}$$

Además, el programa calculará el valor *exacto* de la función exponencial para ese valor de x (en el ejemplo previo, el valor de $e^{1,75}$), y obtendrá el error absoluto cometido en la aproximación, calculando el valor absoluto de la resta del valor real y el valor aproximado. (También puedes calcular el error relativo, dividiendo el error absoluto entre el valor real). Guarda el programa como `AmpTaylor1.py`.
(b) Haz un programa para aproximar la función seno para un valor de x dado hasta la potencia que indique el usuario. (Cuidado: Observa que el desarrollo en serie del seno solo tiene potencias y factoriales impares. Observa también que los términos son positivos y negativos alternativamente). Guarda el programa como `AmpTaylor2.py`.
(c) Haz lo mismo para el coseno. Guarda el programa como `AmpTaylor3.py`.

3. FUNCIONES.

3.1. DEFINIR Y LLAMAR A UNA FUNCIÓN.

A estas alturas ya deberíamos estar familiarizados con las funciones `print()`, `input()`, y `len()`. Python incluye muchas más funciones integradas, y otras funciones específicas dentro de sus múltiples módulos. Sin embargo, Python también nos permite escribir nuestras propias funciones. Una **función** es como un mini-programa dentro de un programa. Para entender mejor cómo funcionan las funciones, vamos a crear una. Escribe el siguiente programa en el editor de archivos y guárdalo como `saludo.py`:

```
# Definición de la función saludo().
def saludo():
    print('Hola, usuario!')
    print('¿Cómo estás?')

# Programa principal.
print('Inicio del programa.')
saludo()    # Llamada a la función saludo().
saludo()    # Otra llamada a la función saludo().
saludo()    # Una tercera llamada a la función saludo().
print('Fin del programa.')
```

La segunda línea es una sentencia `def` que define una función llamada `saludo()`. El bloque de código que sigue a la sentencia `def` es el cuerpo de la función. Aunque en Python es tradición escribir la definición de las funciones al principio del programa, su código solo se ejecutará cuando llamemos a la función desde el programa principal. Más abajo, las tres instrucciones `saludo()`son llamadas que hace el **programa principal** a esa función. En Python, una llamada a una función consiste simplemente en el nombre de la función seguido de unos paréntesis. Cuando la ejecución del programa llega a estas llamadas, el programa salta a la primera línea de la definición de la función, y empieza a ejecutar su código. Cuando llega al final de la función, la ejecución retorna a la línea que llamó a la función, y continúa avanzando por el código como antes. Como el programa llama a la función `saludo()` tres veces, el código dentro de la función se ejecuta tres veces. Cuando ejecutamos este programa, la salida que obtenemos es:

```
Inicio del programa.
Hola, usuario!
¿Cómo estás?
Hola, usuario!
¿Cómo estás?
Hola, usuario!
¿Cómo estás?
Fin del programa.
```

¿Para qué sirven las funciones? Uno de los propósitos fundamentales de las funciones es agrupar el código que debe ejecutarse múltiples veces. Si no hubiésemos definido esta función, tendríamos que copiar y pegar este código cada vez que quisiésemos ejecutarlo, y el programa se parecería a éste:

```
print('Inicio del programa.')
print('Hola, usuario!')
print('¿Cómo estás?')
print('Hola, usuario!')
print('¿Cómo estás?')
print('Hola, usuario!')
print('¿Cómo estás?')
print('Fin del programa.')
```

En general, siempre hay que evitar duplicar código, porque si alguna vez necesitamos modificarlo (por ejemplo, si resulta que tiene un error), tendremos que corregir el código en todos los sitios donde lo habíamos copiado.

3.2. PASAR DATOS A UNA FUNCIÓN: ARGUMENTOS Y PARÁMETROS.

Cuando llamamos a las funciones `print()` o `len()` les pasemos entre sus paréntesis ciertos datos que necesitan para poder funcionar. A los datos que necesitan recibir ciertas funciones para operar correctamente se les denomina **argumentos**. Python también permite definir funciones que acepten argumentos. Escribe el siguiente ejemplo en el editor de archivos y guárdalo como `saludo2.py`:

```
# Definición de la función saludo().
def saludo(nombre):
    print('Hola, ' + nombre + '!')
    print('¿Cómo estás?')

# Programa principal.
print('Inicio del programa.')
saludo('Rubén') # Llamada a la función saludo().
saludo('Susana')    # Otra llamada a la función saludo().
print('Ahora te voy a saludar a tí.')
print('Dime tu nombre:')
tuNombre = input()
saludo(tuNombre)
print('Fin del programa.')
```

Al ejecutar este programa, la salida que obtenemos es:

```
Inicio del programa.
Hola, Rubén!
¿Cómo estás?
Hola, Susana!
¿Cómo estás?
Ahora te voy a saludar a tí.
Dime tu nombre:
Alejandro
Hola, Alejandro!
¿Cómo estás?
Fin del programa.
```

En este programa, la definición de la función `saludo()` incluye entre sus paréntesis un parámetro llamado `nombre`. Un **parámetro** es una variable en la que se almacena el argumento que se le pasa a la función cuando se la llama desde el programa principal. La primera vez que se llama a la función `saludo()`, se hace con el argumento `'Rubén'`. La ejecución del programa salta a la función, y el valor que se almacena en la variable `nombre` se fija automáticamente a `'Rubén'`, que es lo que se imprime con la sentencia `print('Hola, ' + nombre + '!')`.

Una cosa que debemos recordar es que el valor almacenado en un parámetro se borra cuando la ejecución retorna de la función al programa principal. Por ejemplo, si añadiésemos una instrucción `print(nombre)` después de la llamada `saludo('Susana')`, el programa nos daría un error `NameError`, porque en el programa principal no hemos declarado una variable llamada `nombre`. Esta variable solo existe dentro de la función, y se destruye después de que el programa retorne de la llamada `saludo('Susana')`, por lo que la instrucción `print(nombre)` se referiría a una variable que no existe.

3.3. FUNCIONES QUE DEVUELVEN DATOS: LA SENTENCIA RETURN.

Cuando llamamos a la función `len()` y le pasamos como argumento la cadena `'Buenos días'`, la llamada a la función se evalúa al valor entero 11, que es la longitud de la cadena que le hemos pasado. En general, el valor al que se evalúa la llamada a una función se denomina **valor de retorno** de la función.

Al crear una función mediante una sentencia `def`, podemos especificar cuál es el valor que devuelve esa función usando una sentencia `return`. Cuando usamos una expresión con una sentencia `return`, el valor de retorno es aquel al que esa expresión se evalúa. A modo de ejemplo, el siguiente programa define una función para calcular el valor promedio de dos números enteros que se le pasan como argumentos. Escribe este código en el editor de archivos y guárdalo como `media2numeros.py`:

```
# Definición de la función promedio().
def promedio (num1, num2):
    return (num1 + num2) / 2

# Programa principal.
print('Escribe un número entero:')
a = int(input())
print('Escribe otro número entero:')
b = int(input())
media = promedio(a, b)
print('La media aritmética de los dos números es: ' + str(media))
```

Una forma alternativa de definir la función `promedio()` es la siguiente:

```
# Definición de la función promedio().
def promedio (num1, num2):
    ans = (num1 + num2) / 2
    return ans
```

3.4. PROGRAMA DE EJEMPLO: LA BOLA MÁGICA.

A modo de ejemplo, vamos a escribir el programa de una bola mágica. Una bola mágica es un juguete al que le haces una pregunta (¿Aprobaré el examen de matemáticas?), la agitas, y te da una respuesta (que a veces no es la que esperabas). Escribe este código en el editor de archivos y guárdalo como `bolaMagica.py`:

```
import random

# Definición de la función darRespuesta().

def darRespuesta(numRespuesta):
    if numRespuesta == 1:
        return 'Seguramente.'
    elif numRespuesta == 2:
        return 'Es muy probable.'
    elif numRespuesta == 3:
        return 'Sí.'
    elif numRespuesta == 4:
        return 'No lo veo claro. Pregunta otra vez.'
    elif numRespuesta == 5:
        return 'Vuelve a preguntar más tarde.'
    elif numRespuesta == 6:
        return 'Concéntrate y pregunta otra vez.'
```

```
    elif numRespuesta == 7:
        return 'Mi respuesta es no.'
    elif numRespuesta == 8:
        return 'Las perspectivas no son buenas.'
    elif numRespuesta == 9:
        return 'Lo dudo mucho.'

# Programa principal.

r = random.randint(1, 9)
suerte = darRespuesta(r)
print(suerte)
```

Nada más comenzar el programa, Python importa el módulo `random`. A continuación definimos la función `darRespuesta()`. Como aquí solo se estamos definiendo la función (no la estamos llamando), la ejecución se salta el código contenido dentro de ella y salta a la primera línea del programa principal. Allí, el programa principal llama a la función `random.randint()` pasándole dos argumentos, 1 y 9. Esta llamada se evalúa a un entero aleatorio entre 1 y 9 (ambos incluidos), y este valor se guarda en una variable llamada `r`. A continuación, el programa hace una llamada a la función `darRespuesta()` pasándole `r` como argumento. Entonces, la ejecución del programa salta al principio de la función `darRespuesta()`, y el valor `r` se almacena en un parámetro llamado `numRespuesta`. A continuación, dependiendo del valor de `numRespuesta`, la función devuelve una de las muchas cadenas posibles. La ejecución del programa retorna a la línea donde se hizo la llamada a la función `darRespuesta()`. La cadena devuelta se asigna a una variable llamada `suerte`, que a continuación, se le pasa como argumento a la función `print()` para que la muestre por pantalla.

Notar que como podemos pasar los valores de retorno de una función como argumentos de la llamada a otra función, en nuestro programa podríamos haber acortado las últimas tres líneas:

```
r = random.randint(1, 9)
suerte = darRespuesta(r)
print(suerte)
```

, en una sola línea:

```
print(darRespuesta(random.randint(1, 9)))
```

Recordar: Las expresiones están compuestas de valores y operadores. Como una llamada a una función se evalúa a su valor de retorno, podemos usar esa llamada dentro de una expresión como si de un valor se tratara.

3.5. EL VALOR NONE.

En Python hay un tipo de datos llamado `NoneType` cuyo único valor posible es el valor `None`. El valor `None` representa la ausencia de un valor de otro tipo.

¿Para qué sirve este valor sin valor? Cuando una función no necesita devolver ningún valor, su valor de retorno es el valor `None`. En efecto, siempre que tengamos una función que carezca de una sentencia `return`, Python añadirá en segundo plano una sentencia `return None` al final de la definición. A modo de ejemplo, vuelve a escribir el programa que calcula el promedio de dos números, pero esta vez borra la sentencia `return`:

```
# Definición de la función promedio().
def promedio (num1, num2):
    ans = (num1 + num2) / 2
```

```
# Programa principal.
print('Escribe un número entero:')
a = int(input())
print('Escribe otro número entero:')
b = int(input())
media = promedio(a, b)
print('La media aritmética de los dos números es: ')
print(media)
```

Si ejecutamos el programa, la salida será similar a esta:

```
Escribe un número entero:
7
Escribe otro número entero:
3
La media aritmética de los dos números es:
None
```

En efecto, como nuestra función no incluye una sentencia `return`, el valor que devuelve es `None`, lo que representa la usencia de un valor devuelto. Obviamente, el programa no funciona de forma correcta.

3.6. VARIABLES GLOBALES Y LOCALES.

Se dice que los parámetros y variables cuyo valor se asigna dentro de una función solo existen en el **ámbito local** de la función. Por el contrario, se dice que las variables cuyo valor se asigna fuera de las funciones de un programa existen en el **ámbito global** de ese programa. Una variable que solo existe en el ámbito local de la función donde se definió se denomina **variable local**, mientras que una variable que existe en el ámbito global del programa se denomina **variable global**. Toda variable debe ser de uno u otro tipo, no puede ser local y global simultáneamente.

Para entender todo esto, podemos imaginar que el ámbito es una especie de contenedor de variables. Cuando un ámbito se destruye, todas las variables almacenadas en ese ámbito se borran. Sólo hay un ámbito global, y éste se crea al arrancar el programa. Cuando el programa termina, el ámbito global se destruye, y todas sus variables se borran. Si esto no ocurriese, la siguiente vez que ejecutásemos el programa, las variables recordarían los valores que tenían la última vez que lo ejecutamos. Por otro lado, el ámbito local de una cierta función se crea cada vez que se llama a esa función. Todas las variables cuyo valor se asigne dentro de una función existen en el ámbito local de esa función. Cuando una función retorna, su ámbito local se destruye, y todas sus variables se borran. La siguiente vez que llamemos a esta función, las variables locales no recordarán los valores que almacenaban la última vez que llamamos a la función.

Los ámbitos son importantes por varias razones:
- Las variables locales no pueden usarse en el ámbito global.
- Un ámbito local no puede usar las variables de otros ámbitos locales.
- Sin embargo, los ámbitos locales pueden acceder a las variables globales (ya veremos cómo).

La razón por la que Python distingue entre diferentes ámbitos (en vez de hacer que todas las variables sean globales) es la siguiente: Las funciones solo interactúan con el resto del programa a través de sus parámetros y de su valor de retorno. Esto reduce enormemente el número de líneas de código que pueden estar causando un malfuncionamiento (bug). Si nuestro programa solo tuviese variables globales y causase un malfuncionamiento debido a que una variable toma un valor erróneo, sería muy difícil detectar en qué parte del programa se le asignó ese valor erróneo. Podría haber ocurrido en cualquier lugar del programa (y nuestro programa podría tener cientos o miles de líneas de código). Sin embargo, si el error se debe a una variable local con un valor erróneo, sabemos que el código que le asignó ese valor erróneo está en la función a la que pertenece esa variable local.

A modo de ejemplo, vamos a introducir adrede un bug en el programa `media2numeros.py`. Escribe lo siguiente en el editor de archivos (¿puedes ver el error que hemos introducido?):

```
# Definición de la función promedio().
def promedio (num1, num2):
    ans = num1 + num2 / 2
    return ans

# Programa principal.
print('Escribe un número entero:')
a = int(input())
print('Escribe otro número entero:')
b = int(input())
media = promedio(a, b)
print('La media aritmética de los dos números es: ' + str(media))
```

Al ejecutar el programa, la salida será similar a la mostrada:

```
Escribe un número entero:
7
Escribe otro número entero:
9
La media aritmética de los dos números es: 11.5
```

Esta salida es evidentemente errónea, porque la media de dos números enteros no puede ser mayor que ambos. ¿Dónde está el error? En este caso, es fácil acotar la zona del programa donde se produjo el malfuncionamiento. Evidentemente, el error está en la función que calcula la media, y es allí donde debemos buscarlo. En efecto, el error ha sido olvidar poner la suma de los dos números entre paréntesis, para efectuarla antes que la división entre dos.

Este ejemplo es solo una muestra. Imaginemos un programa con miles de líneas, en el que no hay funciones y donde todas las variables son globales. Tratar de buscar el origen de un malfuncionamiento en un programa así sería casi imposible. Ésta es una de las razones por la que existen los ámbitos locales y las funciones: Las funciones nos permiten compartimentar y aislar bloques de código, para facilitar la localización de malfuncionamientos en los programas.

Aunque usar variables globales en programas de tamaño reducido es una práctica aceptable, no es un buen hábito acostumbrarse a usar variables globales conforme nuestros programas se hagan más y más grandes.

LAS VARIABLES LOCALES NO PUEDEN USARSE EN EL ÁMBITO GLOBAL.

Consideremos el siguiente programa, que contiene un error:

```
def rellenaNombre():
    nombre = 'Alejandro'

#Programa principal
rellenaNombre()
print(nombre)
```

Si ejecutamos este programa, la salida se parecerá a esto:

```
Traceback (most recent call last):
  File "C:/Users/Alejandro/Desktop/1.py", line 5, in <module>
    print(nombre)
NameError: name 'nombre' is not defined
```

El error ocurre porque la variable `nombre` solo existe en el ámbito local que se crea cuando el programa llama a la función `rellenaNombre()`. Después de que la ejecución retorne de la función, ese ámbito local se destruye, y ya no hay una variable llamada `nombre`. Por lo tanto, cuando nuestro programa intenta ejecutar la instrucción `print(nombre)`, Python nos da un mensaje de error que indica que `nombre` no está definida. Ésta es la razón por la que solo pueden usarse variables globales en el ámbito global.

UN ÁMBITO LOCAL NO PUEDE USAR LAS VARIABLES DE OTROS ÁMBITOS LOCALES.

Siempre que se llama a una función se crea un nuevo ámbito local, incluso cuando una función llama a otra función. Consideremos este programa:

```
def imprimeDatos():
    nombre = 'Alejandro'
    edad = 15
    rellenaEdad()
    print(nombre)
    print(edad)

def rellenaEdad():
    edad = 29

# Programa principal.
imprimeDatos()
```

Nada más comenzar, el programa principal llama a la función `imprimeDatos()`, y se crea un ámbito local. La función inicializa la variable local `nombre` a `'Alejandro'`, y la variable local `edad` a `15`. A continuación, y dentro de la función, se llama a la función `rellenaEdad()`, y se crea un segundo ámbito local. Notar que pueden existir varios ámbitos locales al mismo tiempo. En este nuevo ámbito local, la variable `edad` se fija a `29`.

Cuando la función `rellenaEdad()` retorna, su ámbito local se destruye. La ejecución del programa continúa en la función `imprimeDatos()` para imprimir el valor de `nombre` y el de `edad`. ¿Qué pasa aquí? Como el único ámbito local que todavía existe es el de `imprimeDatos()`, la variable `edad` almacena el valor `15` (no el valor `29`), y ese será el dato que el programa imprimirá.

La conclusión es que las variables locales de una función están completamente separadas de las variables locales de otra función.

LOS ÁMBITOS LOCALES PUEDEN ACCEDER A LAS VARIABLES GLOBALES.

Consideremos el siguiente programa:

```
def imprimeNombre():
    print('Impresión desde el ámbito local:')
    print(nombre)

# Programa principal.
nombre = 'Susana'
imprimeNombre()
```

Notar que la función `imprimeNombre()` no tiene un parámetro llamado `nombre`, ni tampoco un código que le asigne a `nombre` un valor. Por lo tanto, cuando la función `imprimeNombre()` usa la variable `nombre`, Python considera que es una referencia a la variable global `nombre`, definida en el programa principal. Ésta es la razón por la que se imprime la cadena `'Susana'` cuando ejecutamos el programa.

3.7. LA SENTENCIA GLOBAL.

Si necesitamos modificar el valor de una variable global desde el código de una función, debemos usar una sentencia `global`. Si tenemos una línea de código como `global edad` al principio de una función, esta instrucción le dice a Python: "En esta función, `edad` se refiere a la variable global, así que no crees una variable local con este nombre". Por ejemplo, escribe el siguiente código en el editor de archivos y guárdalo como `global.py`:

```
def rellenaEdad():
    global edad
    edad = 66

# Programa principal.
edad = 25
rellenaEdad()
print(edad)
```

Cuando ejecutemos este programa, la llamada final a la función `print()` dará la siguiente salida:

```
66
```

Como al principio de la función `rellenaEdad()` la variable `edad` se declara como global, cuando la función asigna a `edad` el valor `66`, esta asignación se hace en el ámbito global. No se crea una variable local llamada `edad`.

Hay cuatro reglas para decidir si una variable está en un ámbito local o en el ámbito global:

1) Si una variable se está usando en el ámbito global (esto es, fuera de todas las funciones), entonces siempre se trata de una variable global.
2) Si en una función hay una sentencia `global` para una variable, se trata de una variable global.
3) En otro caso, si la variable se usa en una sentencia de asignación dentro de una función, es una variable local.
4) Pero si la variable no se usa en una sentencia de asignación, es una variable global (sin embargo, ver comentario más abajo).

Para hacernos una idea más precisa de estas reglas, veamos un ejemplo. Escribe este código en el editor de archivos y guáralo como `global2.py`:

```
def rellenaEdad():
    global edad     # Variable global.
    edad = 66

def indicaEdad():
    edad = 12       # Variable local.

def fijaEdad():
    print(edad)     # Variable global.

# Programa principal.
edad = 25       # Variable global.
rellenaEdad()
print(edad)
```

(1) En la función `rellenaEdad()`, `edad` es una variable global, porque hay una sentencia `global` para `edad` al principio de la función. (2) En la función `indicaEdad()`, `edad` es una variable local, porque hay una sentencia de asignación para ella dentro de la función. (3) En la función `fijaEdad()`, `edad` es una

variable global, porque dentro de la función no hay una sentencia de asignación o una sentencia `global` para esa variable. Si ejecutamos el programa, la salida será la siguiente:

```
66
```

3.8. EJERCICIOS.

EJERCICIO 13. **Suma y resta**. Escribe una función que reciba tres números, que haga la suma de los dos primeros, y a continuación, la resta del tercero. La función debe devolver el resultado de esta operación. Tras la definición de la función, en el programa principal, haz unas cuantas llamadas a la función con tres números distintos cada vez e imprime los resultados por pantalla, simplemente para comprobar si la función opera correctamente. Guarda el programa como `Ejer13.py`.

EJERCICIO 14. **Longitud combinada**. Escribe una función a la que se le pasen dos cadenas, y que devuelva la longitud de la cadena resultante de concatenar las dos cadenas. Tras la definición de la función, en el programa principal, haz unas cuantas llamadas a la función e imprime el resultado por pantalla, simplemente para comprobar si la función opera correctamente. Guarda el programa como `Ejer14.py`.

EJERCICIO 15. **Valor absoluto**. Como sabemos, la función integrada `abs()` nos permite obtener el **valor absoluto** de un número cualquiera. Pero imagina que Python no proporciona esta función; vamos a programarla nosotros. Para ello, escribe una función llamada `valorAbsoluto()` que reciba un parámetro llamado `number`. Escribe el código de esta función para que funcione como se espera. Tras la definición de la función, en el programa principal, haz unas cuantas llamadas a la función e imprime el resultado por pantalla, simplemente para comprobar si la función opera correctamente. Guarda el programa como `Ejer15.py`.

EJERCICIO 16. **Día semana**. Escribe una función a la que se le pase un número del 1 al 7, y que devuelva el día de la semana correspondiente. (Por ejemplo, si le pasamos un 1, el programa devuelve "lunes"; si le pasamos un 2, el programa devuelve "martes"; y si le pasamos un 7, el programa devuelve "domingo"). Haz que la función responda adecuadamente si el usuario no escribe un número correcto. (Por ejemplo, si el usuario escribe un 0 ó un 8, el programa debe indicar que se ha equivocado, y que escriba un número del 1 al 7). Tras la definición de la función, en el programa principal, haz unas cuantas llamadas a la función e imprime el resultado por pantalla, simplemente para comprobar si la función opera correctamente. Guarda el programa como `Ejer16.py`.

EJERCICIO 17. **La cadena más larga**. Escribe una función que reciba tres cadenas. La función debe devolver cuál de las tres cadenas es la más larga. Guarda el programa como `Ejer17.py`.

EJERCICIO 18. **Factorial**. El **factorial** de un número n se define como:

$$n! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1$$

El módulo `math` incluye una función para calcular el factorial de un número, a saber, `math.factorial()`. Sin embargo, imagina que esa función no está disponible. El problema consiste en escribir la función que nos permita calcular el factorial de un número entero (por ejemplo, $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$). Como sabrás, la operación factorial es muy importante en combinatoria y en el cálculo de probabilidades.

CUIDADO: Al calcular el factorial, hay ciertos casos especiales que el programa debe poder manejar; en particular, se cumple que:

$$0! = 1$$

Guarda el programa como `Ejer18.py`.

EJERCICIO 19. **Números combinatorios**. Basándote en la función del ejercicio previo, escribe una función que calcule la cantidad:

$$\binom{N}{n} \equiv \frac{N!}{n!\,(N-n)!}$$

Esta cantidad se denomina **número combinatorio**, y se lee "N sobre n". Por supuesto, la función debe recibir como argumentos dos enteros N y n, con $N > n$, y devolver un entero que sea el número combinatorio correspondiente. Como ya sabrás, el número combinatorio es una cantidad de interés en combinatoria y probabilidad. Guarda el programa como `Ejer19.py`.

EJERCICIO 20. **Probabilidad**. La probabilidad de que al lanzar una moneda (no trucada) n veces obtengamos k caras viene dada por:

$$P = \frac{casos\ favorables}{casos\ posibles} = \frac{\binom{n}{k}}{2^n}$$

Basándote en la función del ejercicio previo, escribe una función que reciba como parámetros los valores n y k, y que obtenga la probabilidad buscada. A modo de ejemplo, calcula la probabilidad de que, al lanzar la moneda 10 veces, obtengamos exactamente 6 caras. (El resultado debería ser 0,205, esto es, una probabilidad del 20,5 %). Guarda el programa como `Ejer20.py`.

EJERCICIO 21. **Triángulo de Pascal**. Usando la función del ejercicio 19, escribe un programa que imprima las 20 primeras líneas del "triángulo de Pascal". La línea n –ésima del triángulo de Pascal consta de $n+1$ números, que son los números combinatorios $\binom{n}{0}$, $\binom{n}{1}$, …, $\binom{n}{n}$. Por ejemplo, para obtener la primera línea deberías calcular $\binom{1}{0}$ y $\binom{1}{1}$; para la segunda línea $\binom{2}{0}$, $\binom{2}{1}$, y $\binom{2}{2}$; para la tercera $\binom{3}{0}$, $\binom{3}{1}$, $\binom{3}{2}$, y $\binom{3}{3}$, etc. Las 4 primeras líneas del triángulo de Pascal son:

```
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

Guarda el programa como `Ejer21.py`.

EJERCICIO 22. La secuencia de Collatz. Escribe una función llamada `collatz(number)` que reciba un parámetro llamado `number` (un número entero cualquiera). Si ese número es par, la función `collatz()` debe imprimir y devolver el resultado de la división entera de ese número entre 2. Pero si el número es impar, la función `collatz()` debe imprimir y devolver el resultado de multiplicar el número por 3 y sumarle 1.

A continuación, escribe un programa que le pida al usuario escribir un número entero, y que llame continuamente a la función `collatz()` sobre el resultado devuelto hasta que la función termine llegando al valor 1. (Sorprendentemente, esta secuencia funciona para cualquier entero. Usando esta secuencia, antes o después terminaremos obteniendo el valor 1. Incluso los matemáticos no tienen claro el por qué. Este programa explora la denominada **secuencia de Collatz**, a veces llamada "el problema imposible más fácil de las matemáticas"). La salida de este programa debería parecerse a lo siguiente:

```
Escribe un número entero:
3

Llamando a la secuancia de Collatz con los sucesivos resultados…

10
5
```

```
16
8
4
2
1
```

Guarda el programa como `Ejer22.py`.

EJERCICIO 23. Piedra, papel, o tijeras. En este ejercicio te proponemos programar el juego de piedra, papel o tijeras, en el que el usuario jugará contra el ordenador. Recuerda las reglas:

- La piedra vence a las tijeras (porque las machaca).
- Las tijeras vencen al papel (porque lo corta).
- El papel vence a la piedra (porque la envuelve).

El funcionamiento es el siguiente: En primer lugar, el programa le pide al usuario que elija entre `piedra`, `papel`, o `tijeras`. (El programa debe asegurarse de que el usuario responde una de estas tres opciones, y en caso contrario, le pide que vuelva a elegir). A continuación, el ordenador obtiene al azar uno de estos tres resultados, lo muestra por pantalla, compara ambas apuestas, e imprime quién ha ganado y por qué. Finalmente, el programa le pregunta al usuario si quiere jugar otra partida y actúa en consecuencia. Si hay un empate (digamos, dos tijeras) no tenemos ganador, y el programa debe volver a comenzar hasta que haya un vencedor. Guarda el programa como `Ejer23.py`.

PISTA: Te conviene crear funciones para que el programa sea más fácil de escribir y de leer. Algunos ejemplos podrían ser la función para obtener la apuesta aleatoria del ordenador, o la función que permite comparar las apuestas del usuario y del ordenador para determinar quién ha ganado.

EJERCICIO AMPLIACIÓN. Combinatoria.
La **combinatoria** es la rama de las matemáticas que estudia las diversas formas de realizar agrupaciones con los elementos de un conjunto, y de contar el número de agrupaciones posibles. Existen diversas formas de realizar agrupaciones, dependiendo de si se pueden repetir o no los elementos, de si se pueden tomar todos los elementos de los que disponemos, y de si influye o no en orden de colocación de esos elementos. En función de todo ello, podemos distinguir tres tipos de agrupaciones: **variaciones** (con y sin repetición), **permutaciones** (con y sin repetición), o **combinaciones**. Suponer que tenemos N elementos disponibles, y que tomamos n de esos elementos.

> NOTA: Para un repaso de la combinatoria, puedes visitar la página web de vitutor, https://www.vitutor.com/pro/1/analisis_combinatorio.html, o releer el tema de combinatoria en tu libro de matemáticas de 4º ESO.

Para poder determinar el tipo de agrupación que tenemos en un problema en particular, hemos de responder a estas preguntas:

¿Importa el orden de colocación de los elementos?
Sí ⇒ variaciones o permutaciones.
No ⇒ combinaciones.

¿Tomamos todos los elementos disponibles, o solo algunos?
Cogemos todos ⇒ permutaciones ($N = n$).
Tomamos solo algunos ⇒ variaciones.

¿Se pueden repetir los elementos?
Sí ⇒ variaciones o permutaciones con repetición.
No ⇒ variaciones o permutaciones sin repetición.

La tabla resume todas las posibilidades:

		AGRUPACIONES	Sin repetición	Con repetición
¿Importa el orden de colocación?	**SI**	**Variaciones**. (Tomamos algunos elementos: $n < N$)	$V_N^n = \frac{N!}{(N-n)!} =$ $= N \times (N-1) \times \ldots \times (N-n+1)$	$VR_N^n = N^n$
		Permutaciones. (Tomamos todos los elementos: $n = N$)	$V_N^N = P_n = n!$	$P_n = \frac{n!}{a!\,b!\,c!\,\ldots}$ $a + b + c + \cdots = n$
	NO	**Combinaciones**.	$C_N^n = \frac{Variaciones}{Permutaciones} = \frac{V_N^n}{P_n} = \frac{N!}{n!\,(N-n)!} \equiv \binom{N}{n}$	

, donde $a, b, c, \ldots$ son el número de veces que se repite el $1º, 2º, 3º, \ldots$ elemento.

Crea un programa que empiece pidiendo el número de elementos disponibles, N, y el número de elementos que tomamos, n. A continuación, el programa te va haciendo preguntas. ¿Importa el orden de colocación de los elementos? ¿Tomamos todos los elementos disponibles? ¿Se pueden repetir los elementos?, y en función de la respuesta, aplica la fórmula adecuada y responde con el número de agrupaciones posibles. Guarda el programa como `AmpCombinatoria.py`.

PISTA 1: Ve almacenando las respuestas (`Y` or `N`) en varias variables, por ejemplo, `orderMatters`, `allElements`, y `repAllowed`, y mapea todas las combinaciones posibles. Por ejemplo, si importa el orden de colocación, tomamos todos los elementos, y no hay repetición, tenemos permutaciones sin repetición).

PISTA 2: Utiliza llamadas a la función que escribiste en el ejercicio 4.3 para calcular todos los factoriales de las distintas fórmulas.

PISTA 3: Las permutaciones con repetición tendrás que tratarlas con cuidado: Dado el número total n de elementos (los cuales pueden repetirse), deberás preguntar cuántas veces (a) se repite el primer elemento, cuántas veces (b) se repite el segundo elemento, cuántas veces (c) se repite el segundo elemento, etc., hasta que se cumpla que $a + b + c + \cdots = n$.

EJERCICIO AMPLIACIÓN. Niveles de energía del átomo de hidrógeno.
Cuando se excitan mediante descargas eléctricas, los átomos emiten luz únicamente en unas longitudes de onda (esto es, en ciertos colores) que son características del elemento implicado. La técnica de la espectroscopia permitió medir de forma precisa esas longitudes de onda. Sin embargo, la física teórica se mostró incapaz de explicar la razón por la que esto ocurría, hasta que Niels Bohr propuso su **teoría del átomo de hidrógeno**: Según Bohr, los electrones del átomo no pueden poseer cantidades arbitrarias de energía, sino que únicamente pueden existir únicamente en ciertos **estados estacionarios** cuya energía viene dada por:

$$E_n = -\frac{m_e e^4}{8\varepsilon_0^2 h^2}\,\frac{1}{n^2}$$

, donde $m_e = 9{,}1094 \times 10^{-31}\, kg$ es la masa del electrón, $e = 1{,}6022 \times 10^{-19}\, C$ es la carga elemental, $\varepsilon_0 = 8{,}8542 \times 10^{-12}\ C^2 s^2/kg\, m^3$ es la permitividad eléctrica del vacío, $h = 6{,}6261 \times 10^{-34}\, J\, s$ es la constante de Planck, y n identifica el nivel de energía en el que se encuentra el electrón ($n = 1,\ 2,\ 3,\ 4$ …).

Mientras el electrón se encuentra en un estado estacionario E_n, el átomo no radia luz. Sin embargo, un electrón puede saltar desde un nivel E_n a un nivel E_m de menor energía, lo que implica la emisión de luz con una energía igual a la diferencia de energías entre los niveles:

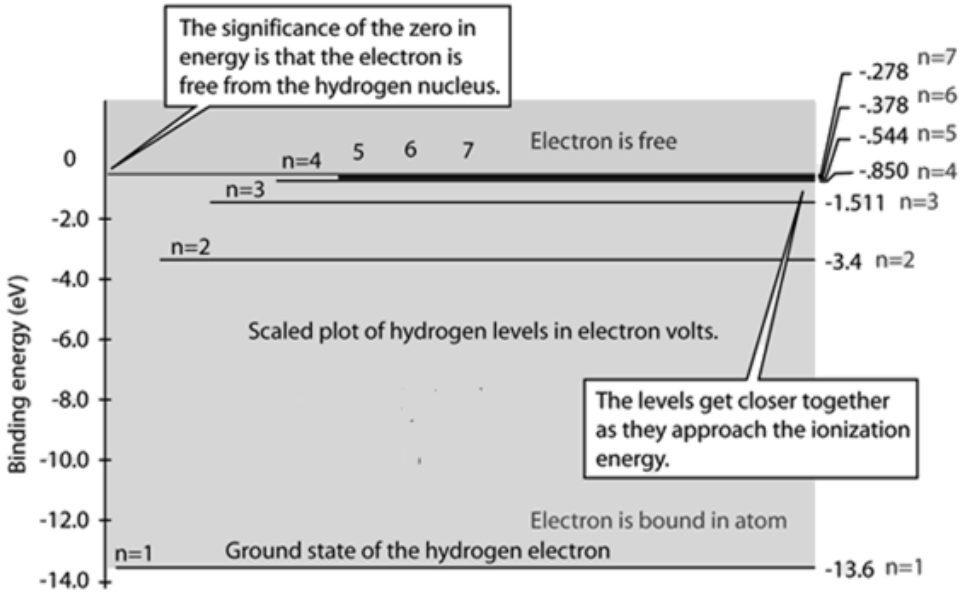
$$\Delta E = E_n - E_m = -\frac{m_e e^4}{8\varepsilon_0^2 h^2}\left(\frac{1}{n^2} - \frac{1}{m^2}\right) = \frac{m_e e^4}{8\varepsilon_0^2 h^2}\left(\frac{1}{m^2} - \frac{1}{n^2}\right)$$

Previamente a los trabajos de Bohr, Einstein había postulado que la energía transmitida por la luz consistía en "paquetes" discretos, llamados **fotones**. Según Einstein, cada fotón transporta una energía $E = hf$, donde f es la frecuencia de la luz. Empleando la relación $c = \lambda f$ (donde c es la rapidez de la luz, y λ la longitud de onda), tenemos que la longitud de onda de la luz emitida por el átomo de hidrógeno cuando un electrón pasa del nivel n al nivel m puede obtenerse como:

$$E = E_n - E_m = hf = \frac{hc}{\lambda}$$

, esto es:

$$\frac{1}{\lambda} = \frac{(E_n - E_m)}{hc} = \frac{m_e e^4}{8\varepsilon_0^2 h^3 c}\left(\frac{1}{m^2} - \frac{1}{n^2}\right)$$



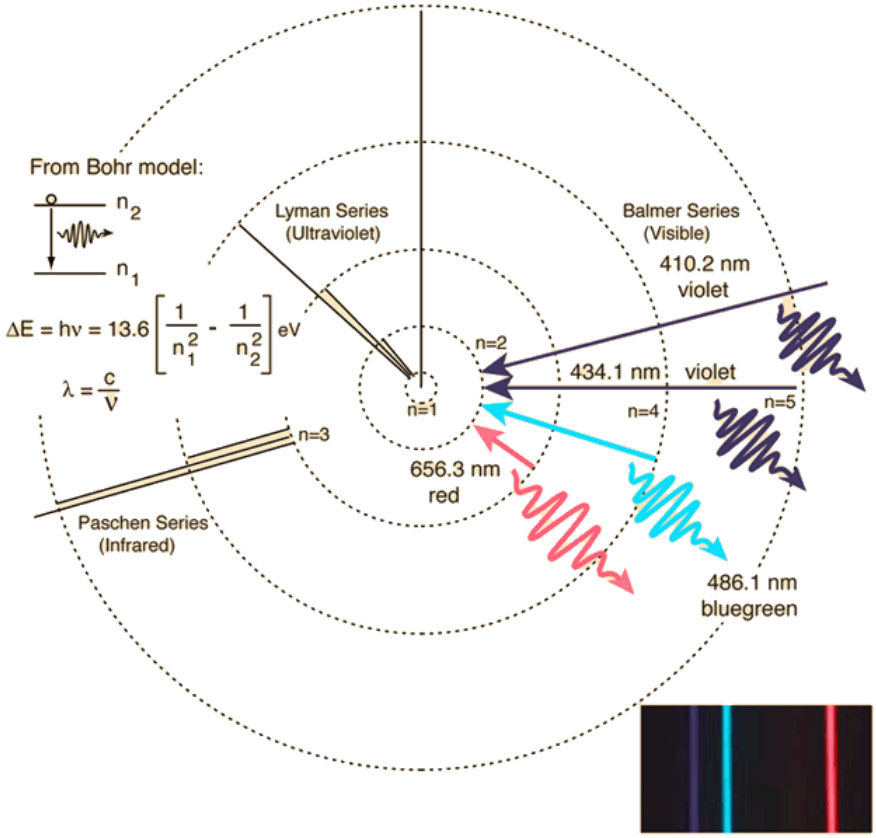
(a) Escribe una función que calcule el nivel de energía E_n de un átomo de hidrógeno, recibiendo como argumento el valor de n (donde $n = 1,\ 2,\ 3,\ 4$ …). La fórmula te dará la energía en julios (J). Convierte a unidades de electrón – voltios (eV) aplicando la conversión $1\, eV = 1{,}6022 \times 10^{-19}\, J$, y busca en internet "niveles de energía del átomo de hidrógeno" para comprobar tus resultados. CUIDADO: en este ejercicio es muy importante poner paréntesis para que las fórmulas que programes funcionen correctamente. Guarda el programa como `AmpAtomoHa.py`.

(b) Escribe un programa que calcule las longitudes de onda de la luz emitida por el átomo de hidrógeno (el llamado **espectro de emisión del hidrógeno**), basándote en la fórmula:

$$\frac{1}{\lambda} = \frac{E_n}{hc} - \frac{E_m}{hc}$$

Pista 1: necesitarás dos bucles anidados: El bucle externo recorrerá los niveles $m = 1, 2, 3$, y para cada uno de estos valores, un bucle interno calculará la longitud de onda de la luz emitida debido a la transición desde los niveles $n = m + k$ (con $k = 1,\ 2,\ 3,\ 4,\ 5$) a ese nivel m.
Pista 2: Utiliza la función que te permitía hallar la energía E_n de un nivel cualquiera.

Esta fórmula te da la longitud de onda en metros. Convierte de metros (m) a angstroms (Å) aplicando la conversión $1\ \text{Å} = 10^{-10} m$, y muestra las longitudes de onda tanto en metros como en angstroms. Busca en internet "espectro de emisión del hidrógeno" para comprobar tus resultados. Guarda el programa como `AmpAtomoHb.py`.



4. LISTAS.

4.1. EL TIPO DE DATOS LISTA.

¿QUÉ ES UNA LISTA?

Una herramienta muy útil a la hora de escribir programas son los tipos de datos **lista** y **tupla**. Las listas y las tuplas son valores pueden contener múltiples valores, lo que nos facilita enormemente escribir programas que tengan que manejar grandes cantidades de datos. Aquí comenzaremos estudiando las listas.

Una **lista** es un valor que contiene múltiples valores en una secuencia ordenada. El término *valor tipo lista* se refiere a la propia lista, y no a los valores que hay dentro de ella. Un valor tipo lista puede parecerse a lo siguiente: `['gato','perro','loro','elefante']`. De la misma forma que los valores tipo cadena se escriben entre comillas simples para indicar dónde empieza y termina la cadena, las listas empiezan abriendo con un corchete y terminan cerrando con otro corchete, `[]`. Los valores dentro de la lista se denominan **objetos** (items). Los objetos están separados por comas.

A modo de ejemplo, escribe lo siguiente en el shell interactivo:

```
>>> [1, 2, 3]
[1, 2, 3]
>>> ['gato','perro','loro','elefante']
['gato', 'perro', 'loro', 'elefante']
>>> ['hola', 3.1415, True, None, 'elefante']
['hola', 3.1415, True, None, 'elefante']
>>> miLista = ['gato','perro','loro','elefante']
>>> miLista
['gato', 'perro', 'loro', 'elefante']
```

Notar que a la variable `miLista` solo se le asigna un único valor: El valor tipo lista. Pero la propia lista contiene múltiples valores. El valor `[]` es una lista vacía que no contiene valores, como la cadena vacía `''`.

ACCEDER A LOS VALORES INDIVIDUALES DE UNA LISTA CON ÍNDICES.

Suponer que tenemos la lista `['gato','perro','loro','elefante']` almacenada en la variable `miLista`. El código Python `miLista[0]` se evaluaría a `'gato'`, el código `miLista[1]` se evaluaría a `'perro'`, y así sucesivamente. El entero dentro de los corchetes que viene tras el nombre de la lista se denomina **índice**. El primer valor de la lista está en el índice `0`, el segundo valor está en el índice `1`, el tercer valor en el índice `2`, etc. La figura muestra un valor tipo lista asignado a la variable `miLista`, junto con la forma en la que se evaluarían las expresiones con índices.

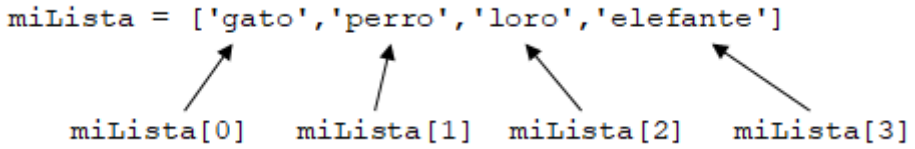


Figura 4.1. Índices de una lista.

A modo de ejemplo, escribe las siguientes expresiones en el shell interactivo. Comienza asignando una lista a la variable `miLista`:

```
>>> miLista = ['gato','perro','loro','elefante']
>>> miLista[0]
'gato'
```

```
>>> miLista[1]
'perro'
>>> miLista[2]
'loro'
>>> miLista[3]
'elefante'
>>> ['gato','perro','loro','elefante'][2]
'loro'
>>> 'El ' + miLista[0] + ' se comió al ' + miLista[2] + '.'
'El gato se comió al loro.'
```

Python nos dará un error de tipo `IndexError` si usamos un índice que exceda el número de valores en nuestra lista:

```
>>> miLista = ['gato','perro','loro','elefante']
>>> miLista[20]
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    miLista[20]
IndexError: list index out of range
```

Los índices solo pueden ser valores enteros, no flotantes. El siguiente ejemplo causa un error de tipo `TypeError`:

```
>>> miLista = ['gato','perro','loro','elefante']
>>> miLista[1]
'perro'
>>> miLista[1.0]
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    miLista[1.0]
TypeError: list indices must be integers or slices, not float
```

Las listas también pueden contener otros valores de tipo lista. Podemos acceder a los valores dentro de estas listas usando índices múltiples, como por ejemplo:

```
>>> miLista = [['gato', 'perro'], [10, 20, 30, 40, 50]]
>>> miLista[0]
['gato', 'perro']
>>> miLista[0][1]
'perro'
>>> miLista[1][4]
50
```

El primer índice indica qué valor tipo lista usar, y el segundo indica qué valor tomar dentro de ese valor tipo lista. Por ejemplo, `miLista[0][1]` imprime `'perro'`, esto es, el segundo valor dentro de la primera lista. Si solo usamos un índice, el programa imprimirá toda la lista situada en ese índice.

ÍNDICES NEGATIVOS.

Aunque los índices empiezan con el 0 y van aumentando, también podemos usar enteros negativos. El valor -1 se refiere al último índice de la lista, el valor -2 al penúltimo índice de la lista, y así sucesivamente. Escribe lo siguiente en el shell interactivo:

```
>>> miLista = ['gato','perro','loro','elefante']
>>> miLista[-1]
'elefante'
```

```
>>> miLista[-2]
'loro'
>>> 'Al ' + miLista[-1] + ' le asusta el ' + miLista[-3] + '.'
'Al elefante le asusta el perro.'
```

OBTENER SUBLISTAS MEDIANTE CORTES.

De la misma forma que un índice nos permite obtener un valor de una lista, un **corte** (slice) nos permite obtener varios valores de una lista. Los cortes se escriben entre corchetes, igual que los índices, pero tienen dos enteros separados por dos puntos. Observa la diferencia entre los índices y los cortes:

- `miLista[2]` es una lista con un índice (un entero).
- `miLista[1:4]` es una lista con un corte (dos enteros separados por dos puntos).

En un corte, el primer entero es el índice donde empieza el corte, y el segundo entero es el índice donde termina el corte. Un corte llegará hasta, pero no incluirá, el valor del segundo índice. Un corte se evalúa a un nuevo valor tipo lista. Escribe lo siguiente en el shell interactivo:

```
>>> miLista = ['gato','perro','loro','elefante']
>>> miLista[1:3]
['perro', 'loro']
>>> miLista[0:-1]
['gato', 'perro', 'loro']
```

A modo de atajo, podemos dejar de poner uno o los dos índices de un corte. Si no ponemos el primer entero, es lo mismo que poner un `0`, esto es, el principio de la lista. Si no ponemos el segundo índice es lo mismo que usar la longitud de la lista, lo que hará que el corte llegue hasta el final de la lista. Escribe lo siguiente en el shell interactivo:

```
>>> miLista = ['gato','perro','loro','elefante']
>>> miLista[:2]
['gato', 'perro']
>>> miLista[1:]
['perro', 'loro', 'elefante']
>>> miLista[:]
['gato', 'perro', 'loro', 'elefante']
```

OBTENER LA LONGITUD DE UNA LISTA CON LEN().

La función `len()` devolverá el número de valores que hay en el valor tipo lista que le pasemos como argumento, de la misma forma que cuenta el número de caracteres de un valor tipo cadena. Inserta lo siguiente en el shell interactivo:

```
>>> miLista = ['gato', 'perro', 'pato', [10, 20, 30, 40, 50]]
>>> len(miLista)
4
```

MODIFICAR LOS VALORES DE UNA LISTA CON ÍNDICES.

Normalmente, el nombre de una variable va a la izquierda de una sentencia de asignación, como por ejemplo, `edad = 42`. Sin embargo, también podemos usar un índice de una lista para cambiar el valor en ese índice. Por ejemplo, `miLista[1] = 'caballo'` significa "cambia el valor que haya en el índice 1 de la lista `miLista` por la cadena `'caballo'`". El valor que hubiese previamente en ese índice desaparece, sustituido por el nuevo valor proporcionado.

A modo de ejemplo, escribe lo siguiente en el shell interactivo:

```
>>> miLista = ['gato','perro','loro','elefante']
>>> miLista[1] = 'caballo'
>>> miLista
['gato', 'caballo', 'loro', 'elefante']
>>> miLista[2] = miLista[1]
>>> miLista
['gato', 'caballo', 'caballo', 'elefante']
>>> miLista[-1] = 12345
>>> miLista
['gato', 'caballo', 'caballo', 12345]
>>> miLista[4] = 'gallina'
Traceback (most recent call last):
  File "<pyshell#46>", line 1, in <module>
    miLista[4] = 'gallina'
IndexError: list assignment index out of range
```

Notar que si indicamos un índice fuera de rango Python da un error, ya que no puede reemplazar un valor en un índice que no existe.

CONCATENACIÓN DE LISTAS.

El operador + puede combinar dos listas para crear un nuevo valor de tipo lista, de la misma forma que combina dos cadenas para crear un nuevo valor de tipo cadena. Escribe lo siguiente en el shell interactivo:

```
>>> [1, 2, 3] + ['A', 'B', 'C']
[1, 2, 3, 'A', 'B', 'C']
>>> miLista = [1, 2, 3]
>>> miLista = miLista + ['A', 'B', 'C']
>>> miLista
[1, 2, 3, 'A', 'B', 'C']
```

QUITAR VALORES DE UNA LISTA CON LA SENTENCIA DEL.

La sentencia `del` borra de una lista el valor localizado en el índice que se le pasa como argumento. Todos los valores de la lista situados detrás del valor borrado se mueven un índice hacia abajo. Por ejemplo, inserta lo siguiente en el shell interactivo:

```
>>> miLista = ['gato','perro','loro','elefante']
>>> del miLista[2]
>>> miLista
['gato', 'perro', 'elefante']
>>> del miLista[2]
>>> miLista
['gato', 'perro']
```

4.2. TRABAJAR CON LISTAS.

Para los principiantes resulta tentador crear muchas variables individuales para almacenar un grupo de valores similares. Por ejemplo, si quisiéramos almacenar los nombres de los alumnos de una clase, podría ser tentador escribir un código como éste:

```
alumno1 = 'Alejandro'
alumno2 = 'Susana'
alumno3 = 'Rubén'
alumno4 = 'Alba'
```

```
alumno5 = 'Irene'
alumno6 = 'Manuel'
```

Pero esta forma de escribir código no es la más adecuada. Por ejemplo, si el número de alumnos cambia, nuestro programa no será capaz de almacenar más alumnos que variables tiene el programa. Además, este tipo de programas también suelen tener un montón de código duplicado o prácticamente idéntico. Por ejemplo, consideremos la cantidad de código duplicado que hay en el siguiente programa. Escríbelo en el editor de archivos y guárdalo con el nombre `alumnos1.py`.

```
print('Escribe el nombre del alumno 1: ')
alumno1 = input()
print('Escribe el nombre del alumno 2: ')
alumno2 = input()
print('Escribe el nombre del alumno 3: ')
alumno3 = input()
print('Escribe el nombre del alumno 4: ')
alumno4 = input()
print('Escribe el nombre del alumno 5: ')
alumno5 = input()
print('Escribe el nombre del alumno 6: ')
alumno6 = input()
print(alumno1 + ' ' + alumno2 + ' ' + alumno3 + ' '
      + alumno4 + ' ' + alumno5 + ' ' + alumno6 + '.')
```

En vez de usar múltiples variables repetitivas, podemos usar una sola variable que contenga un valor tipo lista. Por ejemplo, he aquí una nueva versión mejorada del programa `alumnos1.py`. Esta nueva versión usa una lista y puede almacenar tantos alumnos como el usuario quiera introducir. Escribe el siguiente código en el editor de archivos y guárdalo como `alumnos2.py`:

```
alumnos = []
while True:
    print('Escribe el nombre del alumno ' + str(len(alumnos) + 1) +
          ' (o pulsa ENTER para salir):')
    nombre = input()
    if nombre == '':
        break
    alumnos = alumnos + [nombre]    # Concatenación de dos listas.
print('Los nombres de los alumnos son:')
for name in alumnos:
    print(name)
```

Cuando ejecutemos este programa, la salida se parecerá a lo siguiente:

```
Escribe el nombre del alumno 1 (o pulsa ENTER para salir):
Alejandro
Escribe el nombre del alumno 2 (o pulsa ENTER para salir):
Marta
Escribe el nombre del alumno 3 (o pulsa ENTER para salir):
Rubén
Escribe el nombre del alumno 4 (o pulsa ENTER para salir):
Carlos
Escribe el nombre del alumno 5 (o pulsa ENTER para salir):

Los nombres de los alumnos son:
Alejandro
Marta
Rubén
Carlos
```

La ventaja de usar una lista es que ahora nuestros datos están estructurados, lo que permite que nuestro programa pueda procesarlos con mucha mayor flexibilidad que si los mismos datos estuviesen guardados en múltiples variables repetitivas. Por cierto, tal vez no hayamos entendido del todo la forma en la que se ha usado el bucle `for` en este ejemplo. A continuación, explicaremos cómo usar bucles `for` para recorrer los distintos elementos de una lista.

USAR BUCLES FOR CON LISTAS.

En la sección 3.2 aprendimos a usar bucles `for` para ejecutar un bloque de código un cierto número conocido de veces. Lo que entonces no explicamos es que, en realidad, un bucle `for` repite el bloque de código una vez por cada valor que haya en una lista. Por ejemplo, si ejecutamos este código:

```
for i in range(4):
    print(i)
```

, la salida del programa es:

```
0
1
2
3
```

Esto es así porque el valor de retorno de la función `range(4)` es un valor similar a la lista `[0, 1, 2, 3]`. De hecho, el siguiente programa produce la misma salida que el anterior:

```
numLista = [0, 1, 2, 3]
for num in numLista:
    print(num)
```

El bucle `for` de este último programa repite su bloque de código con la variable `num` tomando los distintos valores de la lista `numLista` sucesivamente en cada repetición.

Una técnica muy común es usar `range(len(miLista))` con un bucle `for` para iterar sobre los índices de una lista. Por ejemplo, escribe el siguiente código en el shell interactivo:

```
>>> materiales = ['lápices', 'bolígrafos', 'tijeras', 'rotuladores']
>>> for i in range(len(materiales)):
     print('El índice ' + str(i) + ' de la lista de materiales es: '
           + materiales[i])

El índice 0 de la lista de materiales es: lápices
El índice 1 de la lista de materiales es: bolígrafos
El índice 2 de la lista de materiales es: tijeras
El índice 3 de la lista de materiales es: rotuladores
```

Haber usado `range(len(materiales))` en el bucle `for` previo ha sido muy útil, porque de esta forma el código dentro del bucle puede acceder al índice (con la variable `i`) y al valor en ese índice (con `materiales[i]`). Y lo mejor de todo, `range(len(matriales))` iterará a través de *todos* los índices de `materiales`, independientemente de cuántos objetos contenga esta lista.

LOS OPERADORES IN Y NOT IN.

Podemos saber si un cierto valor está o no está dentro de una lista con los operadores `in` y `not in`. Como cualquier otro operador, los operadores `in` y `not in` se usan en expresiones y conectan dos valores: El

valor a buscar en la lista, y la lista donde buscarlo. Estas expresiones se evaluarán a un valor Booleano. Escribe el siguiente código en el shell interactivo:

```
>>> alumnos = ['Alex', 'Susana', 'Ana', 'Carlos', 'Paula']
>>> 'Ana' in ['Alex', 'Susana', 'Ana', 'Carlos', 'Paula']
True
>>> 'Luis' in alumnos
False
>>> 'Alex' not in alumnos
False
>>> 'Irene' not in alumnos
True
```

Por ejemplo, el siguiente programa le permite al usuario escribir un nombre de alumno y comprobar si ese nombre está en la lista de alumnos. Abre un nuevo editor de archivos, escribe el siguiente código, y guárdalo como `misAlumnos.py`:

```
misAlumnos = ['Alex', 'Susana', 'Ana']
print('Escribe el nombre del alumno:')
nombre = input()
if nombre not in misAlumnos:
    print('En la lista no hay ningún alumno llamado ' + nombre + '.')
else:
    print(nombre + ' está en la lista de alumnos.')
```

La salida del programa debería parecerse a lo siguiente:

```
Escribe el nombre del alumno:
Cayetano
En la lista no hay ningún alumno llamado Cayetano.
```

EL TRUCO DE LAS ASIGNACIONES MÚLTIPLES.

El **truco de las asignaciones múltiples** es un atajo que nos permite asignar a múltiples variables los valores almacenados en una lista usando una sola línea de código. Por lo tanto, en vez de hacer esto:

```
>>> alumno = ['Alejandro', 'Martínez', '4B', 2004]
>>> nombre = alumno[0]
>>> apellido = alumno[1]
>>> curso = alumno[2]
>>> nacimiento = alumno[3]
```

, podríamos escribirlo todo en una sola línea de código, como:

```
>>> alumno = ['Alejandro', 'Martínez', '4B', 2004]
>>> nombre, apellido, curso, nacimiento = alumno
```

El número de variables y la longitud de la lista debe ser exactamente el mismo. En caso contrario, Python dará un error de tipo `ValueError`.

4.3. MÉTODOS DE LISTAS.

Un **método** es lo mismo que una función, excepto que a los métodos se les llama "sobre un valor" de un tipo determinado. Por ejemplo, suponer que la variable `miLista` almacena un valor tipo lista; en ese caso, podemos llamar al método `index()` (del que hablaremos más adelante) sobre esa lista de la siguiente forma: `miLista.index('gato')`. Notar que el nombre del método viene después del nombre de la

variable que almacena el valor, separado por un punto. Cada tipo de datos tiene su propio conjunto de métodos. En particular, el tipo de datos lista tiene varios métodos útiles para encontrar, añadir, quitar, y manipular los valores de una lista.

ENCONTRAR UN VALOR EN UNA LISTA CON EL MÉTODO INDEX().

Los valores tipo lista tienen un método `index()` al que se le puede pasar un valor, y si ese valor existe en la lista, devuelve el índice de ese valor. Si el valor no está en la lista, Python produce un error `ValueError`. Escribe lo siguiente en el shell interactivo:

```
>>> miLista = ['gato','perro','loro','elefante']
>>> miLista.index('gato')
0
>>> miLista.index('elefante')
3
>>> miLista.index('ornitorrinco')
Traceback (most recent call last):
  File "<pyshell#35>", line 1, in <module>
    miLista.index('ornitorrinco')
ValueError: 'ornitorrinco' is not in list
```

Cuando hay valores duplicados en la lista, `index()` devuelve el índice de la primera aparición. Escribe lo siguiente en el shell interactivo, y observa que `index()` devuelve `1`, y no `3`.

```
>>> miLista = ['gato','perro','loro','perro']
>>> miLista.index('perro')
1
```

AÑADIR VALORES A UNA LISTA CON LOS MÉTODOS APPEND() E INSERT().

Para añadir nuevos valores a una lista, usamos los métodos `append()` e `insert()`. Escribe lo siguiente en el shell interactivo para llamar al método `append()` sobre un valor tipo lista que está almacenado en la variable `miLista`:

```
>>> miLista = ['gato','perro','loro']
>>> miLista.append('pato')
>>> miLista
['gato', 'perro', 'loro', 'pato']
```

La llamada al método `append()` añade el argumento al final de la lista. El método `insert()` nos permite insertar un valor en un índice cualquier de la lista. El primer argumento de `insert()` es el índice del nuevo valor, y el segundo argumento es el nuevo valor a insertar. Escribe lo siguiente en el shell interactivo:

```
>>> miLista = ['gato','perro','loro']
>>> miLista.insert(1, 'gallina')
>>> miLista
['gato', 'gallina', 'perro', 'loro']
```

Notar que escribimos `miLista.append('pato')` y `miLista.insert(1, 'gallina')`, y no `miLista = miLista.append('pato')` ni `miLista = miLista.insert(1, 'gallina')`. Ni `append()` ni `insert()` le dan un nuevo valor a `miLista` como su valor de retorno. (De hecho, el valor de retorno de `append()` e `insert()` es `None`, por lo que es evidente que no querremos guardar este valor como el nuevo valor de la variable). Hablaremos más de todas estas cosas en la sección "Tipos de datos mutables e inmutables".

Los métodos pertenecen a un solo tipo de datos. Por ejemplo, los métodos `append()` e `insert()` son métodos de listas, y solo pueden llamarse sobre valores tipo lista, y no sobre otros valores como enteros o cadenas. Si, por ejemplo, intentamos llamar al método `append()` sobre una cadena (o un entero, o cualquier otro valor que no sea una lista), Python devolverá un error de tipo `AttributeError`.

```
>>> nombre = 'Alejandro'
>>> nombre.append('Martínez')
Traceback (most recent call last):
  File "<pyshell#49>", line 1, in <module>
    nombre.append('Martínez')
AttributeError: 'str' object has no attribute 'append'
```

QUITAR VALORES DE UNA LISTA CON REMOVE().

Al método `remove()` se le pasa el valor a quitar de la lista sobre la que se le llama. Escribe lo siguiente en el shell interactivo:

```
>>> miLista = ['gato','perro','loro','elefante']
>>> miLista.remove('loro')
>>> miLista
['gato', 'perro', 'elefante']
```

Si intentamos borrar un valor que no existe en la lista, Python devolverá un error `ValueError`. Si el valor aparece varias veces en la lista, solo se borrará la primera instancia.

No confundir la sentencia `del` con el método `remove`: La sentencia `del` es útil cuando sabemos el índice del valor que queremos eliminar de la lista. Por su parte, el método `remove()` es útil cuando conocemos el valor que queremos quitar de la lista.

ORDENAR LOS VALORES DE UNA LISTA CON EL MÉTODO SORT().

Las listas de valores numéricos o de valores tipo cadena pueden clasificarse con el método `sort()`. Por ejemplo, escribe lo siguiente en el shell interactivo:

```
>>> numList = [2, 5, 3.14, 1, -7]
>>> numList.sort()
>>> numList
[-7, 1, 2, 3.14, 5]
>>> cadList = ['gato','perro','zorro','asno', 'caballo']
>>> cadList.sort()
>>> cadList
['asno', 'caballo', 'gato', 'perro', 'zorro']
```

También podemos escribir `sort(reverse=True)` para hacer que el método clasifique los valores en orden inverso:

```
>>> cadList.sort(reverse=True)
>>> cadList
['zorro', 'perro', 'gato', 'caballo', 'asno']
```

Hay dos cosas que debemos de tener en cuenta acerca del método `sort()`:

(1) No podemos ordenar listas que contengan tanto números como cadenas, ya que Python no sabe cómo comparar valores de distinto tipo. En caso de hacerlo, Python lanzará un error de tipo `TypeError`.

(2) El método `sort()` utiliza el "orden ASCIIbético" para ordenar cadenas, en vez de usar el orden alfabético real. Esto significa que las letras mayúsculas vienen antes que las letras minúsculas. Por consiguiente, la letra minúscula `a` se clasificará detrás que la letra mayúscula `Z`. He aquí un ejemplo:

```
>>> cadList = ['Alejandro','perro','zorro','Cristina','Roberto', 'asno']
>>> cadList.sort()
>>> cadList
['Alejandro', 'Cristina', 'Roberto', 'asno', 'perro', 'zorro']
```

Si necesitamos clasificar los valores en orden alfabético normal, debemos escribir `sort(key=str.lower)`. Esto hace que la función `sort()` trate todos los objetos de la lista como si estuviesen escritos en minúscula, pero sin cambiar los valores de la lista.

```
>>> cadList = ['Alejandro','perro','zorro','Cristina','Roberto', 'asno']
>>> cadList.sort(key=str.lower)
>>> cadList
['Alejandro', 'asno', 'Cristina', 'perro', 'Roberto', 'zorro']
```

4.4. TIPOS SIMILARES A LAS LISTAS: CADENAS Y TUPLAS.

CADENAS.

Las listas no son los únicos tipos que nos permiten representar secuencias ordenadas de valores. Por ejemplo, las cadenas y las listas son muy similares, si consideramos que una cadena es una "lista" de caracteres individuales. Muchas de las cosas que podemos hacer con las listas también las podemos hacer con las cadenas: indexado, troceado, uso de bucles `for`, de la función `len()`, y de los operadores `in` y `not in`. Para entender todo esto, escribe lo siguiente en el shell interactivo:

```
>>> nombre = 'Paula'
>>> nombre[0]
'P'
>>> nombre[-2]
'l'
>>> nombre[0:3]
'Pau'
>>> 'Pa' in nombre
True
>>> 'z' in nombre
False
>>> 'a' not in nombre
False
>>> for i in nombre:
     print('* * * ' + i + ' * * *')

* * * P * * *
* * * a * * *
* * * u * * *
* * * l * * *
* * * a * * *
```

TIPOS DE DATOS MUTABLES E INMUTABLES.

A pesar de sus similitudes, las listas y las cadenas son intrínsecamente distintas. Un valor tipo lista es un tipo de datos **mutable**, lo que significa que se le pueden añadir, quitar, o cambiar sus valores. Por el contrario, una cadena es **inmutable**, esto es, no puede cambiarse.

Si intentamos cambiar un carácter individual dentro de una cadena, obtendremos un error `TypeError`:

```
>>> nombre = 'Alejandro'
>>> nombre[-1] = 'a'
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    nombre[-1] = 'a'
TypeError: 'str' object does not support item assignment
```

La forma adecuada de "modificar" una cadena es usar el troceado y la concatenación para construir una *nueva* cadena copiando partes de la antigua. Escribe lo siguiente en el shell interactivo:

```
>>> nombre = 'Alejandro'
>>> nuevoNombre = nombre[0:8] + 'ete'
>>> nombre
'Alejandro'
>>> nuevoNombre
'Alejandrete'
```

Notar que la cadena original (`'Alejandro'`) no se ha visto modificada, porque las cadenas son inmutables. Puede que hablar de tipos de datos mutables e inmutables nos parezca una distinción irrelevante, pero al pasarle argumentos a una función, su comportamiento puede ser bien distinto dependiendo de si los argumentos son mutables o inmutables.

A continuación, vamos a presentar el tipo de datos tupla, que es la versión inmutable del tipo de datos lista.

TUPLAS.

El tipo de datos **tupla** es casi idéntico al tipo de datos lista, excepto en dos aspectos: Primero, las tuplas se escriben con paréntesis, `(` y `)`, en lugar de hacerlo con corchetes. Por ejemplo, escribe lo siguiente en el shell interactivo:

```
>>> miTupla = ('hola', 42, 0.25)
>>> miTupla[0]
'hola'
>>> miTupla[1:3]
(42, 0.25)
>>> len(miTupla)
3
```

Pero el aspecto fundamental en el que se diferencian las tuplas de las listas es que las tuplas, al igual que las cadenas, son inmutables: Los valores de las tuplas no se pueden modificar, anexar, ni quitar. Escribe lo siguiente en el shell interactivo, y observa cómo aparece el error `TypeError`:

```
>>> miTupla = ('hola', 42, 0.25)
>>> miTupla[0] = 'adiós'
Traceback (most recent call last):
  File "<pyshell#41>", line 1, in <module>
    miTupla[0] = 'adiós'
TypeError: 'tuple' object does not support item assignment
```

¿Para qué sirven las tuplas? Si queremos que una secuencia de valores no cambie nunca, debemos usar una tupla. Así, podemos usar las tuplas para indicarle a alguien que esté leyendo nuestro código que no pretendemos cambiar una cierta secuencia de valores.

CONVERTIR TIPOS CON LAS FUNCIONES LIST() Y TUPLE().

De la misma forma que `str(42)` convierte el entero `42` en la cadena `'42'`, las funciones `list()` y `tuple()` devuelven las versiones tipo lista y tipo tupla de los valores que les pasemos. Escribe lo siguiente en el shell interactivo, y observa que el valor de retorno es de un tipo de datos distinto al valor pasado:

```
>>> tuple(['perro', 'gato', 5, 1.75])
('perro', 'gato', 5, 1.75)
>>> list(('Alejandro', 'Marta', 42))
['Alejandro', 'Marta', 42]
>>> list('hola')
['h', 'o', 'l', 'a']
```

Convertir una tupla en una lista puede ser útil cuando necesitemos una versión mutable de ese valor tipo tupla.

4.5. PROGRAMA DE EJEMPLO: BINGO.

Para poner en práctica los conceptos que hemos estudiado sobre listas, vamos a escribir un programa que implemente un bingo para dos jugadores. El programa comienza permitiendo que los jugadores puedan elegir el número máximo de bolas que habrá en el bombo (40 por defecto), y crea una lista de números que representa las bolas contenidas en el bombo. A continuación, el programa construye los cartones para los dos jugadores, donde cada cartón contiene seis números aleatorios de entre los presentes en el bombo. Conforme extrae las bolas del bombo, el programa comprueba si el número extraído está en alguno de los dos cartones, y simultáneamente, si alguno de los dos jugadores ha "tachado" todos los números de su cartón y canta bingo.

Copia este código en el editor de archivos, y guarda el programa como `bingo.py`:

```
# Bingo para dos jugadores.

import random
import sys

# La función rellenaLista() permite crear las listas para construir
# los números del bombo y para confeccionar los cartones de los jugadores.
def rellenaLista (numMax):
    listaNumeros = []
    for i in range(numMax):
        listaNumeros.append(i+1)
    return listaNumeros

# La función creaCarton() sirve para crear los cartones de los jugadores.
def creaCarton (listaNumeros):
    tarjeta = []
    for i in range (6):
        numMax = len (listaNumeros)
        numero = listaNumeros[random.randint(0, numMax-1)]
        tarjeta.append(numero)
        tarjeta.sort()
        listaNumeros.remove(numero)
    return tarjeta
```

```
# La función extraeBola()sirve para extraer las bolas de una lista de números.
def extraeBola (listaNumeros):
    numMax = len (listaNumeros)
    bola = listaNumeros[random.randint(0, numMax-1)]
    return bola

# PROGRAMA PRINCIPAL.

# Empezamos eligiendo el valor máximo para nuestras listas de números.
print(' Elige el número de bolas en el bombo (o pulsa INTRO para 40):')
res = input()
if res == '':
    numMax = 40
else:
    numMax = int(res)

# Comenzamos creando una lista de números para las bolas del bombo.
listaBolas = rellenaLista (numMax)

# Creamos dos nuevas lista de números para construir los cartones de los
jugadores.
print('Creando el cartón con el que juega el jugador 1...')
listaTarjeta1 = rellenaLista (numMax)
tarjeta1 = creaCarton (listaTarjeta1)
print('Creando el cartón con el que juega el jugador 2...')
listaTarjeta2 = rellenaLista (numMax)
tarjeta2 = creaCarton (listaTarjeta2)
print('')
print('El cartón con el que juega el jugador 1 es: ' + str(tarjeta1))
print('El cartón con el que juega el jugador 2 es: ' + str(tarjeta2))
print('')

while (True):
    print('Pulsa INTRO para sacar una bola, u otra tecla para salir.')
    tecla = input()
    if tecla != '':
        sys.exit()
    # Cada vez que el usuario pulse INTRO, sacamos una bola del bombo.
    bola = extraeBola (listaBolas)
    print('Sale el número: ' + str(bola))
    listaBolas.remove(bola)     # Quitamos la bola extraída del bombo.
    # Miramos si el número extraído está en alguno de los cartones.
    # Si lo está, eliminamos ese número del cartón correspondiente.
    if bola in tarjeta1:
        tarjeta1.remove(bola)
    if bola in tarjeta2:
        tarjeta2.remove(bola)
    # Mostramos el estado actual de los cartones.
    print('Cartón 1: ' + str(tarjeta1))
    print('Cartón 2: ' + str(tarjeta2))
    # Comprobamos si alguno de los cartones queda vacío.
    # En ese caso, el jugador correspondiente canta bingo.
    if len (tarjeta1) == 0 and not len (tarjeta2) == 0:
        print('El jugador 1 canta bingo!')
        sys.exit()
    elif len (tarjeta2) == 0 and not len (tarjeta1) == 0:
        print('El jugador 2 canta bingo!')
        sys.exit()
    elif len (tarjeta2) == 0 and len (tarjeta1) == 0:
        print('Los dos jugadores cantan bingo!')
        sys.exit()
```

Si ejecutamos el programa, la salida será parecida a la siguiente:

```
Elige el número de bolas en el bombo(o pulsa INTRO para 40):
20

Creando el cartón con el que juega el jugador 1...
Creando el cartón con el que juega el jugador 2...

El cartón con el que juega el jugador 1 es: [4, 6, 8, 15, 17, 18]
El cartón con el que juega el jugador 2 es: [2, 3, 9, 10, 16, 19]

Pulsa INTRO para sacar una bola, u otra tecla para salir.

Sale el número: 19
Cartón 1: [4, 6, 8, 15, 17, 18]
Cartón 2: [2, 3, 9, 10, 16]
Pulsa INTRO para sacar una bola, u otra tecla para salir.

Sale el número: 18
Cartón 1: [4, 6, 8, 15, 17]
Cartón 2: [2, 3, 9, 10, 16]
Pulsa INTRO para sacar una bola, u otra tecla para salir.

Sale el número: 11
Cartón 1: [4, 6, 8, 15, 17]
Cartón 2: [2, 3, 9, 10, 16]
Pulsa INTRO para sacar una bola, u otra tecla para salir.

Sale el número: 17
Cartón 1: [4, 6, 8, 15]
Cartón 2: [2, 3, 9, 10, 16]
Pulsa INTRO para sacar una bola, u otra tecla para salir.

Sale el número: 2
Cartón 1: [4, 6, 8, 15]
Cartón 2: [3, 9, 10, 16]
Pulsa INTRO para sacar una bola, u otra tecla para salir.

Sale el número: 7
Cartón 1: [4, 6, 8, 15]
Cartón 2: [3, 9, 10, 16]
Pulsa INTRO para sacar una bola, u otra tecla para salir.

Sale el número: 6
Cartón 1: [4, 8, 15]
Cartón 2: [3, 9, 10, 16]
Pulsa INTRO para sacar una bola, u otra tecla para salir.

Sale el número: 13
Cartón 1: [4, 8, 15]
Cartón 2: [3, 9, 10, 16]
Pulsa INTRO para sacar una bola, u otra tecla para salir.

Sale el número: 14
Cartón 1: [4, 8, 15]
Cartón 2: [3, 9, 10, 16]
Pulsa INTRO para sacar una bola, u otra tecla para salir.

Sale el número: 12
Cartón 1: [4, 8, 15]
Cartón 2: [3, 9, 10, 16]
Pulsa INTRO para sacar una bola, u otra tecla para salir.
```

```
Sale el número: 1
Cartón 1: [4, 8, 15]
Cartón 2: [3, 9, 10, 16]
Pulsa INTRO para sacar una bola, u otra tecla para salir.

Sale el número: 3
Cartón 1: [4, 8, 15]
Cartón 2: [9, 10, 16]
Pulsa INTRO para sacar una bola, u otra tecla para salir.

Sale el número: 5
Cartón 1: [4, 8, 15]
Cartón 2: [9, 10, 16]
Pulsa INTRO para sacar una bola, u otra tecla para salir.

Sale el número: 15
Cartón 1: [4, 8]
Cartón 2: [9, 10, 16]
Pulsa INTRO para sacar una bola, u otra tecla para salir.

Sale el número: 20
Cartón 1: [4, 8]
Cartón 2: [9, 10, 16]
Pulsa INTRO para sacar una bola, u otra tecla para salir.

Sale el número: 8
Cartón 1: [4]
Cartón 2: [9, 10, 16]
Pulsa INTRO para sacar una bola, u otra tecla para salir.

Sale el número: 9
Cartón 1: [4]
Cartón 2: [10, 16]
Pulsa INTRO para sacar una bola, u otra tecla para salir.

Sale el número: 10
Cartón 1: [4]
Cartón 2: [16]
Pulsa INTRO para sacar una bola, u otra tecla para salir.

Sale el número: 16
Cartón 1: [4]
Cartón 2: []
El jugador 2 canta bingo!
```

4.6. EJERCICIOS.

EJERCICIO 24. Amigos. Escribe una lista con los nombres de tus 4 o 5 mejores amigos. A continuación crea un programa que recorra la lista elemento tras elemento, y que imprima el mismo mensaje para cada uno de ellos. Por ejemplo:

```
Buenos días Juan. Que tengas un buen día.
Buenos días Laura. Que tengas un buen día.
Buenos días Antonio. Que tengas un buen día.
Buenos días Elena. Que tengas un buen día.
```

El programa debe funcionar igual de bien, independientemente del número de amigos en tu lista, y de si los nombres de la lista cambian. Guarda el programa como `Ejer24.py`.

EJERCICIO 25. Lista de cadenas. Escribe una **función** que reciba una lista de cadenas. La función debe devolver la longitud de la cadena más larga en la lista. Por ejemplo, si la lista es `['peras', 'manzanas', 'zanahorias', 'limones']`, la función debería devolver 10. En el programa principal, haz varias llamadas a la función (pasándole distintas listas de cadenas) para comprobar su correcto funcionamiento. Guarda el programa como `Ejer25.py`.

EJERCICIO 26. Listas numéricas.

a) Escribe una función que construya una lista con todos los números enteros desde el uno hasta el entero que le pasemos como argumento a esa función.
b) En el mismo programa, escribe otra función que recorra una lista numérica que se le pase como argumento, y que imprima por pantalla todos los números de esa lista, un número por línea. (Si el programa tarda mucho en ejecutarse, presiona CTRL + C o cierra la ventana del shell).
c) Escribe una tercera función que calcule la suma todos los números de una lista numérica, y que devuelva el resultado de esa suma.
d) Escribe una cuarta función que te permita extraer de una lista numérica los múltiplos de 13, y que devuelva una nueva lista con esos múltiplos.

Desde el programa principal, haz una llamada a la primera función para construir una lista numérica. A continuación, y usando esa lista, llama a las otras tres funciones para comprobar su correcto funcionamiento. Guarda el programa que incluye todas estas funciones, y las llamadas necesarias para probarlas, como `Ejer26.py`.

EJERCICIO 27. Pig Latin. Pig Latin es un lenguaje sin sentido. Para traducir una palabra a "pig latin" debemos mover la primera letra al final de la palabra y añadir "ay" al final. Por ejemplo, `'pera'` se convierte en `'erapay'`, y `'luna'` se convierte en `'unalay'`. Escribe un programa que reciba una cadena con la palabra que queremos traducir a "pig latin", y que devuelva una nueva cadena con esa palabra ya traducida. Recuerda que las cadenas funcionan como las listas, con la diferencia de que son valores inmutables. Guarda el programa como `Ejer27(a).py`. A continuación, escribe un programa que haga la traducción inversa, esto es, de "pig latin" al lenguaje original. Guarda el programa como `Ejer27(b).py`.

EJERCICIO 28. Cadena invertida. Escribe una función que reciba una cadena e invierta el orden de todos sus caracteres. Por ejemplo, si la función recibe la cadena `'gato'`, debe devolver la cadena `'otga'`. En el programa principal, haz varias llamadas a la función (con distintas cadenas) para comprobar su correcto funcionamiento. PISTA: Necesitas utilizar un bucle `for` para recorrer la cdena original al revés (esto es, comenzando con el último carácter, y terminando con el primero). Guarda el programa como `Ejer28.py`.

EJERCICIO 29. Toros y vacas. Escribe un programa que juegue a "Toros y vacas" con el usuario. El juego funciona así: Se genera un número aleatorio de 4 dígitos (donde los dígitos deben ser todos diferentes), y el usuario debe adivinar ese número. Por cada dígito que el usuario haya acertado *en la posición correcta*, se obtiene un "toro". Por cada dígito que el usuario haya acertado en una posición incorrecta, se obtiene una "vaca". (https://en.wikipedia.org/wiki/Bulls_and_Cows). Cada vez que el usuario proponga un número, el programa le dice cuántos "toros" y "vacas" tiene su número. Una vez que el usuario haya adivinado el número correcto el juego termina, y le dice al usuario cuántos intentos ha necesitado para acertarlo. Guarda el programa como `Ejer29.py`.

A modo de ejemplo, digamos que el ordenador ha generado el número 1038. Un ejemplo de interacción con el usuario sería el siguiente:

```
Bienvenido al juego Toros y Vacas!
He pensado un número de 4 dígitos que debes adivinar…
PISTA:
Toro -> Has acertado un dígito en su posición correcta.
Vaca -> Has acertado un dígito, pero en una posición incorrecta.
```

```
Escribe qué número crees que es:
>>> 1234
2 toros, 0 vacas.

Inténtalo otra vez. ¿Qué número crees que es?
>>> 1243
1 toros, 1 vacas.
...
```

PISTA: Para facilitar la resolución del problema, te aconsejamos que trabajes con cadenas de 4 dítitos para representar el número secreto y las apuestas del usuario.

EJERCICIO 30. Encriptado. ROTx es una técnica de encriptación que implica "rotar" cada letra de una palabra *x* posiciones. Rotar una letra significa moverla a lo largo del alfabeto, empezando de nuevo desde el principio si es necesario. Por ejemplo, si la letra es `A` y la rotamos `3` posiciones, se convierte en `D`. Si la letra es `c` y la rotamos `5` posiciones, se convierte en `h`. Escribe una función llamada `rotaPalabra()` que reciba como argumentos una cadena y un entero, y que devuelva una nueva cadena que contenga las letras de la original "rotadas" tantas posiciones como indica el entero pasado. Por ejemplo, si le pasamos la cadena `'hola'` para que la rote 4 posiciones, la función debería devolver `'lspe'`. PISTA: Para hacerlo más sencillo, solo vamos a considerar cadenas escritas con letras minúsculas. Además, te aconsejamos crear una lista con todas las letras minúsculas del alfabeto para trabajar con ella. Guarda el programa como `Ejer30.py`.

EJERCICIO AMPLIACIÓN. Hundir la flota.
En este ejercicio construirás una versión simplificada para un jugador del juego clásico de hundir la flota. El juego construirá un tablero cuadrado de tamaño configurable por el usuario, y ubicará un barco en una casilla aleatoria de este tablero. El usuario deberá adivinar la posición del barco en el tablero, indicando su fila y su columna. Para ello, contará con un número limitado de intentos. La salida del programa debería ser parecida a la siguiente:

```
HUNDIR LA FLOTA.
Vamos a crear un tablero, y a esconder un barco en una de sus casillas.
Tu misión es adivinar la posición del barco.
¿De qué tamaño quieres el tablero? Escribe un entero N para un tablero de NxN,
o pulsa INTRO para 5x5.
4

O O O O
O O O O
O O O O
O O O O

Dispones de un total de 4 oportunidades para acertar al barco.

Turno: 1
Adivina la fila (Elige entre 1 y 4):
2
Adivina la columna (Elige entre 1 y 4):
3
Has fallado el disparo! Sigue probando.

O O O O
O O X O
O O O O
O O O O
```

```
Turno: 2
Adivina la fila (Elige entre 1 y 4):
1
Adivina la columna (Elige entre 1 y 4):
1
Has fallado el disparo! Sigue probando.

X O O O
O O X O
O O O O
O O O O

Turno: 3
Adivina la fila (Elige entre 1 y 4):
4
Adivina la columna (Elige entre 1 y 4):
2
Has fallado el disparo! Sigue probando.

X O O O
O O X O
O O O O
O X O O

Turno: 4
Adivina la fila (Elige entre 1 y 4):
3
Adivina la columna (Elige entre 1 y 4):
3
Enhorabuena! Has hundido mi barco!

X O O O
O O X O
O O B O
O X O O

>>>
```

Para escribir el programa, sigue los siguientes pasos:

1) Importamos los módulos necesarios, que en este es el módulo `random`.
2) Comenzamos con el programa principal (porque una función para dibujar el tablero, que escribiremos más adelante). El programa comienza imprimiendo todos los mensajes informativos, y preguntándole al usuario el tamaño que desea para el tablero. En función de su respuesta, construimos un tablero del tamaño indicado, o de 5×5 por defecto.
3) Rellenamos todas las casillas del tablero con Os (esto es, con cadenas iguales a `'O'`). Para ello creamos una lista vacía llamada `tablero`. Como en el ejemplo, suponer que el usuario ha elegido un tablero de 4×4. Las cuatro filas del tablero serán cuatro sublistas, cada una de ellas rellena con cuatro Os. El código que me permite construir la lista de listas que es el tablero es el siguiente:

```
tablero = []
for i in range(tam):
    fila = []
    for j in range(tam):
        fila = fila + ['O']
    tablero.append(fila)
```

4) A continuación creamos una **función** llamada `imprimeTablero()` para imprimir el estado actual del tablero. Esta función recibe la lista de listas que representa al tablero, e imprime esa lista con el

formato adecuado. Por ejemplo, si el tablero 4×4 está completamente relleno con Os, la lista que lo representa sería `[['O','O','O','O'],['O','O','O','O'],['O','O','O','O'],['O','O','O','O']]`, y la función debería imprimir lo siguiente:

```
O O O O
O O O O
O O O O
O O O O
```

El código para esta función es:

```
def imprimeTablero(tablero):
    print('')
    for fila in tablero:
        filaImpresa = ''
        for i in fila:
            filaImpresa = filaImpresa + i + ' '
        print(filaImpresa)
    print('')
```

5) Volvemos al programa principal, y ahora te toca trabajar a ti. Tras haber creado el tablero inicial totalmente relleno de Os, lo imprimimos llamando a la función recién escrita.
6) Ahora ubicamos el barco en una casilla aleatoria del tablero, cambiando la `'O'` en esa posición por una `'B'` (de barco). Para ello, nos creamos dos variables llamadas `filaBarco` y `columnaBarco`. Suponer que nuestro tablero es de 4×4. En tal caso, ambas variables deberían tomar un valor aleatorio entre 0 y 3 (porque las casillas de nuestro tablero, una lista de listas, se identifican mediante índices que van del 0 al 3).
7) A continuación calculamos el número máximo de intentos que le daremos al jugador para acertar la posición del barco. Para ser justos, le daremos un número máximo de intentos que dependa del tamaño del tablero. En nuestro caso, el número máximo de intentos será el 40% del número de casillas disponibles en el tablero (valor que deberemos redondear para obtener siempre un número de intentos que sea entero).
8) A continuación, empieza la parte del programa en el que le pedimos al usuario que adivine la fila y la columna de la posición del barco en el tablero. Aquí debemos asegurarnos de hacer lo siguiente:
   - El usuario solo contará con el número de intentos calculados en el paso previo.
   - A cada intento, le pedimos al usuario que adivine la fila y la columna de la posición del barco, y guardamos ambos datos en las variables `filaJugador` y `columnaJugador`. Para hacer el juego más intuitivo, el usuario podrá elegir una fila y una columna entre 1 y 4 (si el tablero es 4×4), y no entre 0 y 3 (que es como trabaja internamente la lista del tablero)
   - Tras conocer la apuesta del usuario, debemos comprobar si ha acertado (comparando la apuesta de usuario con la ubicación del barco). En caso de acertar, le felicitamos, y mostramos el tablero con la ubicación del barco.
   - Si el usuario no cierta, le informamos de este hecho, añadimos una `'X'` en la ubicación de su última apuesta, e impirmimos el tablero.
   - También debemos comprobar si el usuario ha vuelto a apostar por una ubicación que ya había elegido antes. En tal caso, le avisamos de ese hecho.
   - Además, debemos chequear si el usuario ha elegido una fila o una columna erróneas (por ejemplo, si el tablero es 4×4, deberíamos comprobar si el jugador ha elegido una fila o columna menor que 1 y mayor que 4). En ese caso, le avisamos de ese hecho.
   - Por último, hemos de controlar si el usuario ha agotado el número máximo de intentos. Ene se caso, el jugador pierde la partida, le informamos de este hecho, e imprimimos el tablero con todas las apuestas del usuario, y la posición del barco oculto.

Guarda el programa como `AmpHundirFlota.py`.

EJERCICIO AMPLIACIÓN. Álgebra de vectores.

Suponer que tenemos dos vectores tridimensionales $\vec{a} = (a_1, a_2, a_3)$ y $\vec{b} = (b_1, b_2, b_3)$ y un número ordinario c (un escalar). El álgebra de vectores define 5 operaciones básicas con vectores:

1) Suma de vectores:

$$\vec{a} + \vec{b} = (a_1 + b_1, a_2 + b_2, a_3 + b_3)$$

2) Resta de vectores:

$$\vec{a} - \vec{b} = (a_1 - b_1, a_2 - b_2, a_3 - b_3)$$

3) Multiplicación de un vector por un escalar:

$$c\vec{a} = (ca_1, ca_2, ca_3)$$

4) Producto escalar de dos vectores:

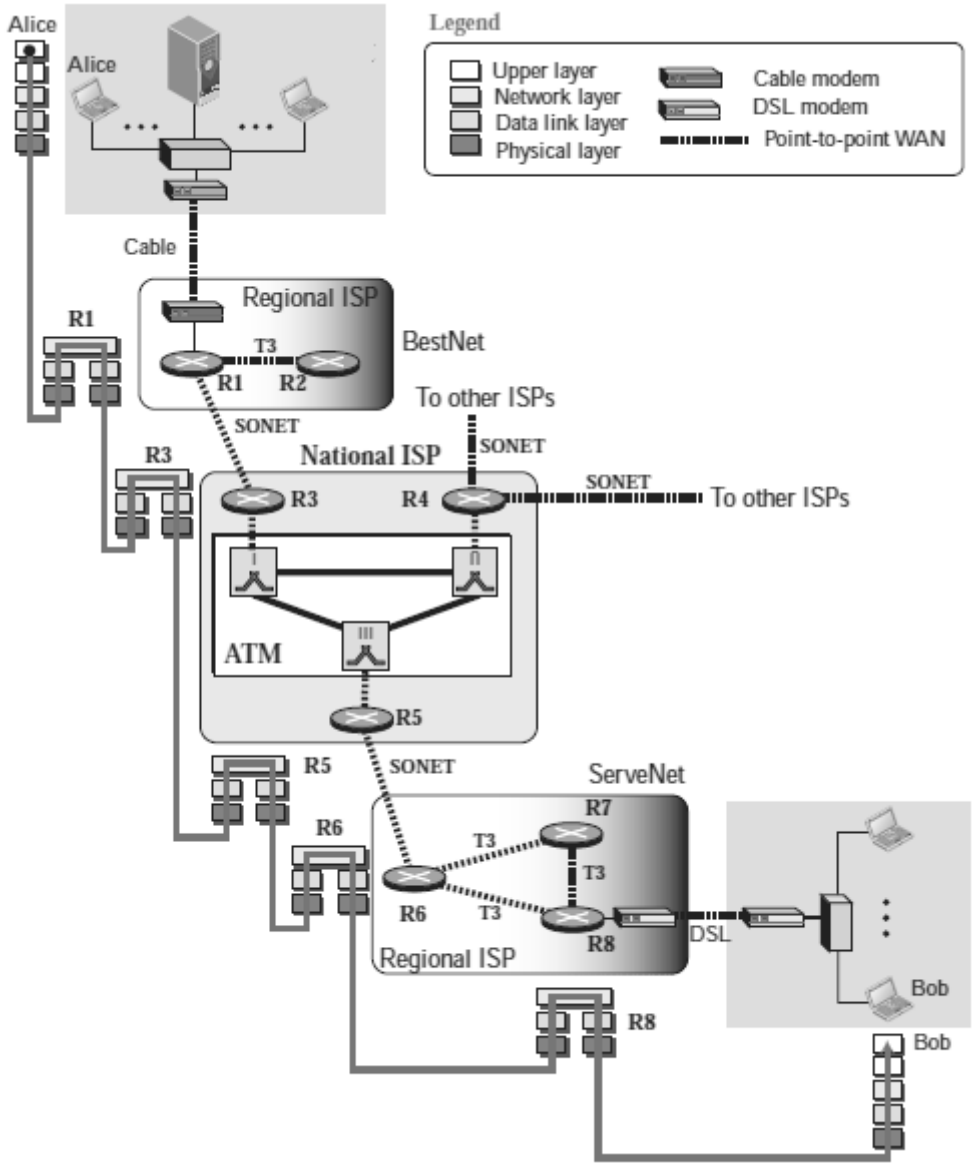
$$\vec{a} \cdot \vec{b} = a_1 b_1 + a_2 b_2 + a_3 b_3$$

5) Producto vectorial de dos vectores:

$$\vec{a} \times \vec{b} = (a_2 b_3 - a_3 b_2, \quad a_3 b_1 - a_1 b_3, \quad a_1 b_2 - a_2 b_1)$$

Escribe programa que defina una función para cada una de las operaciones vectoriales. Los vectores $\vec{a}$ y $\vec{b}$ se le pasan a la función como argumentos en forma de listas con tres elementos (las tres componentes del vector), y el escalar c como un argumento de tipo entero. Dependiendo de la operación a efectuar, la función devolverá un vector en forma de lista con tres elementos, o un entero (en el caso del producto escalar). En el propio programa, efectúa varias llamadas a las funciones definidas e imprime el resultado, para comprobar que todo funciona correctamente. Guarda el programa como `AmpAlgebraVectores.py`.

PARTE III:
INTRODUCCIÓN A REDES.



1. CONCEPTOS BÁSICOS DE REDES.

1.1. INTRODUCCIÓN.

La fusión de los ordenadores y las comunicaciones ha influido de forma determinante en la forma en la que se organizan los sistemas informáticos. El viejo modelo centralizado de un único gran ordenador para atender todas las necesidades de procesado se ha visto reemplazado por otro modelo con un gran número de ordenadores separados pero interconectados que realizan el trabajo. A estos sistemas se les conoce como **redes de ordenadores**.

A lo largo de este texto utilizaremos el término "red de ordenadores" para referirnos a un conjunto de ordenadores interconectados entre sí. Se dice que dos ordenadores están conectados si son capaces de intercambiar datos. Las redes pueden ser de distintas topologías, tamaños, y tecnologías. Además, las distintas redes suelen interconectarse entre sí para formar redes más grandes, siendo **Internet** el ejemplo más popular de red de redes o interred.

1.2. MODELOS DE FUNCIONAMIENTO.

Las redes de ordenadores son importantes porque son *útiles*. La capacidad de intercambiar datos entre ordenadores permite que las redes puedan prestar diversos servicios, como el acceso a páginas web, el correo electrónico, el comercio electrónico, la descarga de archivos, los escritorios compartidos, las redes sociales, las videollamadas en grupo, la formación on – line, etc. Dependiendo de la aplicación, las redes funcionan en base a dos modelos diferenciados: El modelo cliente – servidor, y el modelo igual a igual.

MODELO CLIENTE - SERVIDOR.

La mayoría de empresas disponen de ordenadores, tal vez incluso uno por empleado. En algún momento, es probable que se necesite conectar estas máquinas para **compartir recursos**, de forma que todos los equipos, programas, y datos estén disponibles para cualquier usuario de la red. Un ejemplo sencillo es un grupo de empleados que comparte una impresora: Una impresora en red de gran volumen es más económica, veloz, y fácil de mantener que una extensa colección de impresoras individuales. La capacidad de compartir datos es incluso más importante que la de compartir recursos físicos como impresoras. La mayoría de empresas tienen disponibles en línea registros de clientes, información de productos, inventarios, estados de cuentas, información fiscal, y muchos datos más.

Por concretar, supongamos que el sistema de información de una empresa está formado por varias bases de datos con información de la empresa, y por un cierto número de empleados que necesitan acceder de forma remota a esos datos. En este modelo, los datos se almacenan en potentes ordenadores llamados **servidores**. Por el contrario, los empleados cuentan con ordenadores mucho más básicos, denominados **clientes**, con los que acceden a los datos remotos. Las máquinas clientes y servidoras se interconectan mediante una red, como muestra la figura 1.1.

A este tipo de configuración se le llama **modelo cliente – servidor**. Es un modelo ampliamente usado y constituye la base de muchas redes. Un ejemplo típico es una **aplicación web**, en la que un servidor proporciona páginas web basadas en la información alojada en su base de datos en respuesta a las solicitudes de los clientes. El modelo cliente – servidor es aplicable cuando el cliente y el servidor se encuentran en la misma oficina física, pero también cuando están muy lejos. Por ejemplo, cuando un usuario accede desde su casa a una página en la **World Wide Web** se emplea el mismo modelo, donde el servidor web remoto desempeña el papel de servidor, y el PC del usuario actúa como cliente. En la mayoría de los casos, un solo servidor es capaz de gestionar las solicitudes de cientos o miles de clientes.

En el modelo cliente – servidor hay dos **procesos** (esto es, dos programas en ejecución): Uno en la máquina cliente y otro en la máquina servidora. La comunicación empieza cuando el **proceso cliente** envía un mensaje a través de la red al **proceso servidor**. Entonces, el proceso cliente queda a la espera de un mensaje de respuesta. Cuando el proceso servidor recibe la solicitud, lleva a cabo la tarea solicitada, busca los datos requeridos, y devuelve una respuesta.

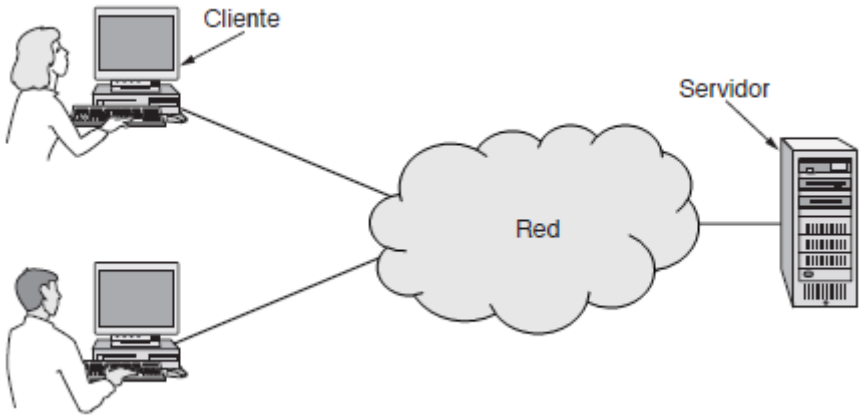


Figura 1.1. Red con dos clientes y un servidor.

MODELO DE IGUAL A IGUAL.

Para acceder a una gran parte de la información disponible en Internet se utiliza el modelo cliente – servidor. Sin embargo, en los últimos años ha emergido un nuevo modelo de comunicación de **igual a igual** (*peer to peer*) en el que los usuarios pueden comunicarse directamente entre sí sin tener que interactuar con un servidor (ver figura 1.2). Muchos sistemas de igual a igual, como **BitTorrent**, no tienen una base de datos centralizada que aloje los contenidos. En cambio, cada usuario mantiene su propia base de datos de forma local, y provee una lista de usuarios cercanos que son miembros del mismo sistema. De esta forma los usuarios pueden consultar la información que dispone cualquier miembro y obtener los nombres de otros miembros para inspeccionar si hay más contenidos y más nombres. Como probablemente ya sabemos, la comunicación de igual a igual se usa con frecuencia para compartir archivos.

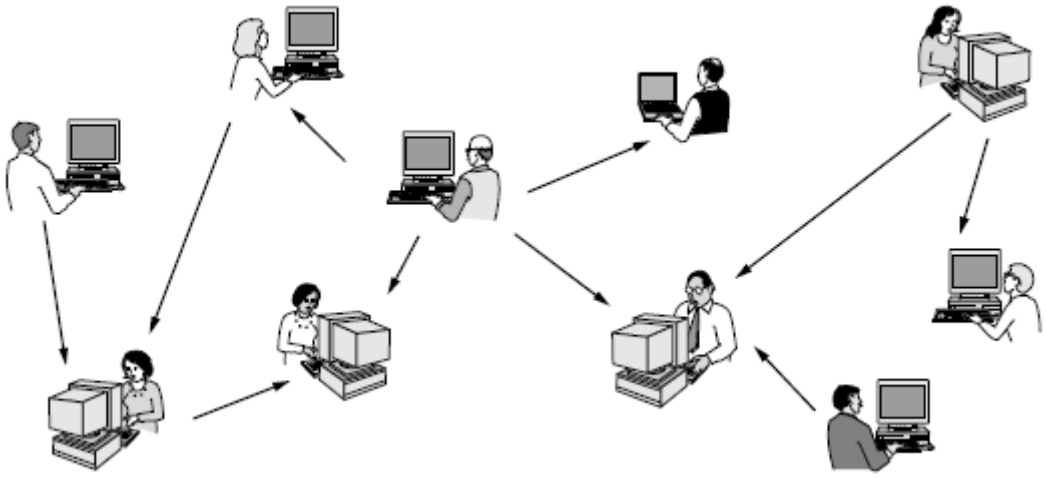


Figura 1.2. En los sistemas de igual a igual no hay servidores ni clientes.

Otro uso importante de las redes de igual a igual es la comunicación persona a persona. Hoy día muchos usuarios utilizan la **mensajería instantánea** como herramienta para intercambiar mensajes de texto en tiempo real (aunque actualmente estos mensajes también pueden incluir contenidos multimedia como fotos, vídeos, u otros archivos). Otro tipo de aplicaciones que se encuentran entre los sistemas de comunicación persona a persona son las **redes sociales**. Aquí el flujo de información se controla mediante las relaciones que los usuarios se declaran entre sí. Un ejemplo es Facebook: Este sitio permite que los usuarios

actualicen sus perfiles y compartan estas actualizaciones con otros usuarios que estén declarados como amigos.

1.3. SISTEMAS DE COMUNICACIONES.

Una **red de ordenadores** es un conjunto de ordenadores interconectados entre sí que son capaces de intercambiar datos. Para que haya comunicación de datos, los dispositivos en comunicación deben formar parte de un **sistema de comunicaciones** compuesto por una combinación de hardware (equipos físicos) y de software (programas).

COMPONENTES DE UN SISTEMA DE COMUNICACIONES.

Un sistema de comunicaciones tiene cinco componentes básicos (ver figura 1.3):

1) Mensaje: El **mensaje** es la información (los datos) que se desean comunicar, y puede ser texto, números, imágenes, audio, y video.
2) Emisor: El **emisor** es el dispositivo que envía el mensaje. Puede tratarse de un ordenador, una estación de trabajo, un teléfono, etc.
3) Receptor: El **receptor** es el dispositivo que recibe el mensaje. De nuevo, puede ser un ordenador, una impresora, un teléfono, etc.
4) Medio de transmisión: El **medio de transmisión** es el camino físico por el que viaja el mensaje de emisor a receptor. Algunos ejemplos son el cable de pares, el cable coaxial, el cable de fibra óptica, y las ondas de radio.
5) Protocolo: El **protocolo** es el conjunto de reglas que gobierna la comunicación de los datos, y representa un acuerdo entre los dispositivos en comunicación. Sin un protocolo, dos dispositivos podrían estar conectados pero no comunicarse, de la misma forma que una persona que solo hable francés no puede entenderse con otra persona que solo hable japonés.

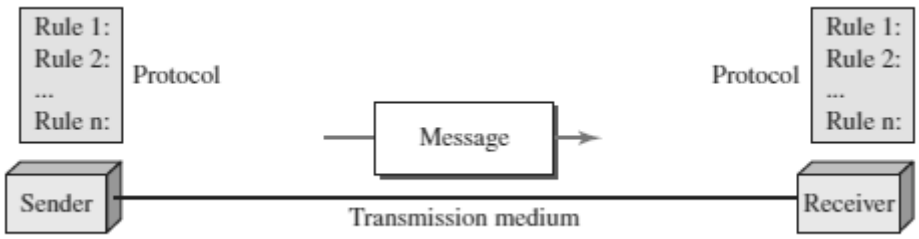


Figura 1.3. Componentes de un sistema de comunicación de datos.

REPRESENTACIÓN DE LOS DATOS.

La información que transmiten los sistemas de comunicaciones aparece en formas muy diversas, incluyendo textos, números, imágenes, audio, y vídeo. Actualmente todos los datos se representan en formato digital, esto es, como secuencias de **bits** (0's y 1's). (Ver anexo A al final del texto). Hay muchas formas de representar la información en forma digital, y en esta sección solo daremos una breve introducción a algunas de ellas.

Textos.

Existen varios códigos para representar digitalmente los distintos caracteres que se usan en los textos. El primer sistema de codificación internacionalmente aceptado se denominó **ASCII** (American Stardard Code for Information Interchange), e incluía 128 símbolos representados con cadenas de 7 bits. ASCII abarca las letras mayúsculas y minúsculas, los dígitos numéricos del 0 al 9, los caracteres de puntuación, y algunos caracteres de control (ver anexo B). Más adelante aparecería el ASCII extendido, un intento de ampliar

ASCII a 8 bits que nunca llegó a estandarizarse internacionalmente. Posteriormente se creó el **Unicode**, un sistema de codificación universal que usa 32 bits para representar todos los símbolos empleados en cualquier idioma del mundo.

Números.

Los números también se representan como secuencias de bits. Pero para representar los dígitos de los números no se emplean códigos como el ASCII; en vez de eso, el número decimal se convierte directamente a número binario para simplificar las operaciones matemáticas. (Ver anexo C).

Imágenes.

Como los textos y los números, las imágenes también se representan mediante patrones de bits. En su forma más simple, una imagen es una matriz de píxeles, donde cada **píxel** es un pequeño punto. El tamaño de cada píxel depende de la *resolución*. Por ejemplo, una imagen puede dividirse en 1000 píxeles o en 10000 píxeles. En el segundo caso tenemos una mejor representación de la imagen (una mejor resolución), pero se necesita más memoria para almacenarla.

Después de dividir la imagen en píxeles, a cada píxel se le asigna un patrón de bits. El tamaño y el valor del patrón de bits dependen de la imagen. Para una imagen que solo esté formada por puntos blancos y negros, solo hace falta 1 bit para representar cada píxel. Pero en una imagen que incluya la escala de grises debemos aumentar el tamaño de los patrones de bits que representan cada píxel. Por ejemplo, para mostrar cuatro niveles de gris (blanco y negro incluidos), debemos usar cadenas de 2 bits. Un píxel negro puede representarse mediante un 00, un píxel gris oscuro mediante un 01, un píxel gris claro mediante un 10, y un píxel blanco mediante un 11.

Hay varios métodos para representar las imágenes a color. Una técnica es la denominada RGB (Red Green Blue), llamada así porque cada color se obtiene mediante una combinación de tres colores primarios: Rojo, verde, y azul. La técnica consiste en medir la intensidad de cada color, y asignar un patrón de bits en base a ella. Otra técnica es la llamada YCM (Yellow Cyan Magenta), en la que cada color es una combinación de otros tres colores primarios: Amarillo, cian, y magenta.

Audio.

El audio se refiere a la grabación y a la difusión de sonido o música. Por naturaleza, el audio es diferente de los textos, números, o imágenes, ya que se trata de una información que es continua, no discreta. Incluso cuando usamos un micrófono para convertir la voz o la música en una señal eléctrica, estamos creando una seña continua. Por consiguiente, la digitalización de la señal de audio implica como primer paso su muestreo, esto es, la conversión de una señal continua a una señal discreta. Después, cada una de las muestras tomadas se codifica mediante un patrón de bits, cuyo número depende de la calidad de la señal digital deseada.

Video.

La señal de video puede verse coma la combinación de una señal de audio, y de una sucesión de imágenes que conjuntamente dan la sensación de movimiento. Por lo tanto, la digitalización de un video incluye la digitalización de un audio y de una secuencia de imágenes.

FLUJO DE LOS DATOS.

La comunicación entre dos dispositivos puede ser simplex, half – dúplex, o full – dúplex (ver figura 1.4):

Simplex.

En **modo simplex** la comunicación es unidireccional. Solo uno de los dos dispositivos del enlace puede transmitir; el otro solo puede recibir. El modo simplex puede usar toda la capacidad del canal para enviar datos en una dirección. Un ejemplo es el antiguo sistema de difusión de televisión: La antena difusora sólo puede transmitir, y los televisores solo pueden recibir.

Half – dúplex.

En **modo half – dúplex** (o **semi - dúplex**) cada estación puede transmitir y recibir, pero no al mismo tiempo. Cuando un dispositivo está enviando, el otro solo puede recibir (y viceversa). En una transmisión half – dúplex el dispositivo que está transmitiendo en un instante dado utiliza toda la capacidad del canal. Los walkie – talkies son un ejemplo de sistema half – dúplex.

Full – dúplex.

En el **modo full – dúplex** (o simplemente **dúplex**) ambas estaciones pueden transmitir y recibir simultáneamente. En modo full – dúplex las señales que viajan en una dirección comparten la capacidad del enlace con las señales que viajan en la otra dirección. Se puede compartir el enlace de dos formas: (1) El enlace puede tener dos rutas de transmisión físicamente independientes, una para enviar y otra para recibir (como por ejemplo, usando dos cables distintos para transmitir y para recibir), o (2) la capacidad del canal puede dividirse entre las señales que viajan en ambas direcciones (por ejemplo, usando un solo cable y dos canales distintos para enviar y para recibir). Un ejemplo de comunicación full – dúplex es la red telefónica: Cuando dos personas hablan por teléfono, ambas pueden hablar y escuchar al mismo tiempo.

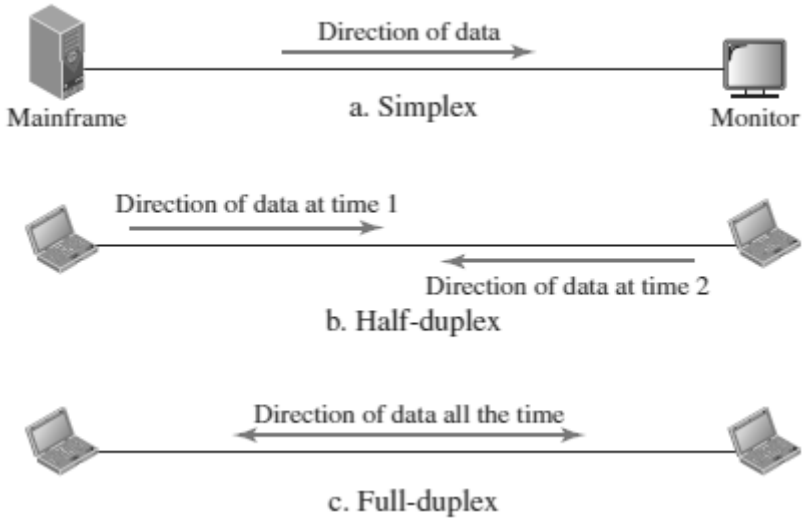


Figura 1.4. Sistemas simplex, semi – dúplex, y full – dúplex.

1.4. REDES.

Una **red** es la interconexión de un conjunto de dispositivos capaces de comunicarse. Los dispositivos que constituyen una red pueden clasificarse en dos grandes grupos: (1) Los **hosts** son los dispositivos finales de la comunicación, como un PC, un portátil, una impresora, un teléfono móvil, etc. (2) Por otro lado, los **dispositivos de conexión** son los equipos que permiten interconectar al resto de dispositivos de la red, pero que no son los receptores finales de la información. Algunos ejemplos son los concentradores (hubs) y los conmutadores (switches) que permiten interconectar los distintos dispositivos de una red, los rúters que sirven para interconectar unas redes con otras, los modems (modem = modulador - demodulador) que cambian la forma de los datos, etc. Los distintos dispositivos de la red se conectan usando medios de transmisión alámbricos (cables) o inalámbricos (a través del aire).

TIPO DE CONEXIÓN.

Una red consiste en dos o más dispositivos conectados a través de enlaces. Un **enlace** es una vía de comunicación que transfiere datos de un dispositivo a otro, y que podemos imaginar como una línea dibujada entre dos puntos. Para que dos dispositivos puedan comunicarse, ambos deben conectarse de alguna forma al mismo enlace, al mismo tiempo. Podemos distinguir dos tipos posibles de conexiones: Punto a punto y multipunto.

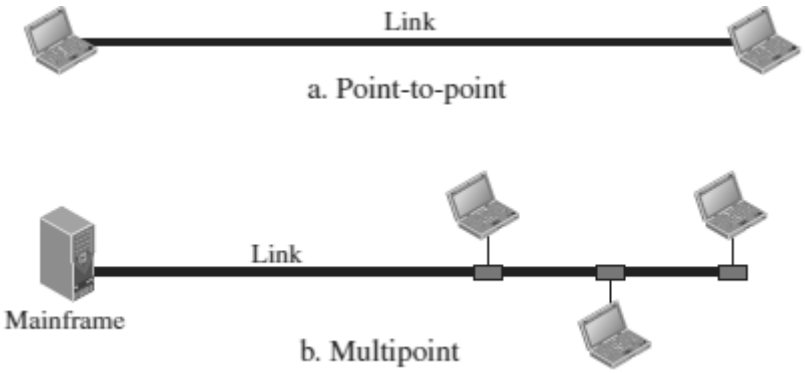


Figura 1.5. Tipos de conexión.

Conexión punto a punto.

Una **conexión punto a punto** proporciona un **enlace dedicado** entre dos dispositivos. (El término *dedicado* significa que ese enlace es exclusivo y solo transporta el tráfico perteneciente a los dos dispositivos que conecta). Por consiguiente, en una conexión punto a punto se reserva toda la capacidad del canal a la transmisión entre esos dos dispositivos. Muchas conexiones punto a punto utilizan un cable para conectar los dispositivos, pero también es posible usar un radioenlace de microondas, o un enlace vía satélite.

Conexión multipunto (conexión de difusión).

Una **conexión multipunto** (también llamada **conexión de difusión**) es aquella en la que más de dos dispositivos comparten el mismo enlace. En una conexión multipunto los dispositivos comparten la capacidad del canal, ya sea transmitiendo en distintos intervalos de tiempo, en distintas bandas de frecuencia, etc.

TIPOS DE TOPOLOGÍA FÍSICA.

El término **topología física** se refiere a la disposición física de la red, y muestra una representación geométrica de los dispositivos conectados (habitualmente llamados **nodos**) y de los enlaces que los interconectan. Existen cuatro topologías básicas posibles:

Topología en malla.

En una **topología en malla** cada dispositivo tiene un enlace punto a punto dedicado con todos los demás dispositivos. En una red mallada con n nodos, cada nodo individual debe conectarse a $n-1$ nodos de la red, por lo que se necesitan $n(n-1)$ enlaces físicos. Pero si cada enlace permite la comunicación en dos direcciones (modo dúplex), el número total de enlaces full – dúplex necesarios en una red totalmente mallada es de $n(n-1)/2$. Ello implica que cada dispositivo debe tener $n-1$ puertos de entrada/salida para poder conectarse a las otras $n-1$ estaciones.

Una red mallada ofrece varias ventajas sobre el resto de topologías: En primer lugar, el uso de enlaces dedicados garantiza que cada conexión transporte únicamente su propia carga de datos, lo que elimina los problemas de tráfico que pueden ocurrir en los enlaces compartidos. En segundo lugar, las topologías en malla son robustas: Si un enlace deja de estar operativo, esto no incapacita al sistema completo. Por último,

los enlaces punto a punto facilitan la detección de averías y su aislamiento. En tales casos, el tráfico puede reencaminarse para evitar los enlaces afectados por el fallo, y la red puede detectar la localización precisa de la avería, su causa, y su solución.

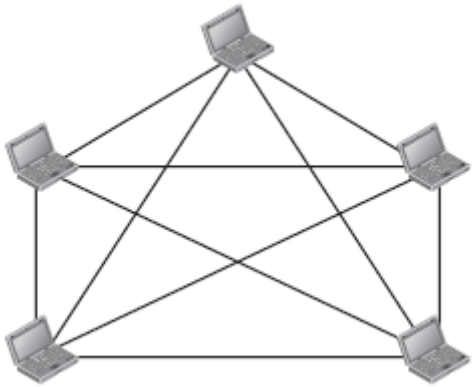


Figura 1.6. Una red totalmente mallada con $n = 5$ dispositivos y 10 enlaces.

Las principales desventajas de una red mallada están relacionadas con la cantidad de cables necesarios y el número de puertos de entrada/salida en cada dispositivo. En primer lugar, como cada dispositivo debe conectarse a todos los demás, la instalación y reconfiguración de la red son bastante complejas. En segundo lugar, el volumen del cableado necesario puede ser mayor que el espacio disponible (en paredes, techos, o suelos) para alojarlo. Finalmente, el hardware requerido para interconectar todos los dispositivos (puertos de entrada/salida y cables) puede llegar a ser inasumible económicamente. Por todas estas razones las topologías en malla solo se usan de forma ocasional, por ejemplo, en el segmento troncal de las grandes redes de telecomunicaciones (redes telefónicas públicas, redes de proveedrores de servicios de Internet, etc.).

Topología en estrella.

En una **topología en estrella** cada dispositivo posee un enlace dedicado que lo conecta únicamente a un elemento de interconexión central, típicamente un **concentrador** (*hub*) o un **conmutador** (*switch*). En esta topología los dispositivos no están directamente interconectados unos a otros, y no hay tráfico directo entre ellos. Si un dispositivo quiere enviar datos a otro, este dispositivo transmite los datos al elemento de conexión central, que a continuación los retransmite al otro dispositivo conectado.

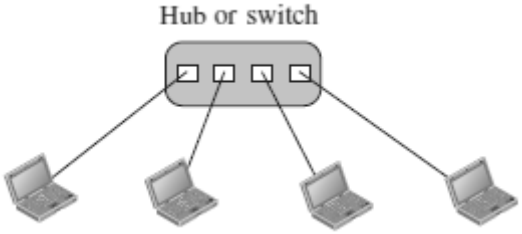


Figura 1.7. Topología en estrella entre cuatro dispositivos.

Una topología en estrella es mucho menos cara que una topología en malla. En una topología en estrella, cada dispositivo solo necesita un enlace y un puerto de entrada/salida para conectarse a cualquier número de dispositivos. Esta característica también facilita la instalación y reconfiguración de la red: Se necesita alojar mucho menos cable, y la adición, traslado, o retirada de un dispositivo solo implica una conexión, a saber, la de ese dispositivo con el elemento central.

Otra ventaja es la robustez: Si un enlace falla, solo él se ve afectado; el resto de enlaces siguen activos. Ello implica que la identificación y el aislamiento de un fallo son más sencillos. Mientras el elemento central siga funcionando, puede usarse para monitorizar problemas en los enlaces.

La principal desventaja de la topología en estrella es la dependencia de la red respecto al elemento central. Si este dispositivo cae, se cae el sistema completo. Y aunque una red en estrella requiere mucho menos cableado que una red en malla, cada dispositivo todavía tiene que conectarse al elemento central. Por esta razón, esta topología requiere más cable que otras topologías (como las topologías en anillo o en bus, de las que hablaremos a continuación).

La topología en estrella suele usarse en **redes de área local** (**LANs**). Por ejemplo, las LANs de alta velocidad suelen usar una topología en estrella con un elemento de conexión central.

Topología en bus.

Las dos topologías previas usaban conexiones punto a punto. En cambio, una **topología en bus** es multipunto. En las topologías en bus hay un cable largo, denominado **bus**, al que se conectan todos los dispositivos de la red.

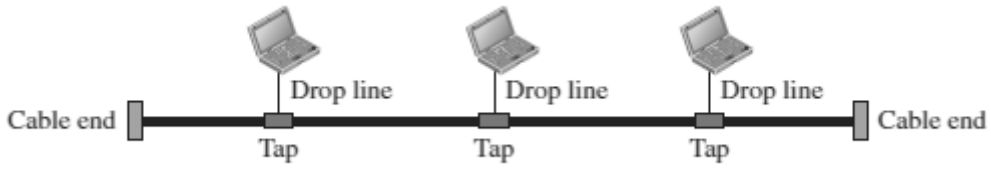


Figura 1.8. Una topología en bus conectando tres estaciones.

Las estaciones se conectan al cable usando líneas de derivación y tomas. Una línea de derivación (*drop line*) es una conexión entre el dispositivo y el cable principal. Una toma (*tap*) es un conector que empalma con el cable principal. Conforme la señal viaja por el cable principal parte de su energía se transforma en calor, y se va debilitando. (A este proceso se le denomina *atenuación*). Por esta razón, hay un límite en el número de tomas que un bus puede soportar y en la distancia entre esas tomas.

La principal ventaja de la topología en bus es su fácil instalación. El cable central puede ponerse a lo largo de la ruta más eficiente, para después conectarse a las estaciones mediante líneas de derivación de distintas longitudes. De esta forma, la topología en bus usa menos cableado que las topologías en malla o en estrella. Por ejemplo, en una topología en estrella con cuatro dispositivos de red en una misma habitación se necesitan cuatro cables que lleguen hasta la ubicación del elemento de interconexión central. En una topología en bus el cable central se extiende a lo largo de toda la instalación, y las líneas de derivación únicamente deben llegar al punto más cercano de ese cable.

Las desventajas son la dificultad de reconexión y el aislamiento de las averías. La topología en bus suele diseñarse para optimizar la instalación, y en consecuencia, puede resultar difícil añadir nuevos dispositivos a la red. La reflexión de las señales en las tomas puede originar una degradación en su calidad. Esta degradación puede controlarse limitando el número y el espaciado entre los dispositivos conectados a una cierta longitud de cable, por lo que la adición de nuevos dispositivos podría implicar una modificación o cambio del enlace central. Sin embargo, una avería o ruptura del bus detendría todas las transmisiones.

La topología en bus fue una de las topologías usadas en el diseño de las primeras redes de área local. Actualmente, este tipo de redes son menos populares que las redes locales con topología en estrella, por razones que veremos en su momento.

Topología en anillo.

En una **topología en anillo** cada dispositivo tiene una conexión punto a punto dedicada únicamente con los dos dispositivos adyacentes (ver figura 1.9). La señal se pasa a lo largo del anillo en una dirección, de dispositivo en dispositivo, hasta que llega a su destino. Cada dispositivo del anillo incorpora un **repetidor**:

Cuando un dispositivo recibe una señal destinada a otro dispositivo, su repetidor regenera los bits y los vuelve a pasar al cable.

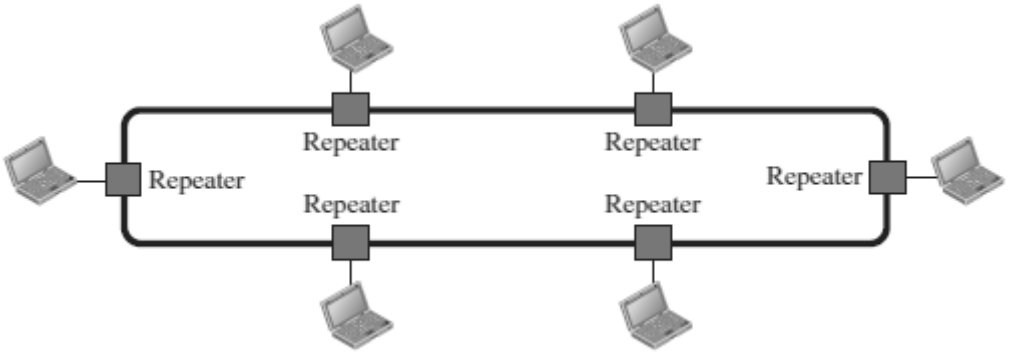


Figura 1.9. Topología en anillo conectando seis estaciones.

Una red en anillo es relativamente fácil de instalar y reconfigurar. Cada dispositivo se conecta solamente a sus vecinos más inmediatos (ya sea físicamente o lógicamente). Para añadir o quitar dispositivos de la red, solo debemos cambiar dos conexiones. Las únicas limitaciones vienen dadas por la longitud máxima del anillo y el número máximo de dispositivos. Además, este tipo de redes simplifica el aislamiento de las averías. Generalmente, en un anillo la señal está circulando todo el tiempo. Si un dispositivo no recibe señal en un periodo de tiempo dado, puede emitir una alarma. La alarma alerta al operador de la red del problema y de su localización. Sin embargo, un anillo unidireccional puede ser un inconveniente. En un anillo simple, un hipotético corte (como por ejemplo, una estación caída) puede deshabilitar la red completa. Esta debilidad puede resolverse, por ejemplo, con un anillo doble.

La topología en anillo fue muy frecuente en la tecnología Token Ring desarrollada por IBM para implementar redes de área local. Hoy día, la necesidad de redes LAN más rápidas ha hecho esta topología menos habitual en este tipo de redes.

1.5. TIPOS DE REDES.

En esta sección vamos a discutir los distintos tipos de redes que podemos encontrar hoy día. Para establecer la clasificación hemos usado criterios muy diversos, como el tamaño, la cobertura geográfica, y la titularidad de la red.

REDES DE ÁREA LOCAL.

Una **red de área local** (**LAN** = Local Area Network) suele ser una red de propiedad privada que conecta unos cuantos hosts en un aula, oficina, edificio, o campus. Dependiendo de las necesidades de la organización, una LAN puede ser tan simple como dos PCs y una impresora en el domicilio de un usuario, o puede extenderse por toda una gran empresa e incluir dispositivos de audio y video. Cada host de la LAN tiene un identificador (una dirección) que define de manera unívoca al host dentro de la LAN. Cualquier paquete enviado por un host a otro host de la red lleva las direcciones de los hosts origen y destino.

En el pasado todos los hosts de una red LAN se conectaban a un cable común (bus) o a un concentrador (hub), lo que implicaba que todo paquete enviado por un host a otro también llegaba al resto de hosts de la red. El destinatario previsto conservaba el paquete, y los demás hosts lo descartaban. Hoy día, la mayoría de LANs usan un **conmutador** inteligente (*switch*) capaz de reconocer la dirección de destino del paquete, y de enviarlo al destinatario correcto sin difundirlo al resto de hosts. El conmutador disminuye el tráfico en la LAN y permite que más de dos hosts se comuniquen simultáneamente si no hay un emisor o un receptor común entre ellos. Notar que nuestra definición de LAN no especifica un número mínimo o máximo de hosts en la red. La figura muestra tanto una antigua LAN con cable común y una moderna LAN conmutada.

Cuando las LANs se usaban aisladas (lo cual es muy inusual hoy día), se diseñaban para compartir recursos entre los hosts de la red. Actualmente las LANs suelen conectarse unas a otras, y también a redes WAN (de las que hablaremos a continuación), para establecer comunicaciones de más alto nivel.

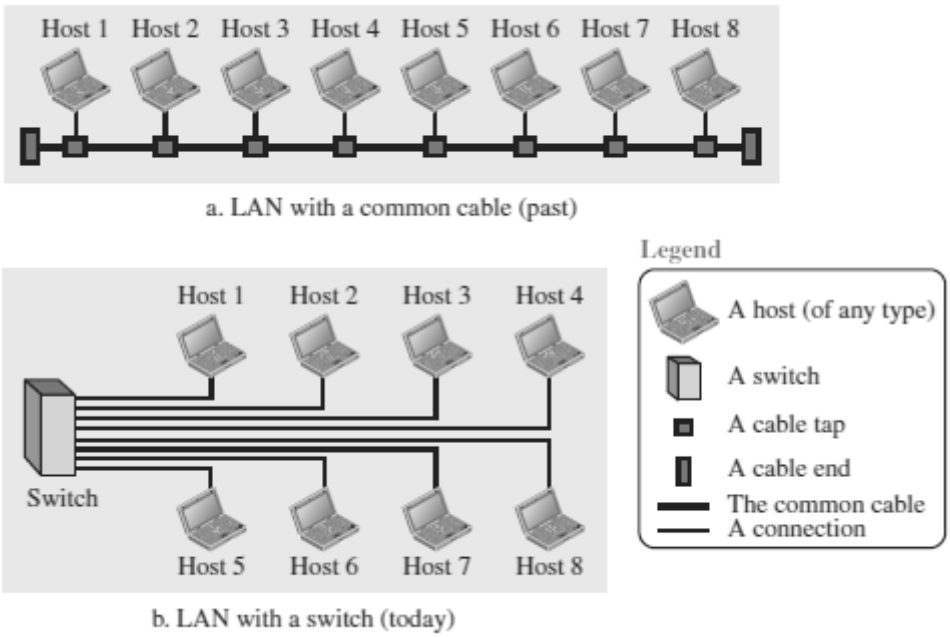


Figura 1.10. Una LAN aislada en el pasado y en la actualidad.

REDES DE ÁREA AMPLIA.

Una **red de área amplia** (**WAN** = Wide Area Network) es una red que interconecta dos o más LANs, normalmente a través de líneas alquiladas a una red telefónica pública. Como en el caso de una red LAN, una red WAN también es una interconexión de dispositivos capaces de comunicarse. Sin embargo, existen algunas diferencias importantes que nos permiten diferenciarlas: (1) Una LAN suele tener un tamaño limitado, y abarca una oficina, un edificio, o un campus; por el contrario, una WAN tiene una cobertura geográfica mucho más amplia, y se extiende a lo largo de una ciudad, un país, o incluso el mundo. (2) Una LAN interconecta hosts; una WAN interconecta dispositivos de conexión como conmutadores, rúters, o módems. (3) Una LAN suele ser de propiedad privada y solo pertenece a la organización que la usa; una WAN suele pertenecer a compañías de telecomunicaciones que prestan sus servicios a las organizaciones que los contratan. Podemos distinguir dos tipos de redes WAN: Las WAN punto a punto y las WAN conmutadas:

Una **WAN punto a punto** es una red WAN que conecta dos dispositivos remotos mediante un enlace punto a punto dedicado, como por ejemplo, una línea disponible en una red pública (como la red telefónica). Algunos ejemplos típicos de WANs punto a punto son las conexiones a Internet mediante líneas ADSL, redes de cable, o redes de fibra óptica. La figura 1.11 muestra un ejemplo de WAN punto a punto.

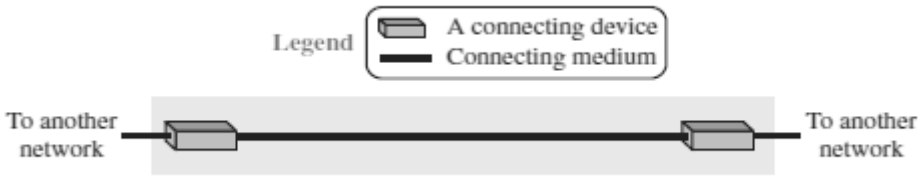


Figura 1.11. WAN punto a punto.

Una **WAN conmutada** es una red WAN con una amplia extensión geográfica (típicamente, un estado o un país) que proporciona acceso a los usuarios en varios puntos. Las redes troncales de los sistemas de comunicaciones globales y de Internet suelen ser WANs conmutadas. El interior de una WAN conmutada está formado por una malla de WANs punto a punto interconectadas mediante conmutadores (switches).

Esos conmutadores permiten la interconexión de cualquiera de sus líneas de entrada con cualquiera de sus líneas de salida. La figura 1.12 ilustra un ejemplo de WAN conmutada.

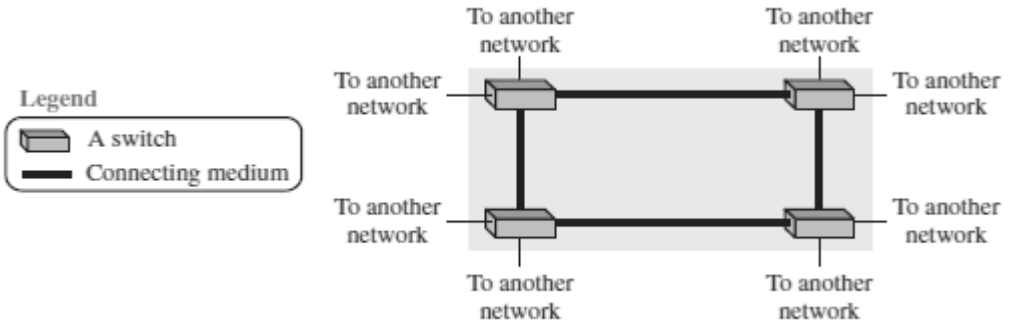


Figura 1.12. WAN conmutada.

INTERREDES.

Hoy día es muy raro tener una LAN o una WAN aisladas. En la práctica, las redes suelen interconectarse unas con otras para formar una **interred** o **internet** (con *i* minúscula).

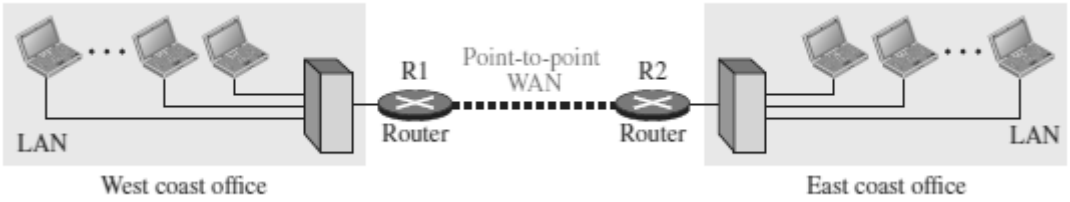


Figura 1.13. Una interred formada por dos LANs y una WAN punto a punto.

A modo de ejemplo, supongamos que una organización tiene dos oficinas, una en la costa este y otra en la costa oeste. En cada oficina hay una LAN que permite a los empleados de la oficina comunicarse entre sí. Para que los empleados en diferentes oficinas puedan comunicarse, el gerente de la empresa alquila un enlace punto a punto dedicado (una WAN punto a punto) a un proveedor de servicios de comunicaciones, como una compañía telefónica, para interconectar las dos LANs. A partir de ese momento, la empresa dispone de una interred o internet privada (ver figura 1.13). Cuando un host en la oficina de la costa oeste envía un mensaje a otro host en la misma oficina, el rúter bloquea el mensaje, pero el conmutador lo redirige al destino apropiado. Por otro lado, cuando un host en la costa oeste envía un mensaje a un host en la costa este, el rúter R1 encamina el paquete hacia el rúter R2, y el paquete alcanza su destino.

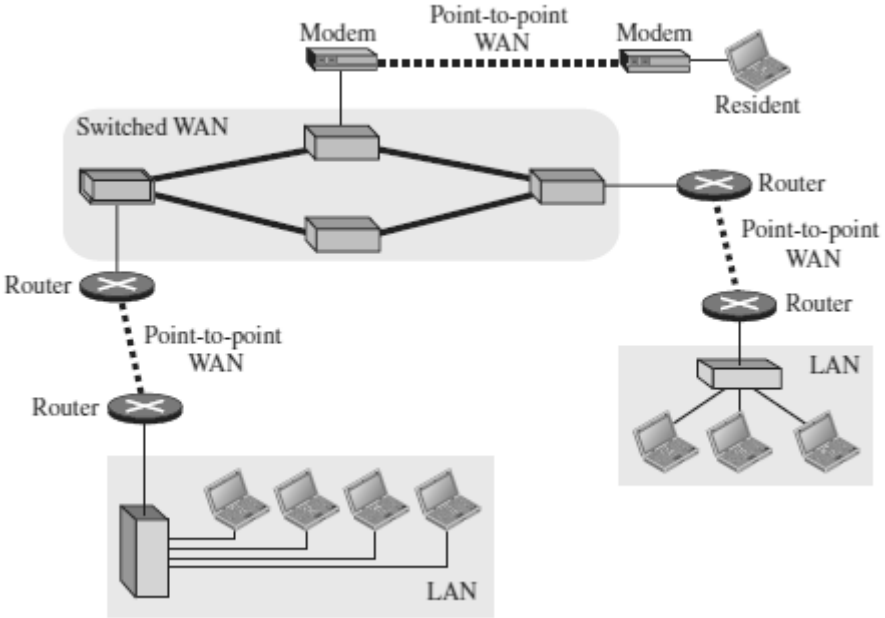


Figura 1.14. Una interred heterogénea formada por cuatro WANs y tres LANs.

La figura 1.14 muestra otra interred con varias LANs y WANs conectadas. Una de las WAN es una WAN conmutada con cuatro conmutadores. Para otros ejemplos de interred, ver la figura de la portada y la figura 1.15 en la siguiente sección.

Habitualmente, la conexión entre dos hosts a través de Internet pasa por múltiples redes, tanto LANs como WANs. Por ejemplo, en la figura de la portada, la comunicación entre los ordenadores Alice y Bob discurre a través de las LANs a la que pertenecen Alice y Bob, las WANs punto a punto que conectan a Alice y a Bob a sus respectivos ISPs regionales (sendas líneas de cable y ADSL, respectivamente), y las distintas WANs conmutadas de los ISPs internacionales que constituyen las redes troncales de Internet.

CONMUTACIÓN.

Las interredes de hoy en día están formadas por muchas redes LAN y WAN interconectadas a través de dispositivos de conexión (conmutadores y rúters). Uno de los problemas más importantes a resolver es cómo mover los datos desde el host origen al host destino a través del conjunto de redes y dispositivos de conexión que constituyen una interred (ver figura 1.15).

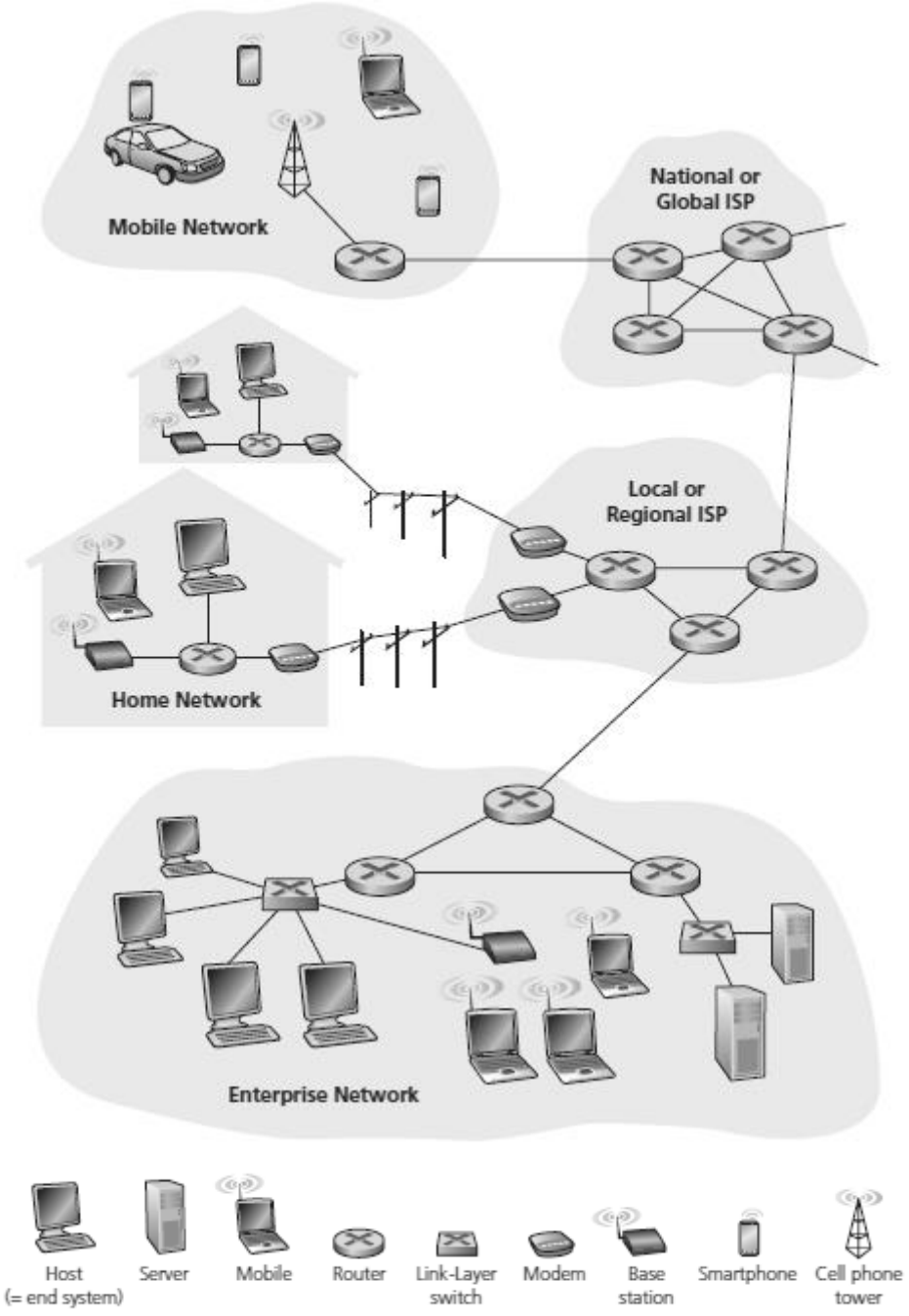


Figura 1.15. Una interred como una combinación de LANs y WANs.

En este tipo de interredes, los elementos de interconexión (switches y rúters) actúan como **conmutadores**. Cuando los datos llegan a uno de sus puertos de entrada, el conmutador los reenvía a través de uno de sus puertos de salida hacia el siguiente conmutador de la ruta, o hacia el host de destino. A este proceso se le denomina **conmutación**. Las redes pueden operar en base a dos técnicas de conmutación distintas: La conmutación de circuitos, y la conmutación de paquetes.

Redes de conmutación de circuitos.

En una red de **conmutación de circuitos** siempre se establece un circuito físico (una conexión dedicada) entre el origen y el destino antes de comenzar a enviar los datos. Tras establecer la conexión, los datos se transmiten íntegramente. Una vez enviados todos los datos, el emisor informa a la red de que la transmisión ha finalizado. En ese momento, la red libera todos los enlaces y dispositivos de conexión involucrados en esa transmisión para poder usarlos en otra conexión.

El ejemplo más habitual de red de conmutación de circuitos es la antigua red telefónica. Cuando el usuario que llamaba marcaba el número de teléfono del usuario con el que quería hablar, la red creaba un circuito exlusivo entre el terminal origen y el terminal destino. Cuando el usuario receptor levantaba el teléfono el circuito ya estaba establecido, y la comunicación de voz podía cursarse sin interrupciones en ambas direcciones. Cuando el emisor o el receptor colgaba el teléfono el circuito se desconectaba, y los recursos empleados se liberaban, quedando a disposición de una nueva llamada telefónica.

La figura 1.16 muestra una red telefónica conmutada muy sencilla, que conecta cuatro teléfonos en cada extremo. En la figura, cada grupo de teléfonos está conectado a un conmutador (switch). Los conmutadores conectan un terminal telefónico a un lado con un terminal telefónico al otro lado. La línea gruesa que conecta los dos conmutadores representa una línea de comunicación de alta capacidad capaz de cursar cuatro comunicaciones de voz simultáneas. La capacidad de esta línea se comparte entre todas las parejas de terminales telefónicos.

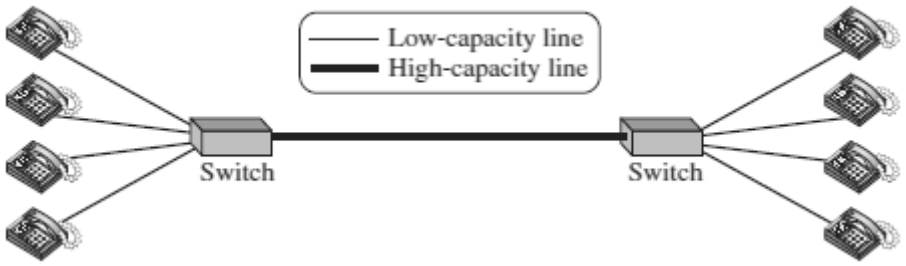


Figura 1.16. Una red de conmutación de circuitos.

Vamos a analizar dos casos: En el primer caso, todos los terminales telefónicos están ocupados, esto es, cuatro personas en un extremo están hablando con otras cuatro personas en el otro extremo; la capacidad de la línea gruesa está siendo completamente usada. En el segundo caso solo hay un terminal telefónico en un extremo conectado a un terminal en el otro extremo; aquí solo se está usando un cuarto de la capacidad de la línea gruesa. Esto significa que una red de conmutación de circuitos solo es eficiente cuando opera a su capacidad máxima; la mayor parte del tiempo es ineficiente, porque solo trabaja a una parte de su capacidad total. La razón por la que hemos elegido que la capacidad de la línea gruesa sea cuatro veces la capacidad de cada línea de voz es porque no queremos que la comunicación falle cuando todos los terminales telefónicos en un extremo quieran conectarse con todos los terminales telefónicos en el otro extremo. En la práctica, las líneas de alta capacidad no se diseñan de esta forma, porque serían muy costosas e ineficientes.

Redes de conmutación de paquetes.

Las redes de ordenadores de hoy día son redes de **conmutación de paquetes**. En este tipo de redes, los mensajes que la estación emisora desea transmitir se dividen en bloques de tamaño manejable denominados **paquetes**, que se envían uno a uno a través de la red. La estación receptora recibe los paquetes uno a uno, y espera a que lleguen todos para volver a ensamblar el mensaje original. En otras palabras, en vez de tener la comunicación continua que vemos entre dos terminales telefónicos, los ordenadores intercambian paquetes de datos individuales. Esto implica que los conmutadores (rúters, en este caso) deben funcionar tanto reenviando paquetes como almacenándolos, porque cada paquete es una entidad independiente de todos los demás, y puede almacenarse para ser reenviado más tarde. La figura 1.17 muestra una pequeña red de conmutación de paquetes que conecta cuatro ordenadores en un extremo con cuatro ordenadores en otro extremo.

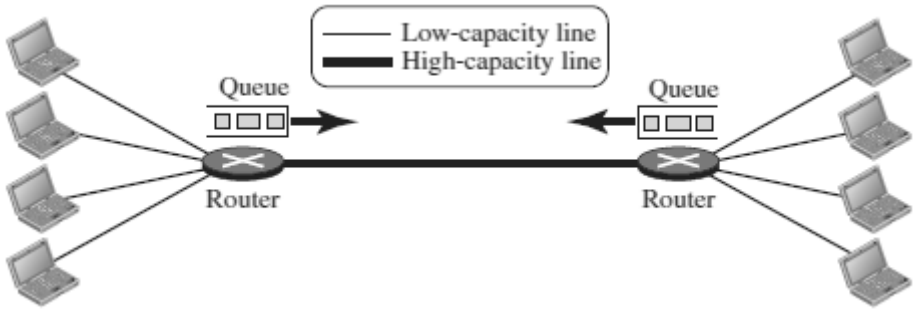


Figura 1.17. Una red de conmutación de paquetes.

Los rúters de una red de conmutación de paquetes tienen una cola (una memoria) en la que pueden almacenar paquetes para reenviarlos posteriormente. Supongamos que la capacidad de la línea gruesa es solo el doble de la capacidad de la línea de datos que conecta cada ordenador con su rúter. Si solo hay dos ordenadores que desean comunicarse (uno en cada extremo), no hay retardos por esperas en cola de los paquetes. Pero si los paquetes llegan a un rúter cuando la línea gruesa ya está trabajando a su capacidad máxima, los paquetes deben almacenarse y reenviarse en el orden en el que llegaron. Estos dos ejemplos sencillos muestran que una red de conmutación de paquetes es más eficiente que una red de conmutación de circuitos, aunque los paquetes pueden experimentar algunos retardos, llegar desordenados, o incuso perderse.

INTERNET.

Como hemos discutido antes, una interred o internet (con *i* minúscula) consiste en dos o más redes que pueden comunicarse unas con otras. La internet más popular es la llamada **Internet** (con *I* mayúscula), que está compuesta por miles (si no millones) de redes LAN y WAN unidas mediante dispositivos de conexión.

La figura 1.18 muestra un diagrama conceptual de Internet, donde vemos que está formada por varias redes troncales (*backbones*), redes de proveedores, y redes de clientes. En el nivel más alto están las **redes troncales**, que son grandes redes propiedad de compañías de telecomunicaciones globales como Verizon, AT&T, British Telecom, Telefónica, etc. Las redes troncales están interconectadas mediante complejos sistemas de conmutadores, denominados puntos de interconexión (*peering points*). En el segundo nivel hay unas redes más pequeñas, llamadas **redes de proveedores**, que usan los servicios de las redes troncales pagando una cuota. Las redes de los proveedores se conectan a las redes troncales, y en ocasiones, a otras redes de proveedores. Estas redes pertenecen a empresas proveedoras de servicios de Internet, como Yoigo, Jazztel, etc. que prestan sus servicios a las redes de los clientes finales. Por último, las **redes de clientes** son las redes de los usuarios, que están situadas en los bordes más exteriores de Internet, y que son las que realmente usan los servicios de Internet. Estas redes de clientes pagan una tarifa a las redes de proveedores para poder acceder a esos servicios.

A las redes troncales y las redes de proveedores también se las conoce como **Proveedores de Servicios de Internet** (**ISPs** = Internet Service Providers). Las redes troncales se suelen denominar **ISPs internacionales**, y a las redes de proveedores se las llama **ISPs nacionales o regionales** (ver figura 1.15).

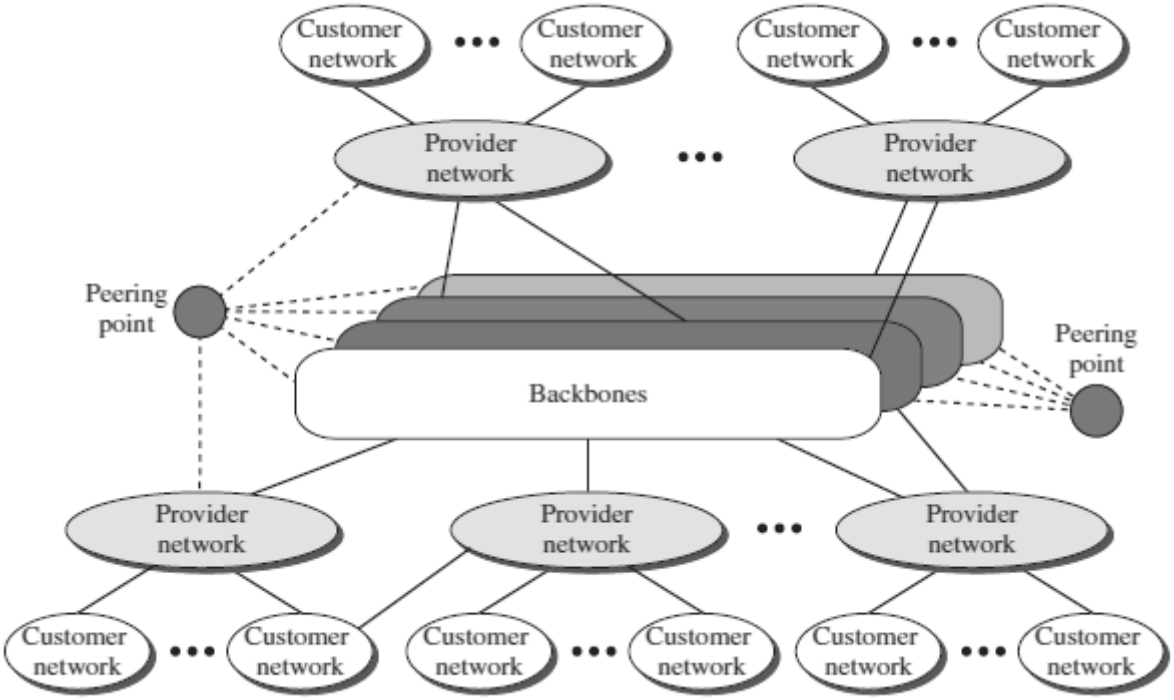


Figura 1.18. Estructura de Internet.

TECNOLOGÍAS DE ACCESO A INTERNET.

Para poder usar los servicios de Internet, los usuarios necesitan estar físicamente conectados a un ISP que les proporcione acceso a la red. Esta conexión física suele hacerse a través de una WAN punto a punto (red telefónica, de cable, de fibra, o inalámbricas). En esta sección vamos a describir brevemente las distintas opciones de conexión a Internet.

Conexión mediante líneas telefónicas.

Hoy día la mayoría de hogares y de pequeñas empresas disponen de servicio de telefonía fija, lo que significa que están conectados a una red telefónica pública. Como la mayoría de redes telefónicas ya están conectadas a Internet, los hogares y las pequeñas empresas pueden conectarse a Internet usando la línea de voz que va desde el hogar o la empresa hasta la central telefónica más próxima como una WAN punto a punto. Esto puede hacerse de dos formas:

- **Servicio de marcación** (*dial-up*). La primera opción es usar la línea telefónica convencional para transmitir los datos digitales en lugar de la voz. Para ello debemos usar un **módem** que convierta los datos digitales en una señal analógica. El software instalado en el ordenador del usuario marca el número del ISP y simula hacer una conexión telefónica ordinaria.

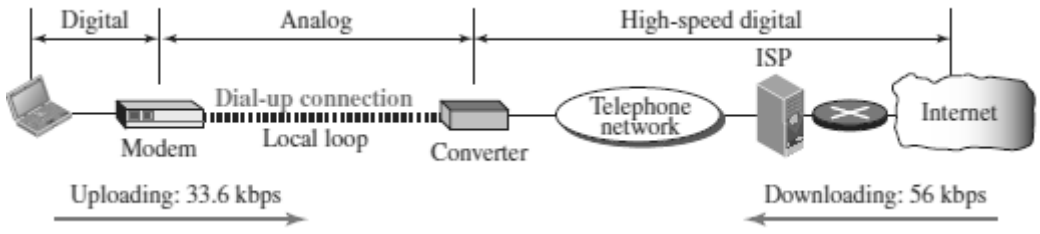


Figura 1.19. Acceso a Internet mediante el servicio de marcación.

Desafortunadamente, el servicio de marcación ofrece unas velocidades de transmisión muy bajas, y cuando la línea ya se está usando para transmitir datos, no puede usarse para transmitir voz al mismo tiempo.

- **Servicio DSL**. Desde que se popularizó el uso de Internet, algunas compañías telefónicas han actualizado sus líneas para poder proporcionar a los usuarios residenciales y a las pequeñas empresas servicios de acceso a Internet mucho más rápidos que los proporcionados por el antiguo sistema de marcación. La tecnología **DSL** (Digital Subscriber Line) permite que la línea pueda cursar simultáneamente servicios de voz y de datos a mayores velocidades.

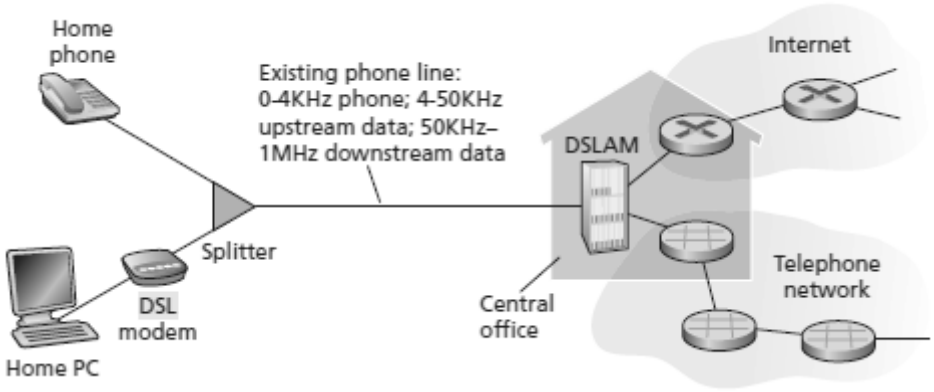


Figura 1.20. Acceso a Internet a través de una línea ADSL.

Conexión mediante redes de cable.

En las últimas décadas muchos usuarios han empezado a usar los servicios de televisión por cable en lugar del tradicional servicio de difusión de televisión a través del aire. Las compañías de cable han actualizado sus **redes de cable** para que los usuarios residenciales y las pequeñas empresas puedan conectarse a Internet a través de los canales de televisión que no se usan. Estas redes proporcionan mayores velocidades de conexión, aunque dependientes del número de usuarios conectados al mismo cable.

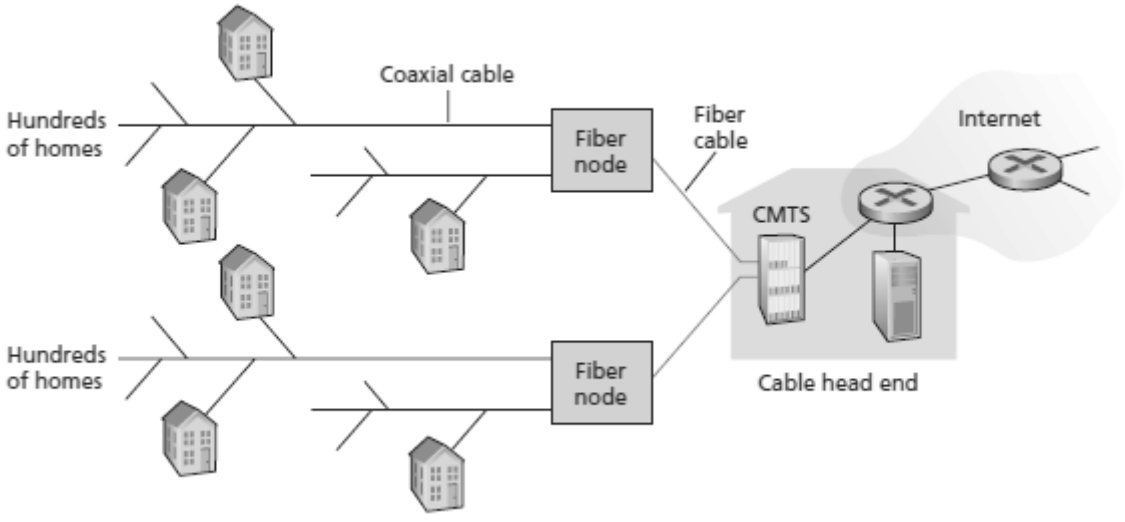


Figura 1.21. Acceso a Internet a través de una red de cable.

Conexión mediante redes de fibra.

Las redes **FTTH** (Fiber To The Home) proporcionan un enlace de fibra óptica desde la central telefónica a la casa del abonado.

Las redes de fibra pueden proporcinar servicios de acceso a Internet a velocidades del orden de gigabits por segundo.

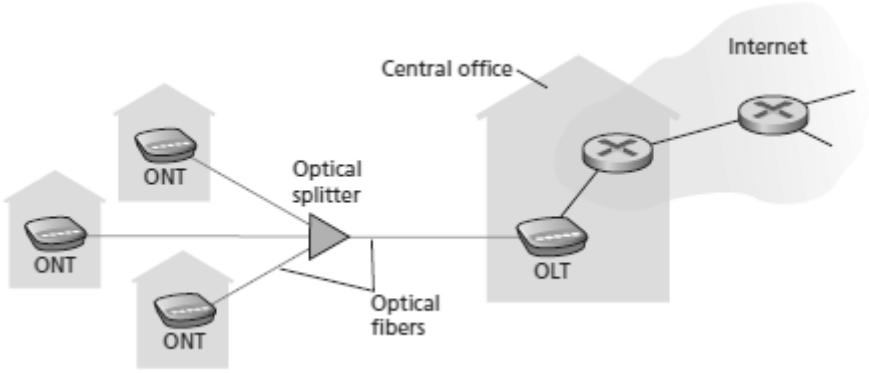


Figura 1.22. Acceso a Internet a través de una red de fibra FTTH.

Conexión mediante redes inalámbricas.

La conectividad inalámbrica se ha vuelto muy popular en los últimos años. Los hogares y las pequeñas empresas pueden contratar los servicios de redes WAN inalámbricas públicas, como las **redes celulares**, las **redes vía satélite**, o las **redes WiMAX**, para acceder a Internet.

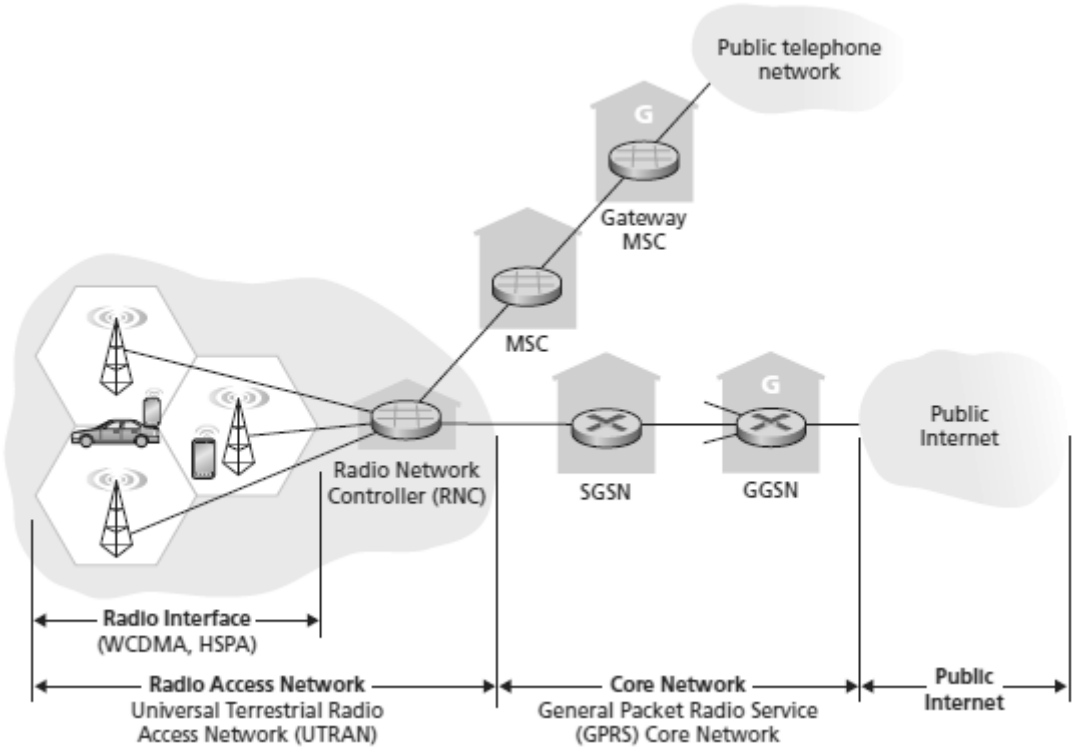


Figura 1.23. Acceso a Internet a través de una red celular.

Conexión mediante líneas alquiladas.

Las grandes empresas y las corporaciones multinacionales pueden convertirse ellas mismas en ISPs locales y conectarse directamente a Internet. Para poder hacerlo, la organización debe alquilar una **conexión dedicada** de alta velocidad a un proveedor de servicios de transporte para conectarse a un ISP regional o nacional. Esa línea dicada es exclusiva, y solo transmite los datos de la organización que la alquila. Por ejemplo, una universidad con varios campus podría crear su propia interred, y alquilar una línea dedicada para conectarla a Internet.

PRÁCTICA A: DIAGRAMAS DE RED.

A.1. DIAGRAMAS DE RED.

Para entender la estructura de una red es necesario documentarla. La **documentación de una red** es toda aquella información que nos ayuda a describirla y a explicar la forma en la que los ordenadores de la red están interconectados (tanto física como lógicamente). Por ejemplo, la conexión física podría ser mediante cables formando una topología en estrella, y la conexión lógica las distintas direcciones que usan los dispositivos de la red.

Microsoft Visio es una herramienta muy común para generar la documentación de una red, pero hay otras herramientas gratuitas que también permiten dibujar diagramas de redes. En estas prácticas vamos a utilizar el editor online gratuito **draw.io** (https://www.draw.io/).

A.2. PRÁCTICA 1: RED LAN BÁSICA.

La figura A.1 muestra un ejemplo de una red LAN básica. Como vemos, el elemento central de esta red es un **concentrador** (**hub**). Un concentrador es el tipo más básico de dispositivo de conexión, y sirve para interconectar mediante cable todos los ordenadores (o hosts) de la red. Cuando un ordenador necesita enviar un paquete de datos, primero se lo envía al hub, el cual lo amplifica y lo difunde al resto de ordenadores de la red. Por supuesto, solo el host de destino conserva los datos recibidos; el resto de hosts los descartan. Aunque esta forma de configurar una red LAN no es la más eficiente, fue el estándar durante bastante tiempo.

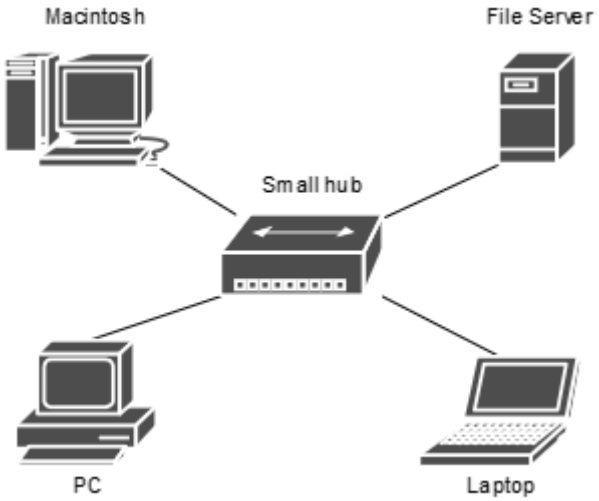


Figura A.1. Red LAN básica.

En la figura vemos varios hosts conectados al concentrador, entre los que tenemos:

- Un servidor, que se usa para almacenar los datos y para compartirlos (o servirlos) al resto de hosts de la red.
- Un PC (ordenador personal) que suele actuar como cliente de la red obteniendo información del servidor. Los PCs también pueden almacenar información localmente en su disco duro.
- Un ordenador Mac (Macintosh), que es otro tipo de ordenador cliente. De nuevo, un Mac también puede almacenar información localmente, u obtenerla de un servidor.
- Un portátil, que podría ser un PC o un Mac. A pesar de tratarse de un ordenador móvil, puede almacenar y acceder a los datos exactamente igual que el resto de ordenadores.

Vamos a dibujar este diagrama usando draw.io. Para ello:

1) Abrimos un navegador web y nos conectamos a https://www.draw.io/. En la ventana emergente, seleccionamos el idioma español, y la opción de guardar los diagramas en el dispositivo. (Ver figura A.2).
2) Como todavía no hemos creado ningún diagrama, en la siguiente ventana seleccionamos la opción "Crear un nuevo diagrama", ver figura A.2.

Figura A.2. Pantallas iniciales de draw.io.

3) En la siguiente ventana, le ponemos nombre al archivo ("Práctica 1. Red básica.drawio"), y elegimos un diagrama en blanco. Al pulsar el botón de "Crear", se abre la interfaz de usuario del programa (figura A.3).

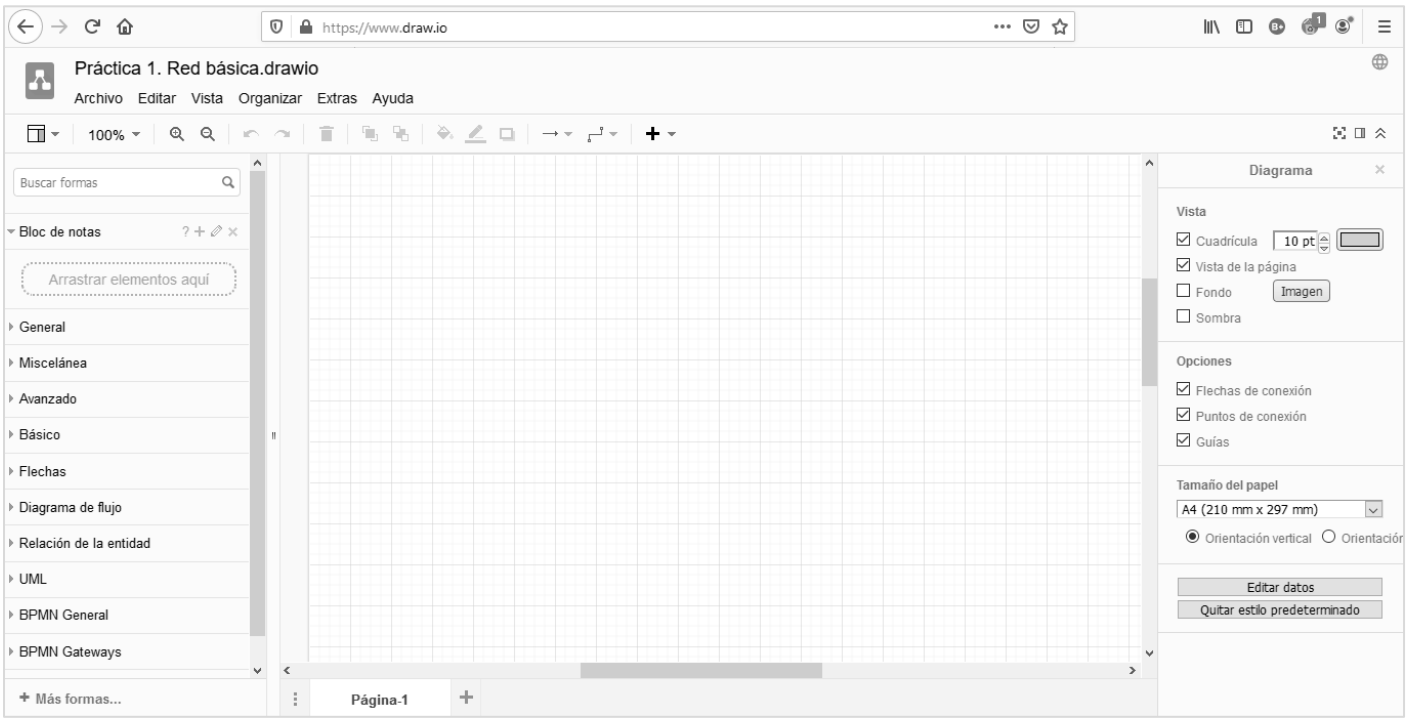


Figura A.3. Interfaz de usuario de draw.io

4) Pulsamos en el botón "Más formas" abajo y a la izquierda para configurar las formas que queremos tener disponibles de forma directa en el editor. Se abrirá una pantalla como la mostrada en la figura A.4. Por el momento, las únicas formas que necesitamos para construir diagramas de red son las de la carpeta "Cisco" (dentro de la categoría "Red"). Nos aseguramos de tener seleccionada únicamente la carpeta

"Cisco", y pulsamos en "Aplicar". Al volver a la interfaz de usuario, y a mano izquierda, veremos que las únicas formas que tenemos disponibles son las de Cisco.[12]

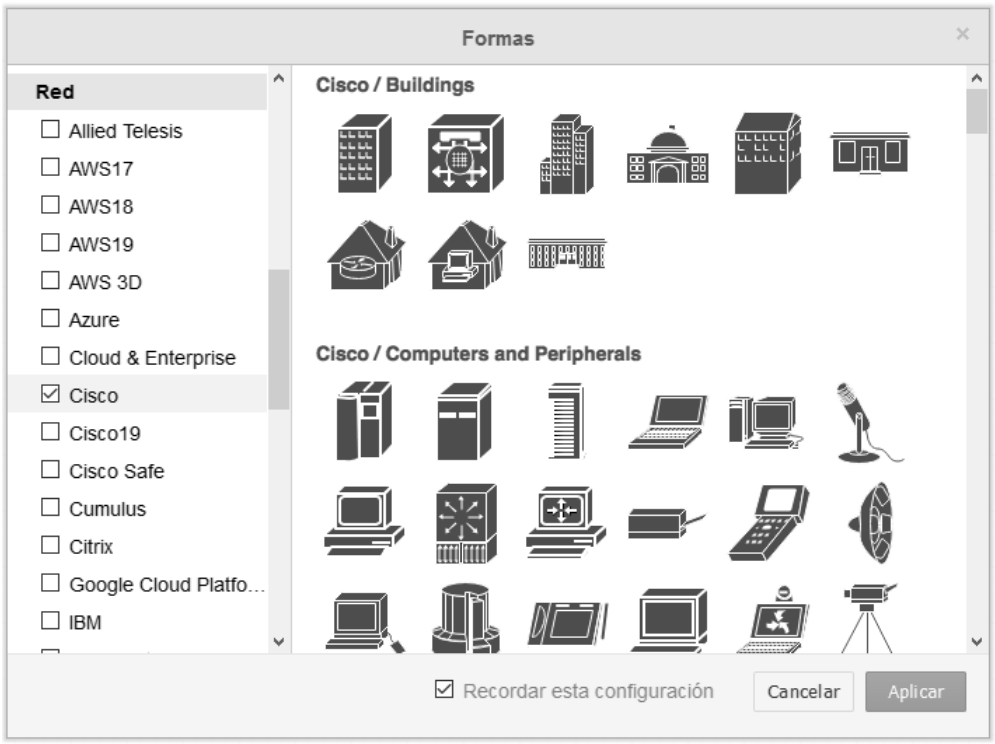


Figura A.4. Selección de las formas visibles en la interfaz de usuario.

5) Comenzamos a diseñar el diagrama de red ubicando en la zona de trabajo todos los dispositivos indicados en la figura A.1, a saber, un PC, un portátil (laptop), un Macintosh, un servidor de archivos (file server), y un concentrador (small hub). El PC, el protátil, y el Macintosh están disponibles en la carpeta de formas "Cisco/Computers and Peripherals"; el servidor de archivos lo encontraremos en la carpeta "Cisco/Servers", y el concentrador se localiza en la carpeta "Cisco/Hubs and Gateways".

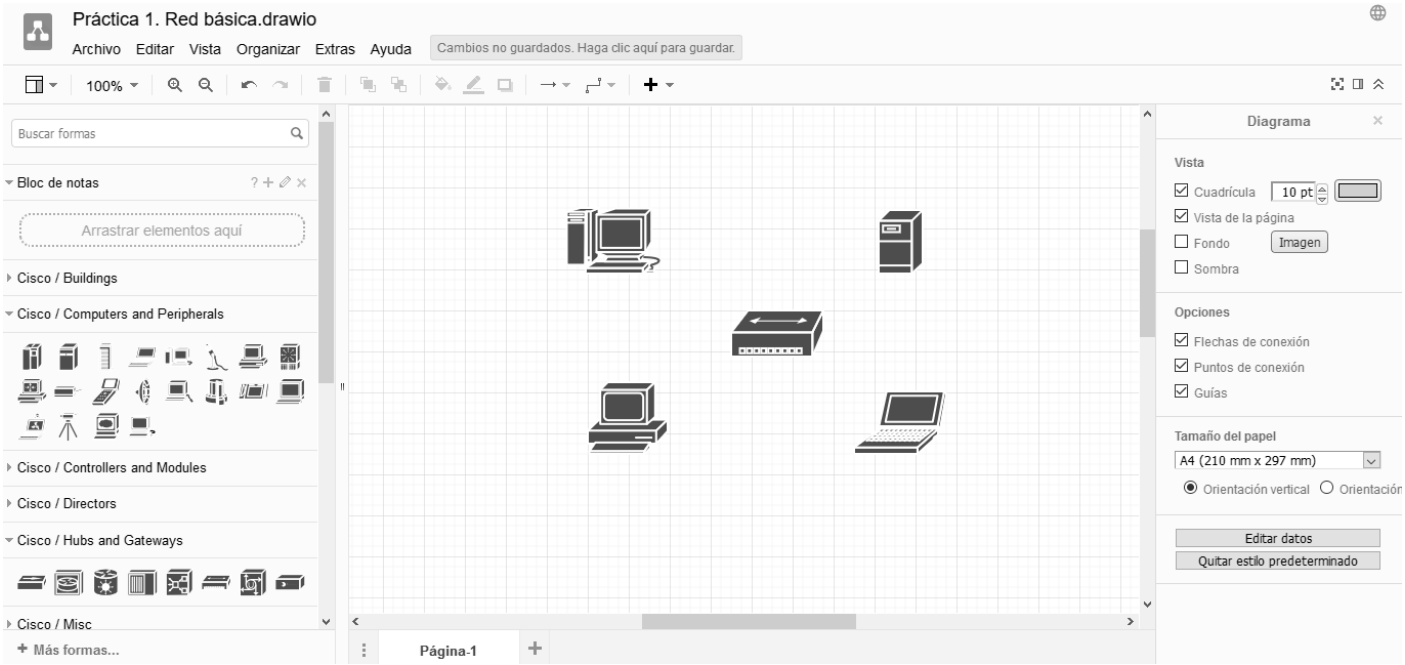


Figura A.5. Dispositivos del diagrama de red básica.

[12] **Cisco** es uno de los mayores fabricantes de equipos de redes del mundo, y los símbolos que utiliza para representar los dispositivos de red se han convertido en una especie de estándar en la construcción de diagramas de redes.

6) Ahora conectamos los distintos hosts (ordenadores y servidor) al concentrador. Al pasar el cursor sobre el PC veremos que aparecen una serie de cruces sobre la forma. Pulsamos con el botón derecho del ratón en una de las cruces, y manteniendo el botón pulsado, arrastramos hasta llegar a una cruz del concentrador. Aparecerá una línea que interconecta ambos dispositivos. (Ver figura A.6).

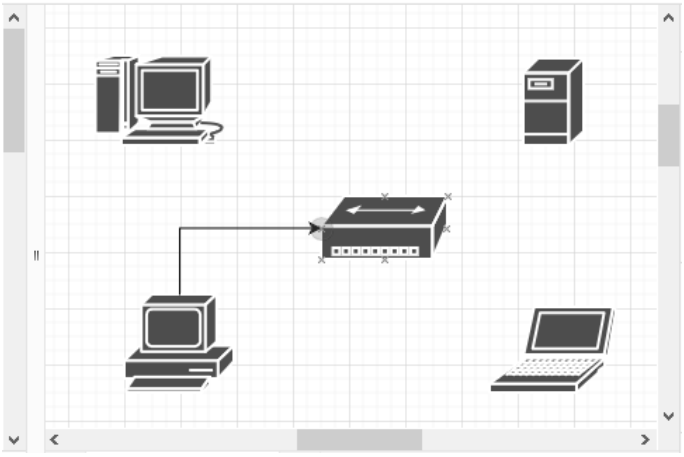


Figura A.6. Interconexión de dispositivos (1).

Vamos a cambiar el formato de la línea de conexión. Pulsamos sobre la línea para que se abra el panel de "Formato". Aquí elegimos que la línea sea diagonal, y que no tenga flecha ni al inicio ni al final de la línea. La conexión debe quedar como ilustra la figura A.7.

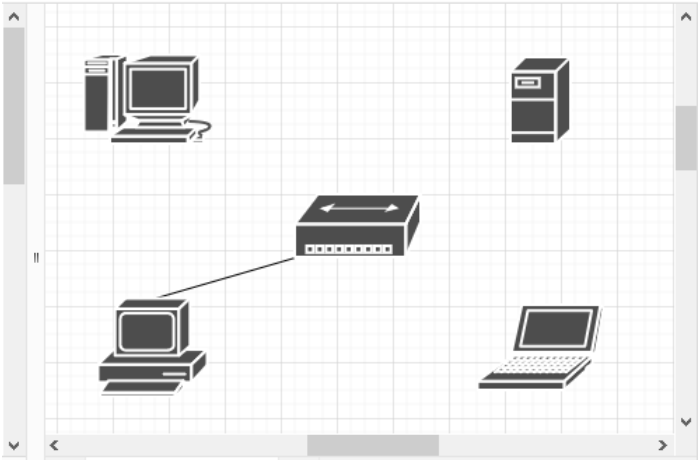


Figura A.7. Interconexión de dispositivos (2).

Repetimos el proceso para todos los hosts de la red, hasta que el diagrama quede como muestra la figura A.8.

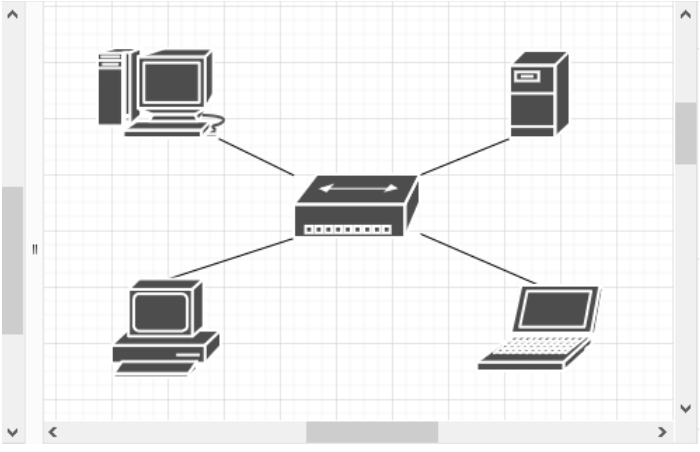


Figura A.8. Interconexión de dispositivos (3).

7) Finalmente, añadimos unos textos para mostrar el tipo de dispositivo representado. Para añadir un texto, pulsamos en el botón del símbolo + (Insertar), y seleccionamos la opción "Texto". Aparecerá una caja de texto en el área de trabajo que podemos mover a la posición deseada, y cuyo contenido podemos editar. Etiquetamos cada dispositivo con su nombre, para que el diagrama quede como el de la figura A.1. Con esto ya hemos terminado.
8) Guardamos el diagrama: Seleccionamos la opción "Archivo" → "Guardar". En la ventana emergente, elegimos la opción "Guardar archivo", y especificamos tanto el nombre del archivo a guardar ("Práctica 1. Red básica.drawio") como la carpeta donde queremos guardarlo (nuestra carpeta de trabajo).

A.3. PRÁCTICA 2: RED LAN INTERMEDIA.

La figura A.9 muestra una LAN algo más compleja, en la que el dispositivo de interconexión central es ahora un **conmutador** (**switch**). Un conmutador es un dispositivo más complejo que un concentrador, porque examina las direcciones físicas de destino de los paquetes que recibe para reenviarlos únicamente al host de destino correcto dentro de la red. El conmutador se conecta a un **rúter**, que es un dispositivo de conexión capaz de analizar las direcciones lógicas de destino de los paquetes recibidos para encaminarlos a la red de destino correcta. En este caso, el rúter permite la interconexión de la red LAN a Internet. La conexión entre el rúter e Internet marca el límite de la red LAN. En otras palabras, la red está formada por el PC, el portátil, el servidor, el conmutador, y el rúter. Todo lo que queda más allá del rúter se considera que está fuera de la red LAN.

Abre draw.io y dibuja el diagrama de la figura. El conmutador (switch) está en la carpeta de formas "Cisco/Switches", el rúter está en la carpeta "Cisco/Routers", y la nube que representa Internet la encontramos en la carpeta "Cisco/Storage". Para cambiar el color de línea y de relleno de la nube, la seleccionamos y acudimos a la etiqueta de "Estilo" a la derecha de la interfaz de usuario de draw.io. Guarda el archivo con el nombre "Práctica 2. Red intermedia.drawio".

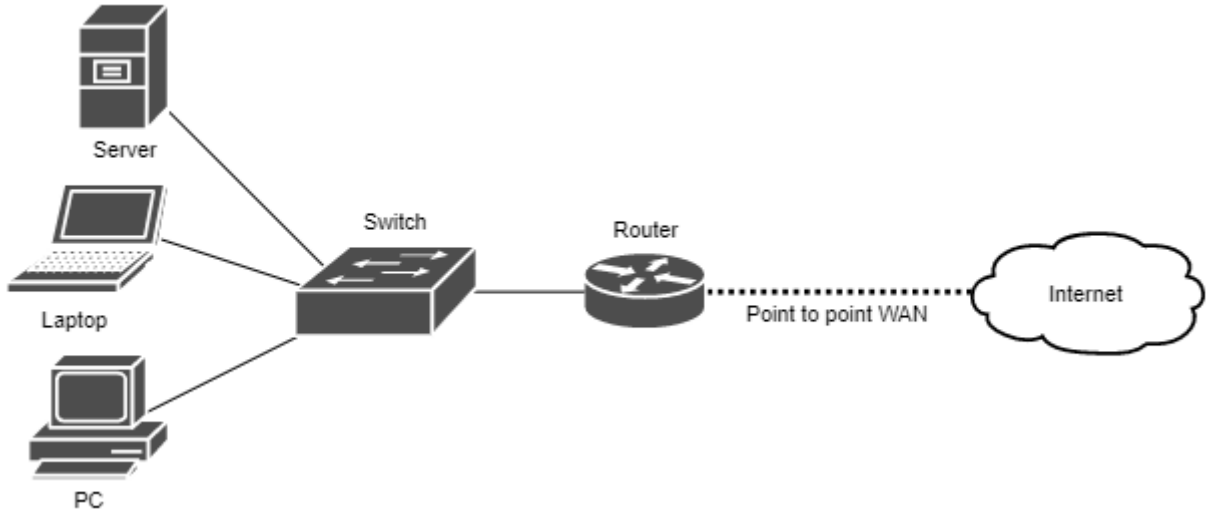


Figura A.9. Red LAN intermedia.

A.4. PRÁCTICA 3: INTERRED.

La figura A.10 muestra un nuevo diagrama de red, esta vez con tres LANs interconectadas mediante un rúter. En esta figura vemos unos cuantos dispositivos nuevos, como un **firewall** que protege a las LANs de Internet, y un **superordenador** que ocupa su propia LAN.

Dibuja este diagrama en draw.io. El firewall se encuentra en la carpeta de formas "Cisco/Security", y el superordenador en la carpeta "Cisco/Computers and Peripherals". Guarda el archivo como "Práctica 3. Interred.drawio".

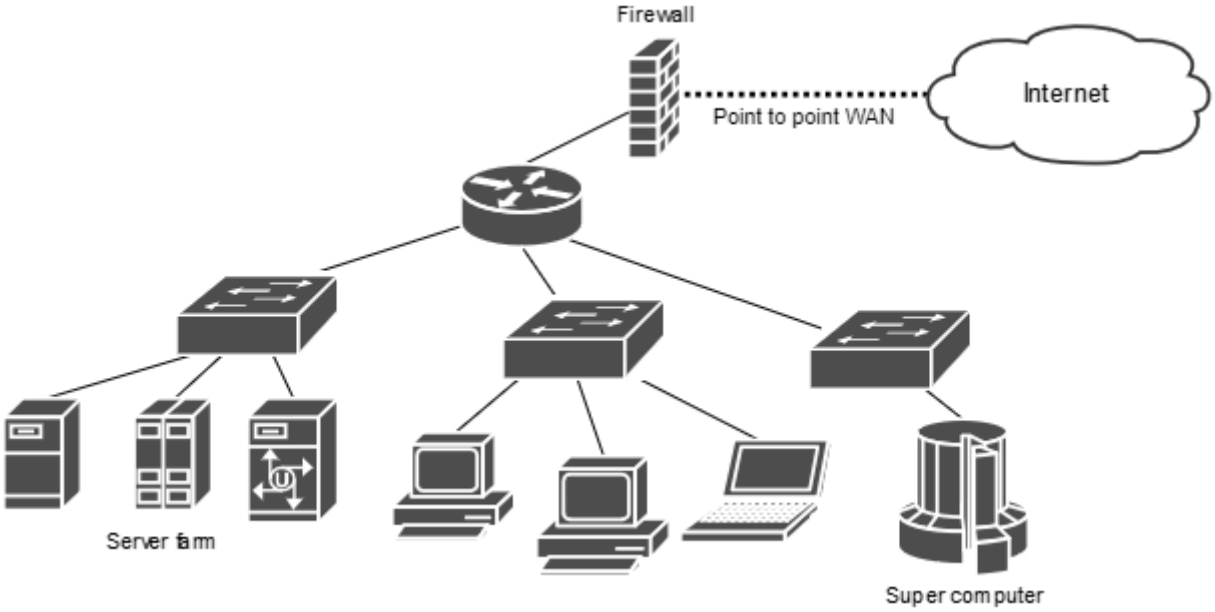


Figura A.11. Interred.

A.5. PRÁCTICA 4: RED LAN INALÁMBRICA.

La figura A.12 muestra una LAN donde algunos dispositivos están conectados mediante cable (Ethernet), y otros dispositivos están conectados inalámbricamente (WiFi). El elemento central es un rúter inalámbrico, que permite la interconexión de los distintos dispositivos de la red, y la conexión de la red a Internet (a través de un módem). Dibuja este diagrama en draw.io y guarda el archivo como "Práctica 4. LAN inalámbrica.drawio".

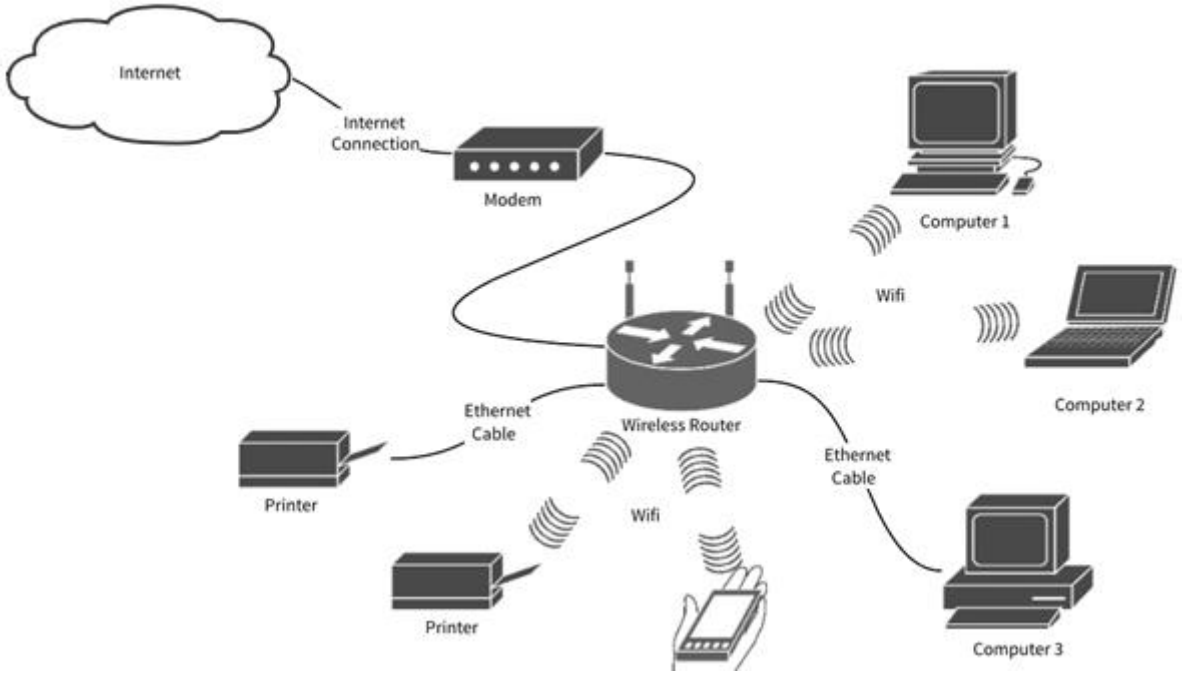


Figura A.12. Red LAN inalámbrica.

PRÁCTICA B: CONCEPTOS SOBRE REDES DE ÁREA LOCAL (LANs).

Una **red de área local** (**LAN**) es una red privada cuya extensión se limita a una zona geográfica reducida, típicamente un hogar, un aula, un edificio, o un campus.

La puesta en marcha de una LAN requiere de ordenadores con adaptadores de red (tarjetas de red), un medio de transmisión (cableado o inalámbrico) para el envío físico de los datos, dispositivos de conexión que interconecten esos ordenadores (hubs, switches, rúters, etc.), y un esquema de direccionamiento que permita identificar los distintos ordenadores de la red (por ejemplo, direcciones IP). Una red LAN también puede incluir otros dispositivos como servidores, firewalls, impresoras, superordenadores, etc.

B.1. ADAPTADORES DE RED.

Todo equipo perteneciente a una red necesita un **adaptador de red** o **tarjeta de interfaz de red** (**NIC** = Network Interface Card) que le permita enviar y recibir datos a través de la red a la que se conecta. Este dispositivo puede estar integrado en la placa base del ordenador, o puede ser un dispositivo independiente conectado a una ranura PCI en la placa base o por USB. El adaptador de red puede conectarse a la red a través de un **cable** (redes **Ethernet**) o **inalámbricamente** (redes **WiFi**).

A la izquierda de la figura B.1 vemos una típica tarjeta de red Ethernet. La zona central de la figura muestra un detalle del puerto de conexión a la tarjeta: Se trata de un **puerto RJ-45** al se conecta la clavija RJ-45 del cable de red correspondiente. El RJ-45 es el puerto de tarjeta de red más habitual, y permite la conexión a la mayoría de las actuales redes cableadas. A modo de ejemplo, podemos tratar de localizar el puerto RJ-45 del adaptador de red de nuestro ordenador (solo en el caso de que nuestro ordenador se conecte a la red vía cabe; en caso contrario, la tarjeta de red será inalámbrica y no dispondrá de puerto RJ-45). A continuación, desconectamos el cable de red del puerto y observamos el conector. Finalmente, tratamos de identificar el dispositivo al que está conectado el otro extremo del cable (hub, switch, o rúter).

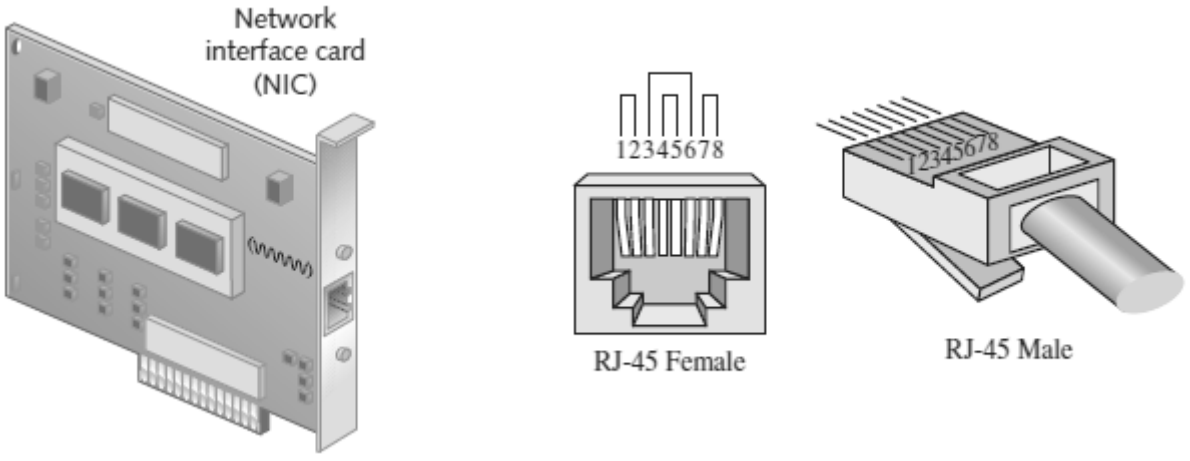


Figura B.1. Un adaptador de red (izquierda), su puerto RJ-45 (centro), y un cable de red (derecha).

Para consultar las propiedades del adaptador de red, debemos acceder al Sistema Operativo (SO) del ordenador. (En esta práctica asumiremos que el SO del ordenador es Windows 10; para otras versiones el proceso debería ser muy parecido):

a) Pulsamos en el botón de "Inicio" en la esquina inferior izquierda del escritorio de Windows, y pulsamos en el botón del engranaje para acceder a la opción de "Configuración" (o simplemente escribimos "Configuración" en la ventana de Inicio).

b) A continuación, seleccionamos la categoría "Red e Internet".
c) Seleccionamos la opción "Cambiar opciones del adaptador".
d) En la ventana de **conexiones de red** (ver figura B.2), seleccionamos el adaptador de red que tengamos activo (Ethernet o Wi-Fi), y hacemos doble clic sobre él. Se abrirá una de las dos ventanas de **estado de la conexión** mostradas en la figura B.3, dependiendo de tipo de conectividad de la que dispongamos y del adapatador de red activo. Esta ventana muestra el tipo de conectividad, la velocidad de conexión, y el tiempo de conexión. También muestra los bytes enviados y recibidos en total.

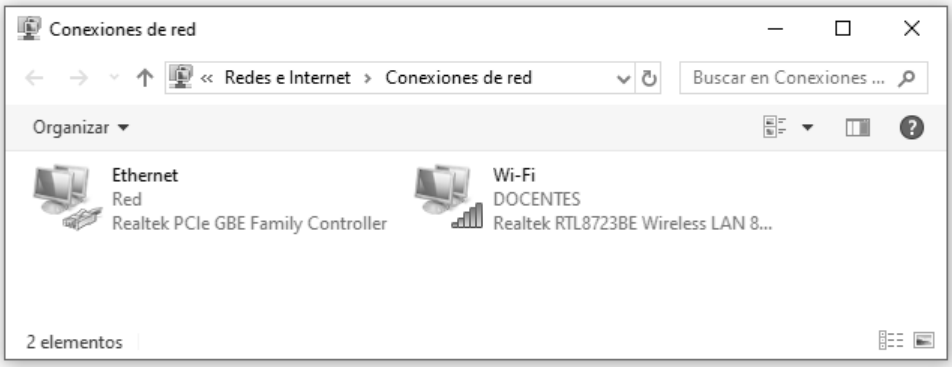


Figura B.2. Ventana de conexiones de red.

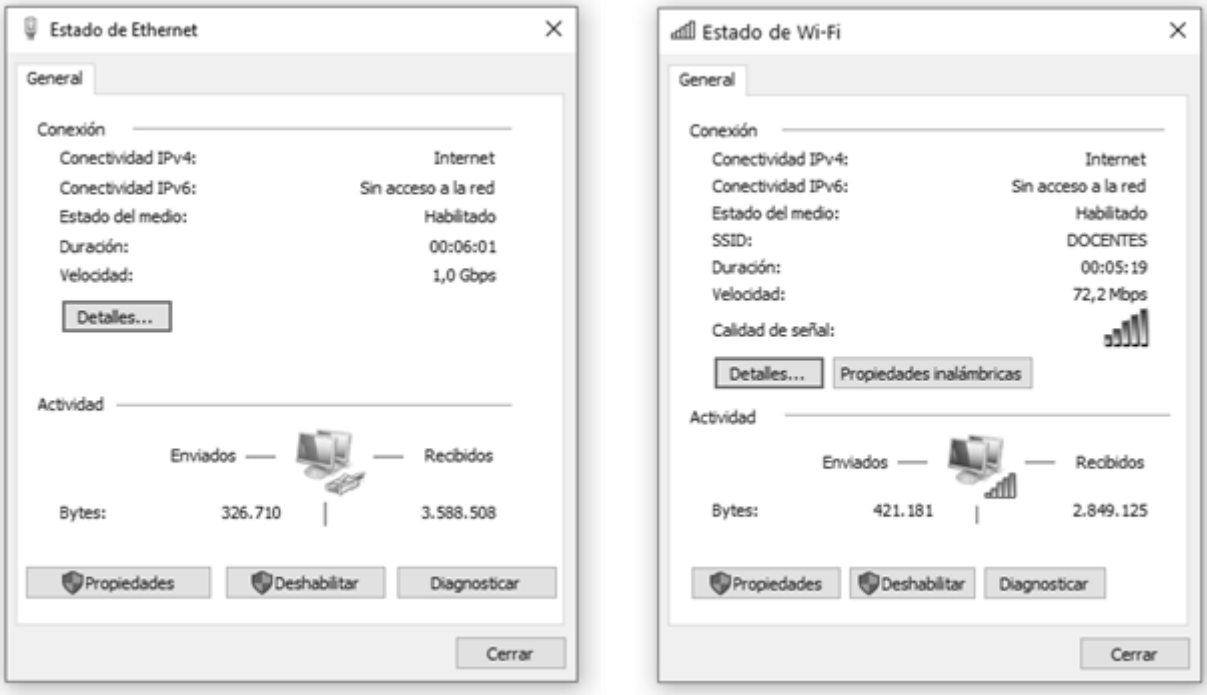


Figura B.3. Las ventanas de estado de la conexión (Ethernet y Wi-Fi).

e) En la ventana de estado de la conexión, pulsamos sobre el botón "Detalles". Se abrirá la ventana de **detalles de la conexión de red** mostrada en la figura B.4. En esta ventana podemos consultar muchos datos útiles sobre el adaptador de red y sobre la conexión de red, como la descripción del adaptador (marca, tipo, etc.), la dirección física y la dirección IP del adaptador, la puerta de enlace predeterminada (esto es, la dirección IP del rúter que conecta con Internet), etc.

Nos apuntamos los valores de la dirección IPv4 y de la puerta de enlace predeterminada, porque los necesitaremos más adelante en esta misma práctica. En mi caso, los valores son:

- Dirección IP del adaptador: 192.168.1.138.
- Puerta de enlace por defecto: 192.168.1.1.

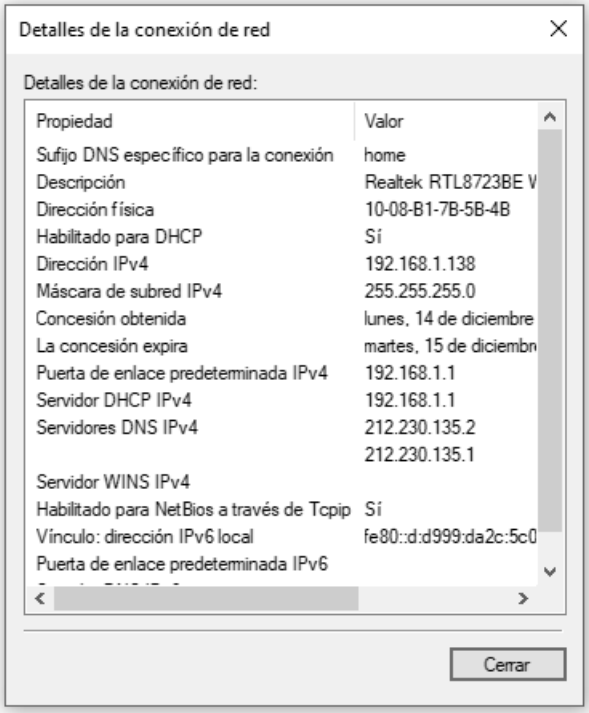


Figura B.4. Detalles de la conexión de red.

f) Cerramos la ventana de detalles de la conexión (figura B.4), y la ventana de estado de la conexión (figura B.3), y volvemos a la ventana de conexiones de red (figura B.2). Seleccionamos el adaptador activo, y pulsamos el botón derecho del ratón sobre él. En la lista desplegable, seleccionamos la opción "Propiedades". En la ventana emergente de **propiedades del adaptador** (ver figura B.5), pulsamos el botón "Configurar". Se abrirá la ventana de **configuración del adaptador de red** correspondiente (Ethernet o Wi-Fi), ver figura B.6.

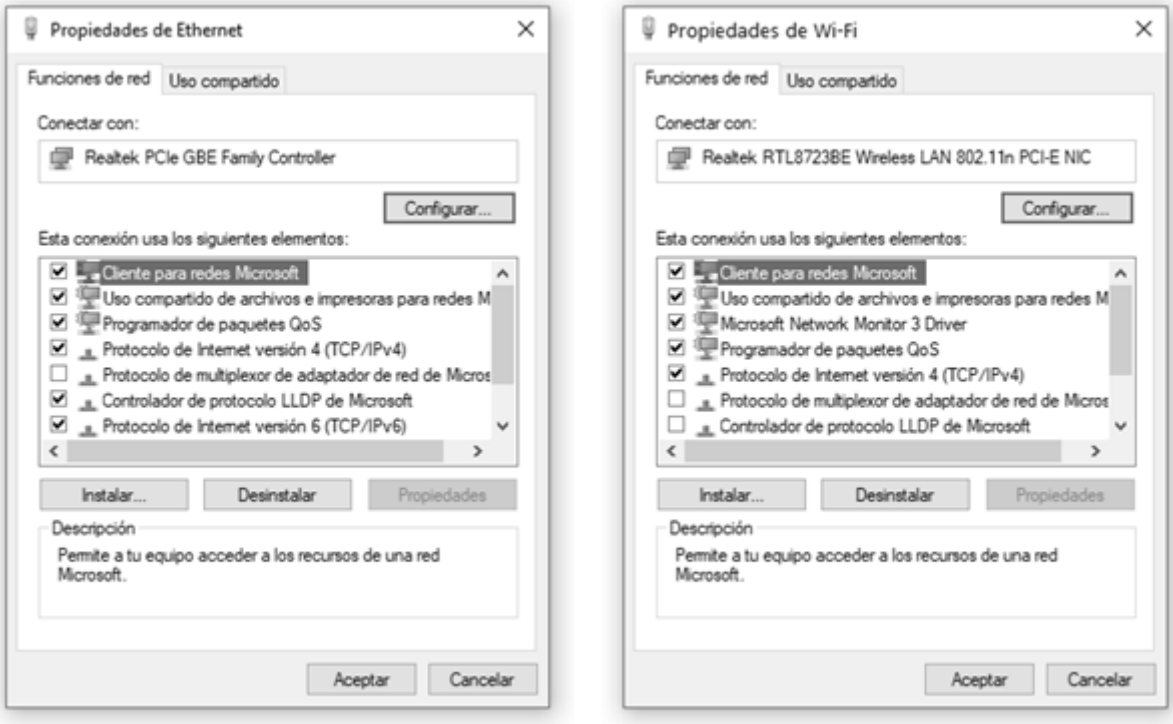


Figura B.5. Propiedades del adaptador de red.

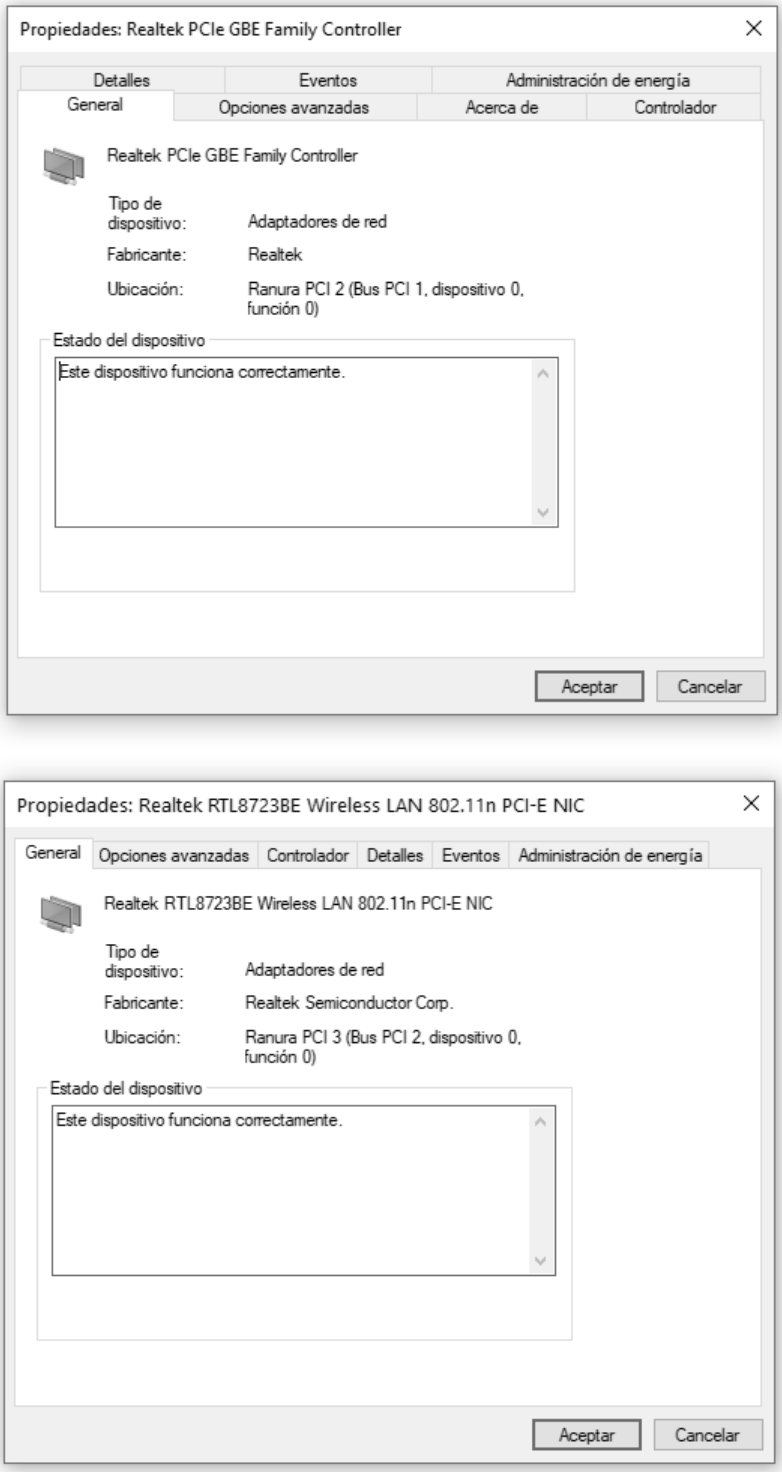


Figura B.6. Ventanas de configuración del adaptador de red.

g) En la ventana de configuración del adaptador pulsamos en la etiqueta "Opciones avanzadas" (ver figura B.7). Dependiendo del tipo de adaptador de red (Ethernet o Wi-Fi), tendremos acceso a distintas opciones avanzadas, que podemos modificar.

En el caso de que estemos conectados a través de cable, nuestro adaptador Ethernet ofrecerá la opción avanzada "Velocidad y dúplex" (ver imagen superior de la figura B.7). Esta opción nos permite elegir entre varias velocidades (en mi caso $10\ Mbps$, $100\ Mbps$, y $1\ Gbps$) y el tipo de dúplex. Recordemos que **full – dúplex** significa que la tarjeta de red puede enviar y recibir datos simultáneamente. Por su parte, **half –dúplex** significa que el adaptador puede enviar y recibir datos, pero no al mismo tiempo. Por supuesto, la opción full – dúplex es superior, y es la que debemos seleccionar siempre que nuestro adaptador la soporte. Una conexión full – dúplex puede enviar y recibir el doble de información por segundo que una conexión half –dúplex.

En el caso de estar conectados inalámbricamente, el adaptador Wi-Fi ofrece algunas opciones avanzadas de las que no vamos a hablar aquí, como la selección del ancho de banda (bandwidth), el modo inalámbrico, etc.

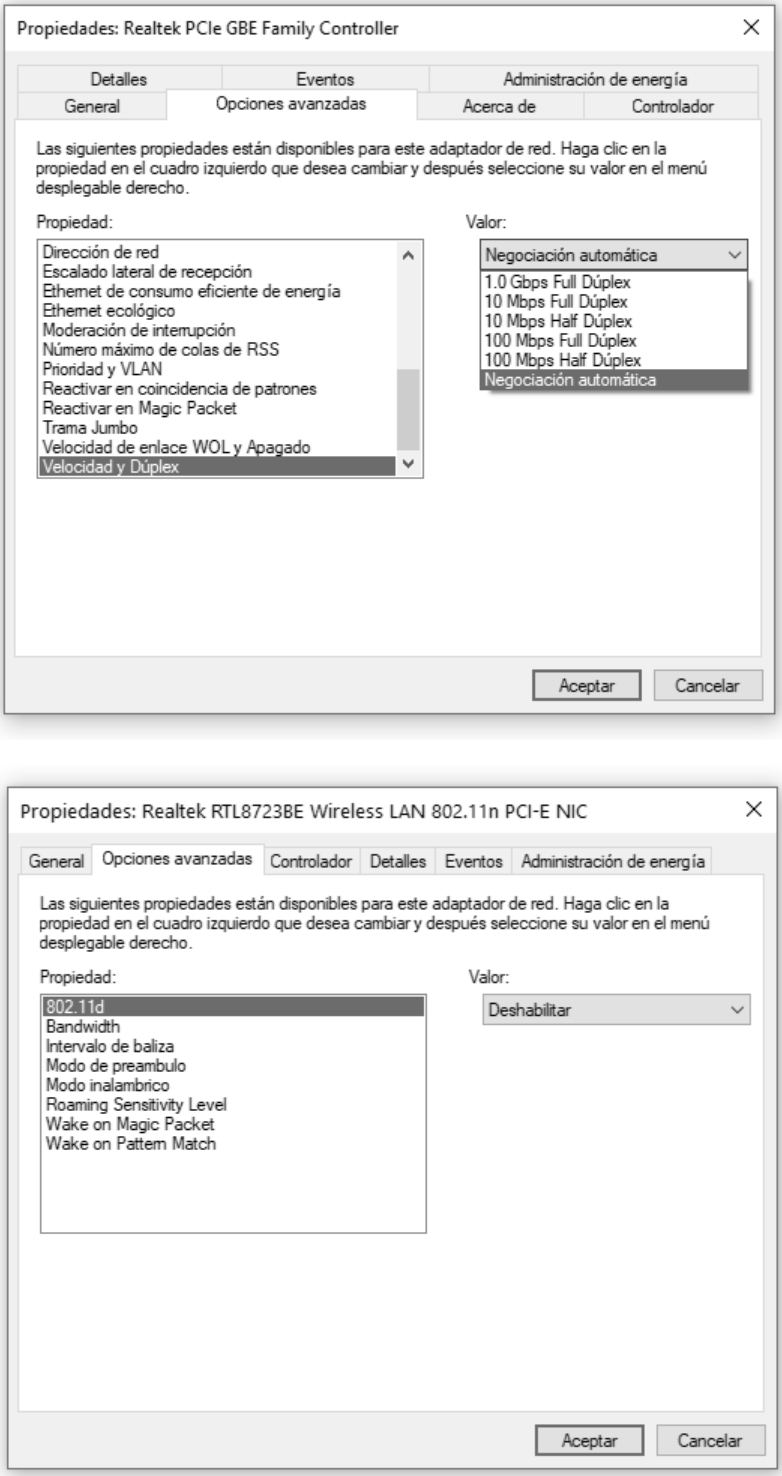


Figura B.7. Opciones avanzadas.

B.2. DIRECCIONAMIENTO.

DIRECCIONES IP.

Las **direcciones IP** constituyen la piedra angular de la interconexión de redes, ya que permiten la comunicación uno a uno entre los distintos hosts conectados a una red o interred. De la misma forma que la dirección de nuestra vivienda indica el número de la casa y la calle en la que se encuentra, una dirección IP identifica a un ordenador y a la red en la que reside. Un ejemplo de dirección IP es 192.168.1.138. Toda

dirección IP está dividida en dos partes: La **porción de red** (en el ejemplo previo, 192.168.1), que identifica a la red de la que forma parte el ordenador; y la **porción de host** (en este caso, 138), que es el número que identifica al ordenador dentro de esa red, y lo distingue del resto de ordenadores conectados a ella. Ahora bien, ¿cómo sabemos qué parte de la dirección IP identifica a la red, y qué parte identifica al host? Eso nos lo dice la máscara de subred.

La **máscara de subred** es un grupo de cuatro números que define la red IP de la que un host es miembro. Todos los 255's de una máscara de subred identifican colectivamente la porción de red de una dirección IP, mientras que los 0's se refieren a la porción de host. Por ejemplo, la máscara de subred 255.255.255.0 asociada a la dirección IP 192.168.1.138 nos indica que la porción de red es 192.168.1 y que la porción de host es 138.

Las direcciones IP suelen asignarse a los adaptadores de los ordenadores de la red, pero también se pueden asignar a otros dispositivos, como los rúters. A continuación, vamos a ver cómo configurar las direcciones IP de una red LAN en Windows 10 (aunque esta configuración será similar en otras versiones de Windows).

CONFIGURAR LAS DIRECCIONES IP.

Para configurar las direcciones IP de un ordenador que queremos conectar a una LAN debemos seguir los siguientes pasos:

1) Accedemos a la ventana de propiedades del adaptador de red activo (figura B.5).
2) Pinchamos en la opción "Protocolo de Internet Versión 4 (TCP/IPv4)" y pulsamos en el botón de "Propiedades". Cuando se abre la ventana de diálogo, nos apuntamos las configuraciones actuales (si las hay) para poder devolver el ordenador a su configuración inicial tras terminar este ejemplo.
3) La opción habilitada por defecto suele ser "Obtener una dirección IP automáticamente" y "Obtener la dirección del servidor DNS automáticamente", ver figura B.8. Esto significa que el adaptador de red intentará obtener automáticamente su configuración IP de un servidor DHCP o de otro dispositivo, como un rúter.

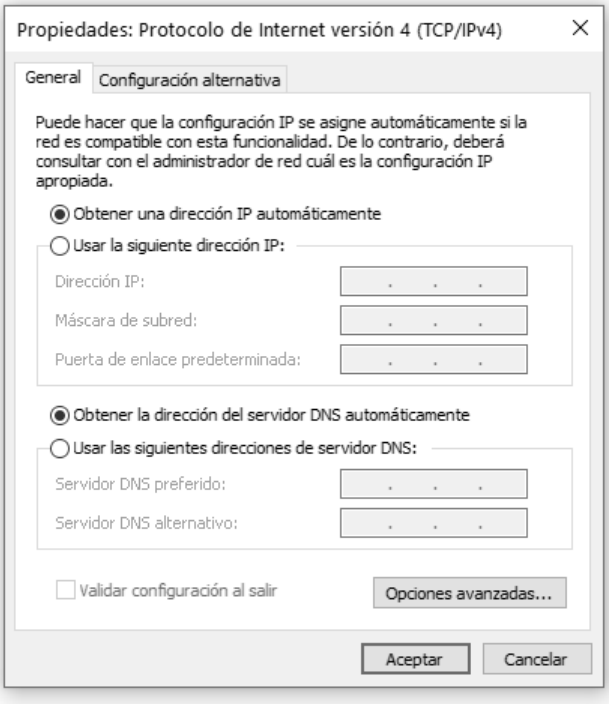


Figura B.8. Configuración IP automática.

Para consultar la dirección IP que se nos ha asignado automáticamente, podemos hacer dos cosas:

a) Consultar los detalles de la conexión de red, algo que ya hicimos en el apartado (e) de la sección previa, y cuyos valores ya nos anotamos entonces. En mi caso eran:
   - Dirección IP del adaptador: 192.168.1.138.
   - Puerta de enlace por defecto: 192.168.1.1.

b) Utilizar la ventana de comandos. Para ello, pulsamos las teclas WINDOWS + R y en la ventana de "Ejecutar" emergente, escribimos `cmd`. Una vez abierta la ventana de comandos, escribimos la instrucción `ipconfig` y pulsamos ENTER. El resultado debería ser similar al mostrado en la figura B.9. Notar que los datos obtenidos deben ser los mismos que obtuvimos consultando los detalles de la conexión.

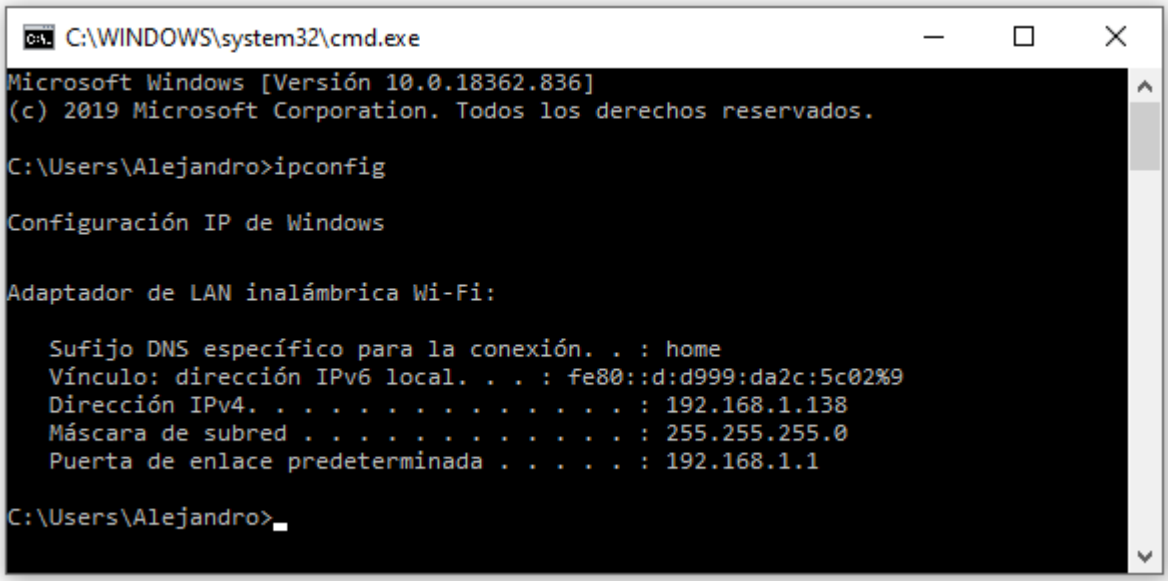


Figura B.9. Consulta de la configuración IP automática mediante la ventana de comandos.

4) Ahora vamos a establecer la configuración IP de forma manual. Para ello volvemos a la ventana de propiedades del protocolo de Internet, activamos la opción "Usar la siguiente dirección IP", y en los campos bajo ella escribimos lo siguiente:

a) Dirección IP: Asignamos un nuevo valor a nuestra dirección IP, dentro del rango de direcciones IP válidos para la máscara de subred actual. Para ello, cambiamos únicamente el identificadrde host de la dirección IP que teníamos asignada automáticamente. Por ejemplo, si mi dirección IP automática es 192.168.1.**138**, cambio su valor a 192.168.1.**101**. Concretamente, no debemos elegir la misma dirección IP que nuestra puerta de enlace predeterminada (que en mi caso es 192.168.1.**1**).

   IMPORTANTE: Si estamos trabajando con otros compañeros conectados a la misma red, cada persona debería poner una dirección IP distinta. Por ejemplo, en el primer ordenador deberíamos fijar la dirección IP 192.168.1.101, en el segundo ordenador 192.168.1.102, y así sucesivamente. Con esto evitaremos cualquier posible conflicto de direcciones IP.

b) Máscara de subred: Escribimos el mismo valor que tenía antes; en mi caso, 255.255.255.0.
c) Puerta de enlace predeterminada: Mantenemos la dirección IP de la puerta de enlace predeterminada de la configuración automática. En mi caso, este valor es 192.168.1.1.
d) Servidor DNS preferido: El servicio DNS permite efectuar la resolución de nombres (es decir, la traducción de nombre de dominio como www.facebook.com a direcciones IP). En nuestro caso, vamos a usar los servidores DNS públicos de Google, cuyas direcciones IP son 8.8.8.8 y 8.8.4.4. Rellenamos el

campo Servidor DNS preferido con la IP 8.8.8.8, y el campo Servidor DNS aternativo con la IP 8.8.4.4.

La figura B.10 muestra como debería quedar la ventana de propiedades del protocolo de Internet tras finalizar nuestra configuración manual.

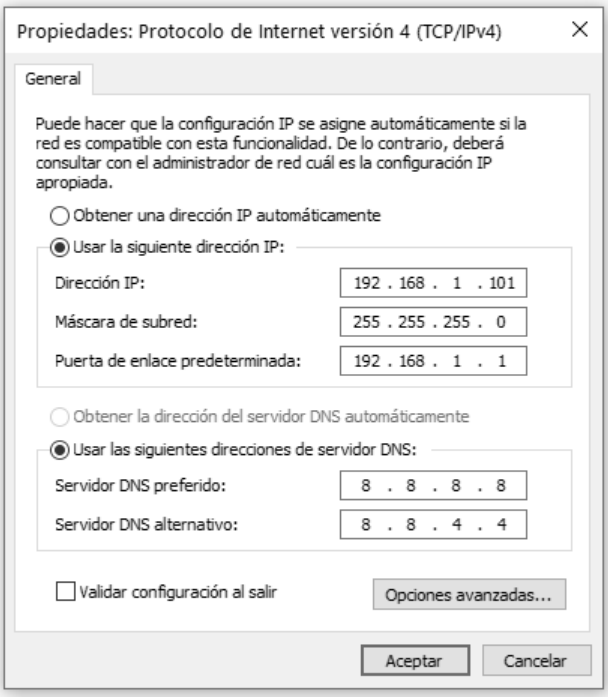


Figura B.10. Configuración IP manual.

5) Una vez configurado todo, pulsamos en "Aceptar". A continuación, en la ventana de propiedades del adaptador, pulsamos "Cerrar".

6) Vamos a probar nuestra nueva configuración manual. Para ello, primero usaremos el comando `ipconfig`, y a continuación, el comando `ping`:

   a) Abrimos la ventana del símbolo de sistema y ejecutamos el comando `ipconfig`. El resultado debería ser similar al mostrado en la figura B.11.

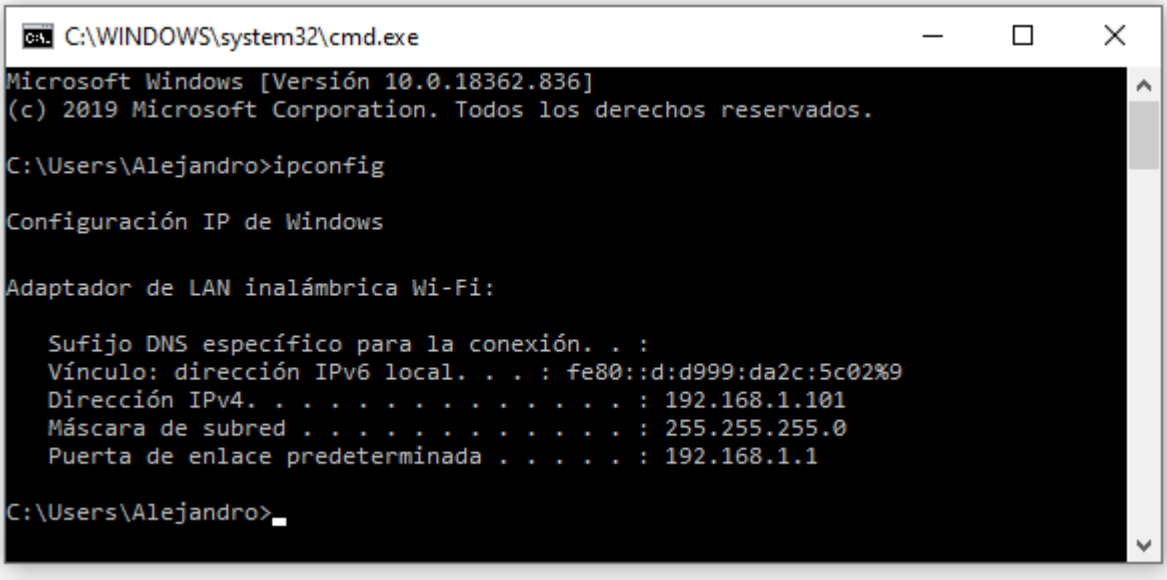


Figura B.11. Resultado del comando `ipconfig` en la ventana de símbolo del sistema.

El valor de la dirección IPv4 debería ser el que configuramos previamente. De no ser así, volvemos atrás y comprobamos la configuración del Protocolo de Internet versión 4.

b) Ahora hacemos un `ping` a otro ordenador de la red (Si no hay otros ordenadores, hacemos un `ping` a la puerta de enlace predeterminada, o a nuestra propia dirección IP). Por ejemplo, suponer que uno de los ordenadores de la red tiene como dirección IP 192.168.1.102. En tal caso, escribimos el siguiente comando:

```
ping 192.168.1.102
```

Este comando envía una serie de cuatro paquetes de datos de prueba a la otra dirección IP. Si el otro ordenador está funcionando y está configurado correctamente, debería responder. Un `ping` positivo debería tener un aspecto similar al mostrado en la figura B.12, donde vemos que se han recibido cuatro respuestas desde el ordenador al que hemos hecho el `ping`.

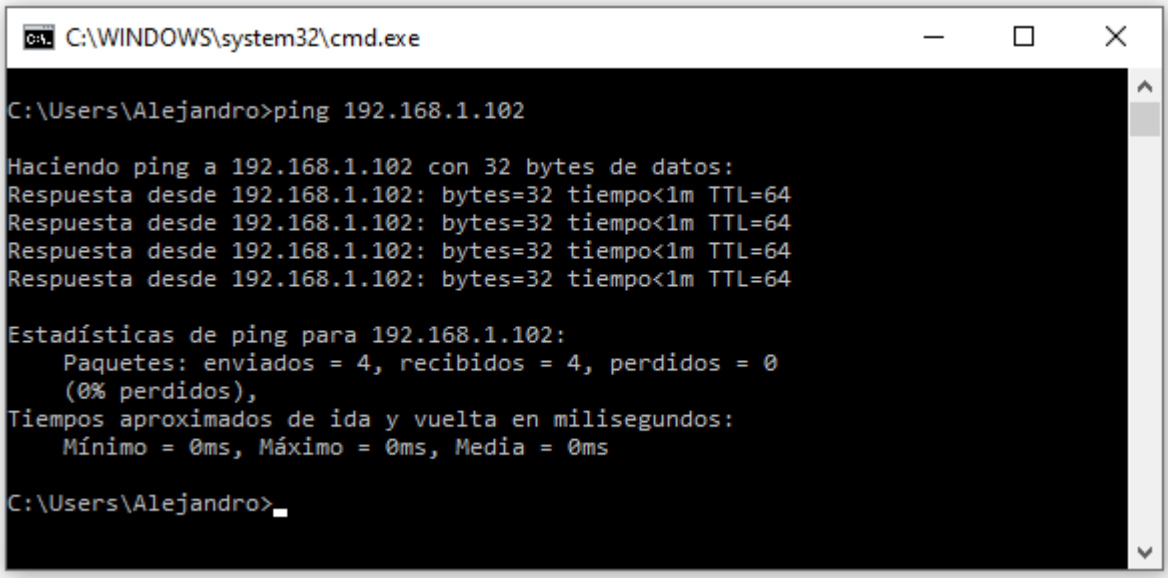


Figura B.12. Resultado del comando `ping` en la ventana de símbolo del sistema.

Si no recibimos respuesta alguna, y en cambio obtenemos mensajes del tipo "tiempo de espera agotado" o "host de destino inaccesible", debemos comprobar nuestra propia configuración IP, y asegurarnos de que el ordenador al que le estamos haciendo el `ping` también está configurado correctamente. Además, debemos comprobar si los ordenadores están físicamente conectados a la red (por ejemplo, en el caso de una red Ethernet, deberíamos comprobar que los ordenadores están conectados mediante un cable a la red).

También podemos probar sendos `ping`'s a la puerta de enlace predeterminada (esto es, al rúter), y a un ordenador externo a la red (por ejemplo, al servidor web de alguna página bien conocida, como Facebook). El ping a mi rúter, cuya dirección IP es 192.168.1.1, sería:

```
ping 192.168.1.1
```

El ping al servidor web de Facebook sería:

```
ping www.facebook.com
```

Notar que, en este caso, hemos hecho un `ping` por nombre de dominio, y no por dirección IP. Veremos que lo primero que hace el comando `ping` es traducir el nombre de dominio www.facebook.com del servidor web de Facebook a su correspondiente dirección IP (que en el momento de escribir esta práctica, era

31.13.83.8). Esta conversión ha sido posible gracias a la ayuda del servidor DNS público de Google, tal y como lo configuramos en la ventana de propiedades del protocolo de Internet.

Al terminar, volvemos a restaurar la configuración IP previa del ordenador en el que estamos trabajando.

PRÁCTICA C. CREAR UNA RED LOCAL CON WINDOWS.

C.1. CONFIGURAR UNA RED DOMÉSTICA CON WINDOWS 10.

En la práctica previa aprendimos a configurar y modificar los parámetros de una red LAN que ya estaba creada. En esta práctica vamos a desplegar nuestra propia red LAN doméstica con Windows.

El sistema operativo Windows es muy práctico a la hora de crear redes domésticas. En esta práctica vamos a configurar una pequeña red local (LAN) con esta plataforma. Esta práctica está pensada para hacerse con Windows 10, aunque el proceso debería ser muy parecido en otras versiones. Los pasos a seguir son los siguientes:

Primer paso. Acudimos a la configuración de Windows. Para ello pulsamos en el botón de inicio (abajo y a la izquierda del escritorio), y en la ventana emergente presionamos en el botón del engranaje en el lado izquierdo. En la ventana de configuración, seleccionamos el apartado "Red e Internet".

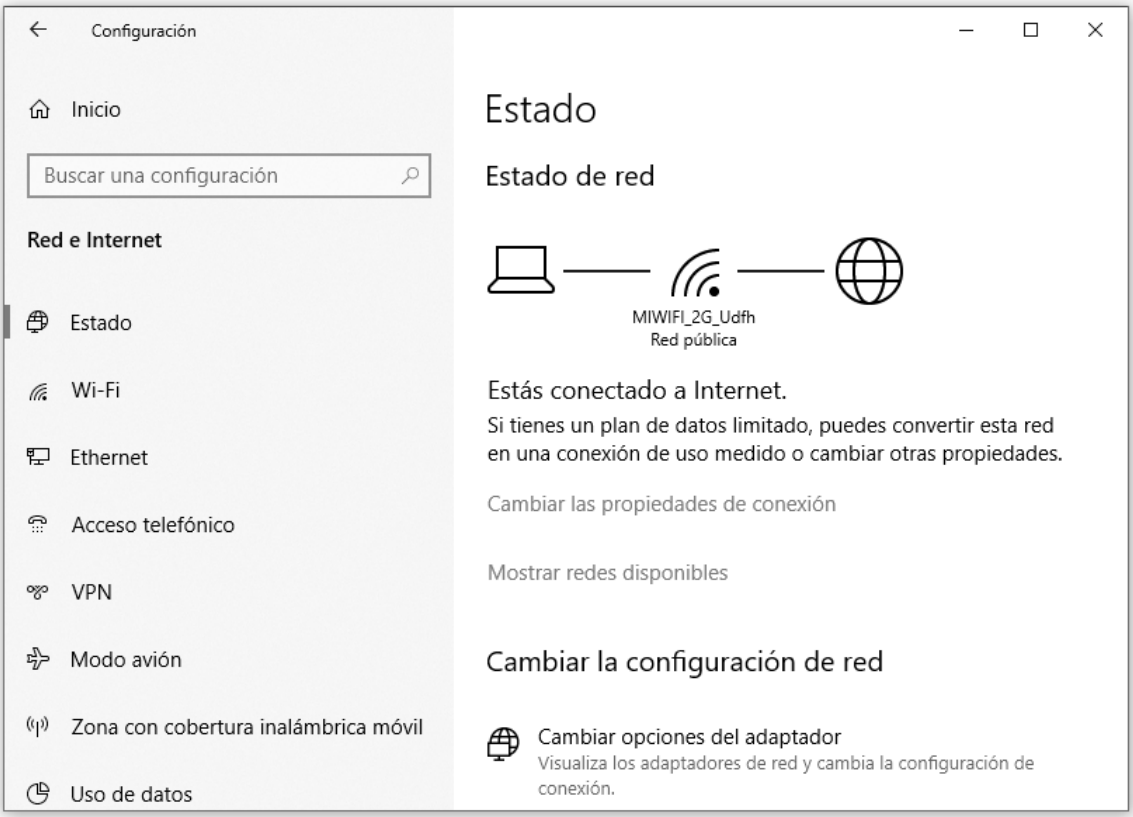


Figura C.1. Configuración de Red e Internet: Estado.

Segundo paso. En la ventana de "Red e Internet" podremos ver una sección de "Estado" de la conexión, independientemente de si nuestra conexión es a través de cable (Ethernet) o inalámbrica (WiFi), ver figura C.1. Lo único importante aquí es mantener nuestro perfil en la modalidad privada. Para ello, pulsamos en la opción "Cambiar las propiedades de la conexión". Una vez dentro, en la opción "Perfil de red", seleccionamos "Privada". De esta forma, los equipos serán visibles en la red y podremos compartir archivos e impresoras fácilmente.

Tercer paso. Volvemos a la sección de "Estado", y bajando por la pantalla, llegaremos al apartado "Opciones de uso compartido", donde podremos cambiar las opciones de uso compartido para los distintos perfiles de

red. Ampliamos el desplegable del perfil "Privado" (que es nuestro perfil actual), y habilitamos la detección de redes y el uso compartido de archivos e impresoras, ver figura C.2.

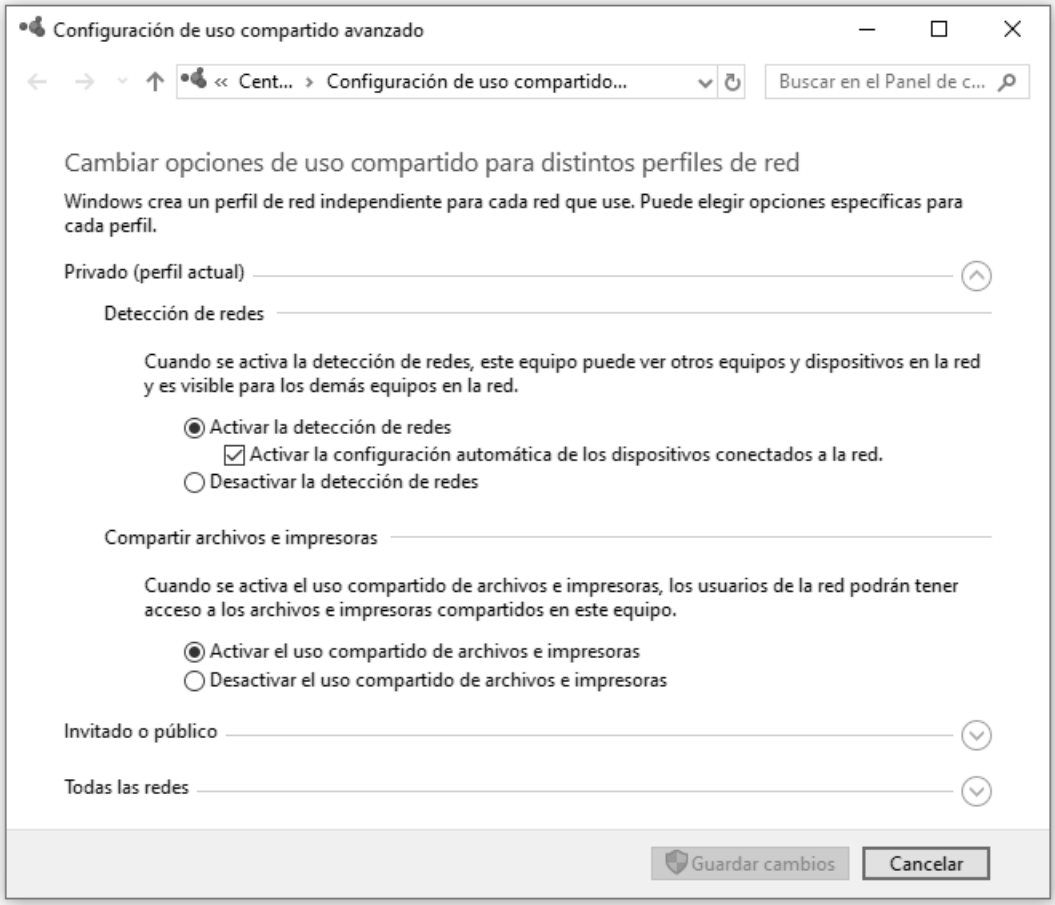


Figura C.2. Opciones de uso compartido para el perfil privado.

Cuarto paso. A continuación, y todavía dentro del apartado "Opciones de uso compartido", acudimos al desplegable "Todas las redes". Los ajustes que tiene activados por defecto deberían bastar para que la red funcione perfectamente sin cambiar nada. En nuestro caso solo debemos pinchar en "Elegir opciones de transmisión por secuencias de multimedia". La primera vez que entremos aquí aparecerá un aviso que nos indica que al activar esta opción estaremos permitiendo que otros ordenadores y dispositivos puedan acceder a los archivos multimedia (música, imágenes, y vídeos) del nuestro. Para continuar, pulsamos en el botón "Activar la transmisión por secuencias de multimedia". Aquí podemos elegir qué dispositivos pueden acceder a nuestro contenido multimedia. Los dispositivos se incluirán en una lista que incluye tanto ordenadores como dispositivos inteligentes (televisores, TV Boxes, etc.), ver figura C.3. Cuando terminemos con la configuración pulsamos en Aceptar.

Quinto paso. De vuelta al desplegable "Todas las redes", nos vamos abajo del todo hasta llegar al menú "*Uso compartido con protección por contraseña*", donde elegimos la opción "Desactivar el uso compartido con protección por contraseña" (ver figura C.4). Esto hará que el resto de ordenadores no tengan que escribir una contraseña para acceder a nuestros recursos compartidos. Aquí es importante tener una contraseña WiFi robusta para evitar intrusiones, ya que todo equipo que se conecte a la red WiFi tendrá acceso a nuestros recursos. (Por supuesto, si queremos que otros ordenadores solo puedan acceder con una contraseña, elegimos la otra opción). Cuando esté todo listo, pulsamos en "Guardar cambios".

Sexto paso. Repetir los pasos previos para todos los ordenadores que queramos conectar a la red.

C.2. COMPARTIR CARPETAS EN RED.

Una vez configurado nuestro ordenador dentro de una red doméstica, toca elegir qué carpetas queremos compartir. Para ello entramos en el *Explorador de Archivos* de Windows, y pinchamos con el botón derecho del ratón sobre la carpeta que deseamos compartir con el resto de equipos de la red.

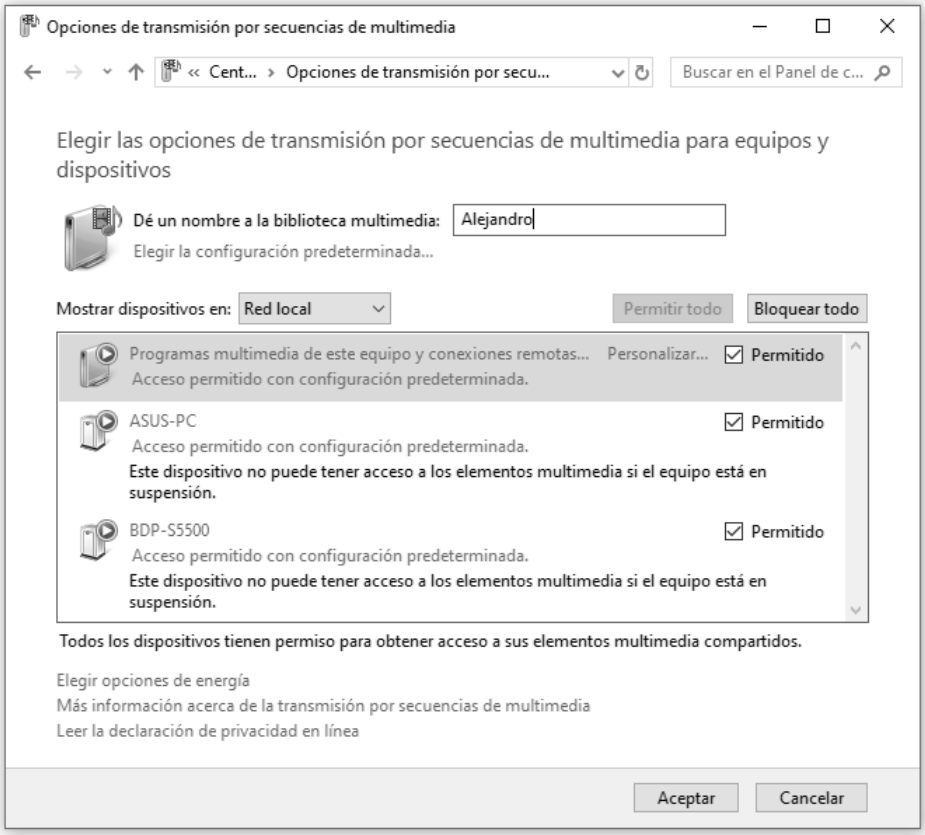


Figura C.3. Opciones de transmisión por secuencias multimedia.

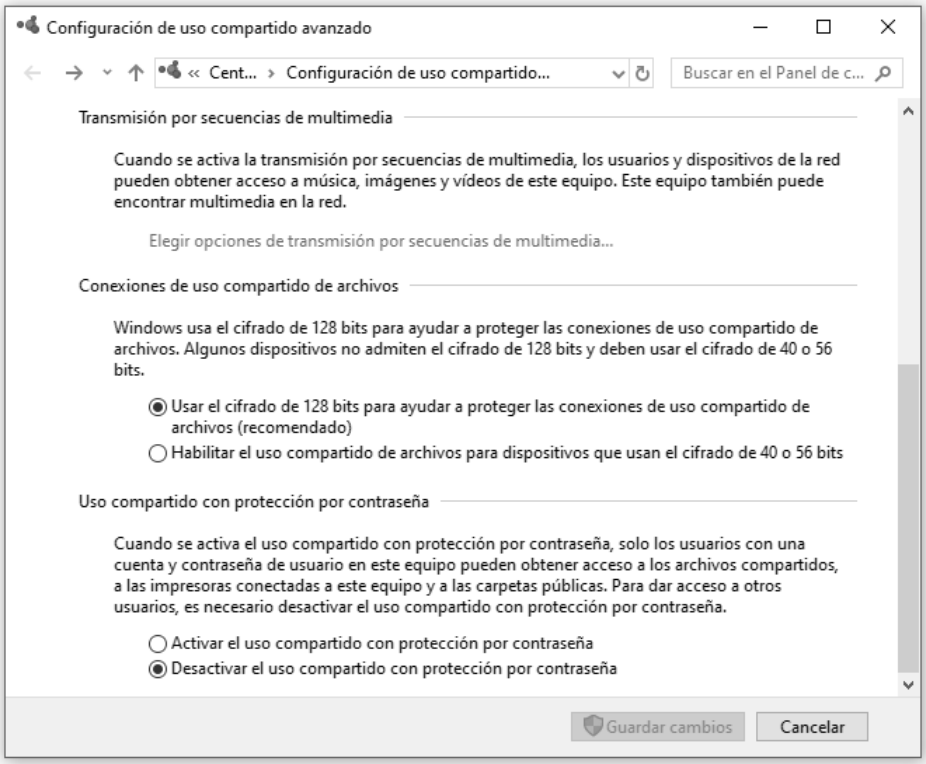


Figura C.4. Desactivar el uso compartido con protección por contraseña.

En el menú contextual emergente, pulsamos en la opción "Conceder acceso a". Aparecerá un submenú en el que deberemos presionar en "Usuarios específicos", ver figura C.5.

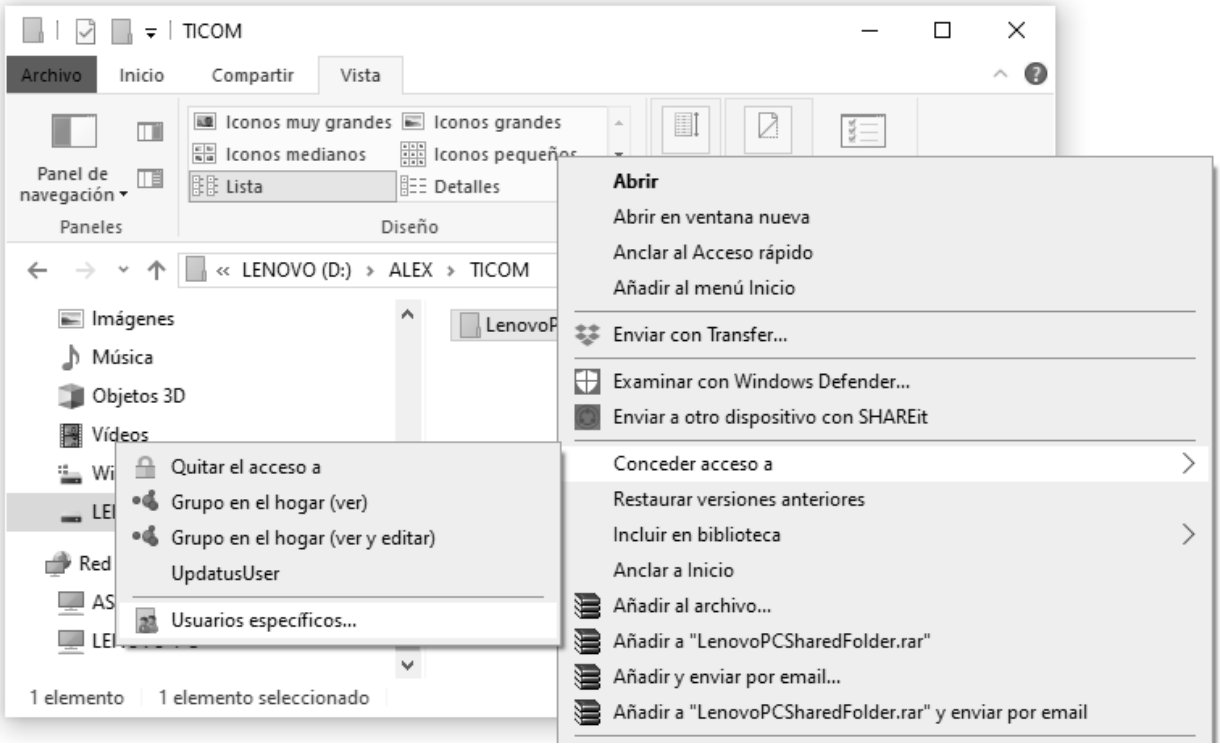


Figura C.5. Compartir una carpeta.

Una vez hecho esto accederemos a una ventana en la que podemos elegir las personas con las que vamos a compartir la carpeta. Dentro de ella pulsamos en la flechita hacia debajo de la lista desplegable que aparece en la parte superior, y seleccionamos la opción "Todos". Una vez seleccionada, pulsamos en el botón "Agregar" para añadir "Todos" a la lista (ver figura C.6). Al hacer esto, todos los usuarios que se conecten a nuestra red doméstica podrán acceder a nuestros archivos.

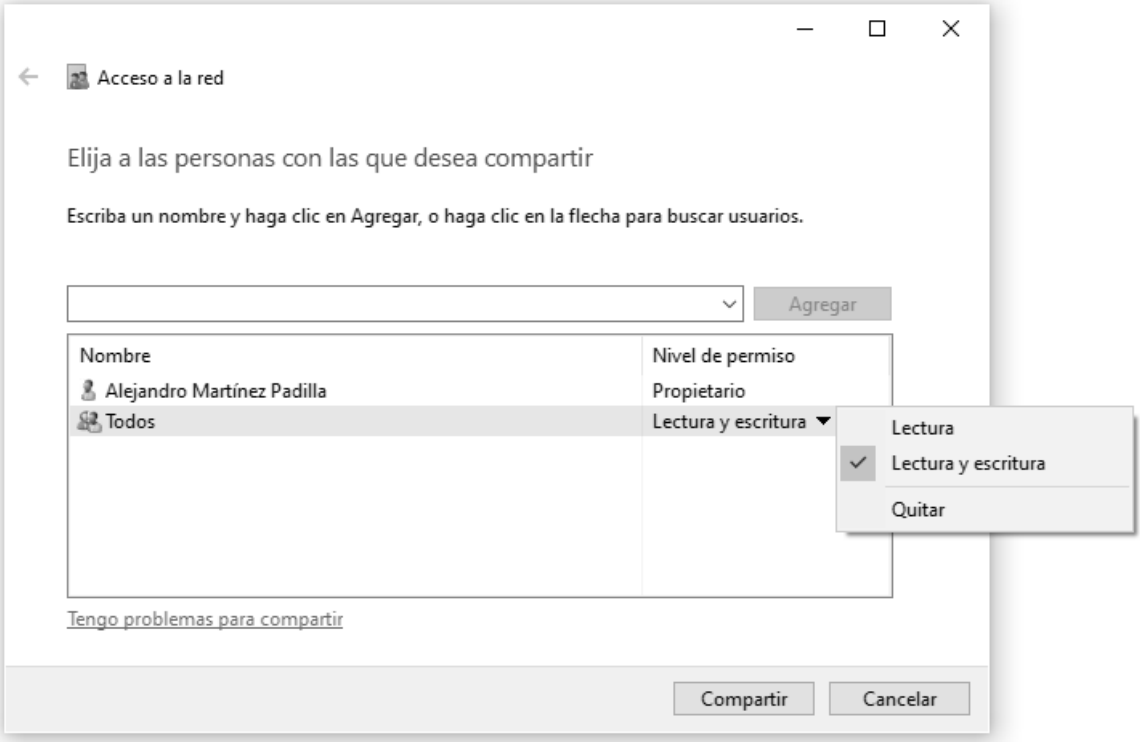


Figura C.6. Elegir a las personas con las que deseamos compartir.

A la derecha de cada usuario al que hayamos concedido acceso, ya sea "Todos" o un usuario concreto, podemos pulsar en la opción "Nivel de permiso" para decidir si pueden ver los archivos (Lectura), si pueden verlos y cambiarlos (Lectura y escritura), o si queremos eliminarlos de la lista. Una vez lo tengamos todo a nuestro gusto, pulsamos en el botón "Compartir" para guardar los cambios.

Con esto ya habremos dado permiso al resto de usuarios para acceder a la carpeta que acabamos de compartir. Ahora sólo nos queda seguir compartiendo todas las carpetas a las que queremos que otros equipos puedan acceder, incluyendo televisiones o dispositivos inteligentes.

Por supuesto, para compartir carpetas en otros ordenadores que hayamos conectado a la misma red doméstica, debemos seguir exactamente los mismos pasos.

C.3. COMPROBAR EL FUNCIONAMIENTO DE LA RED.

Acudiendo a la sección "Red" del Explorador de Archivos de Windows, podemos consultar qué equipos tenemos conectados a la red. En el caso de mi red doméstica, en el momento de escribir estos apuntes tenía conectados dos PCs (Lenovo-PC y Asus-PC), un reproductor multimedia (BDP-S5500), y el rúter (Sagemcom F@st5655), ver en la figura C.7. En casa no dispongo de impresora, pero de tenerla, sería muy interesante conectarla para poder compartirla entre todos los ordenadores de la red.

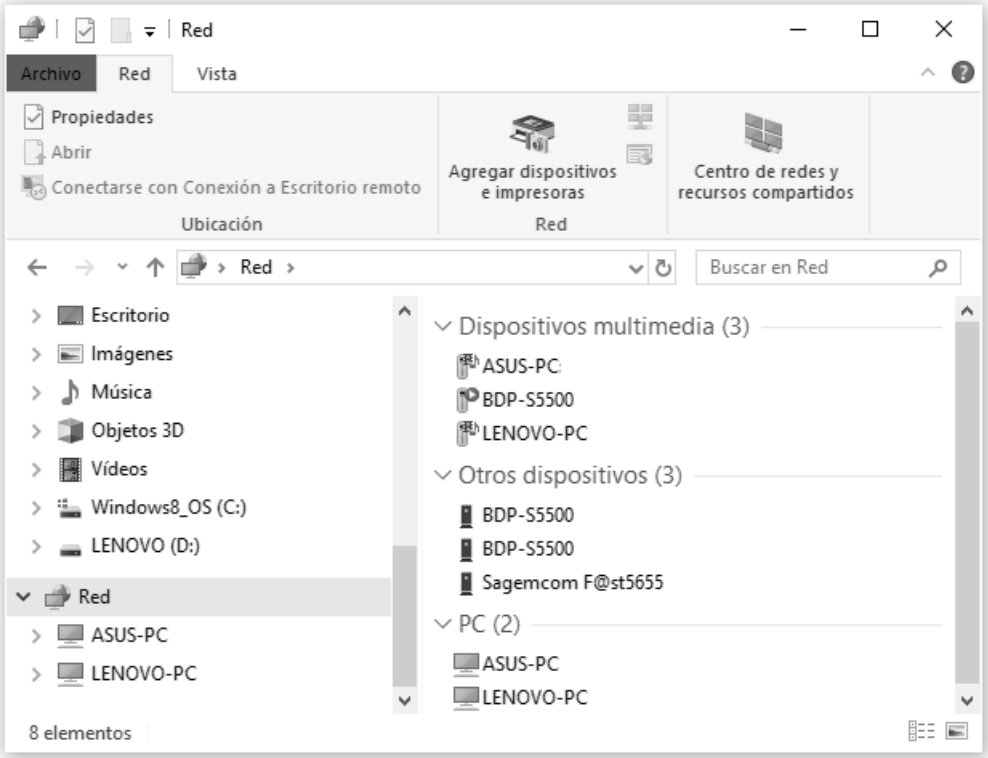


Figura C.7. Dispositivos en red.

Pulsando con el botón izquierdo del ratón sobre cualquiera de los equipos, y seleccionando la opción "Propiedades", podemos consultar datos relevantes de cada uno de los equipos conectados a la red, como su dirección física y su dirección IP (ver figura C.8). Si nos apuntamos la dirección IP de los dispositivos conectados a la red, podemos comprobar la conectividad haciendo un `ping` desde la consola de comandos. La figura C.9 muestra un ping desde mi ordenador (Lenovo-PC) al otro ordenador conectado a la red (Asus-PC), cuya dirección IP es 192.168.1.135.

Ahora comprobamos el acceso a las carpetas compartidas. En mi caso, y a modo de ejemplo, creé una carpeta compartida en cada uno de los ordenadores (carpeta LenovoPCSharedFolder y carpeta AsusPCSharedFolder), otorgándoles a ambas permisos de lectura y escritura. A continuación, creé un

archivo de texto en mi ordenador (Lenovo-PC), y lo copié en la carpeta compartida del otro ordenador (Asus-PC). El archivo se transfirió a través de la red doméstica desde mi ordenador al otro, apareciendo en la carpeta compartida del ordenador destino. A la inversa, creé un archivo de texto en la carpeta compartida de mi ordenador, y desde el otro ordenador accedí a esa carpeta para leer el contenido del archivo en remoto, consiguiéndolo sin problemas.

Otras pruebas que también podemos hacer es intentar conectar nuestro ordenador a una televisión inteligente, acceder a una impresora compartida, etc. Todo depende de qué dispositivos hayamos conectado a nuestra red.

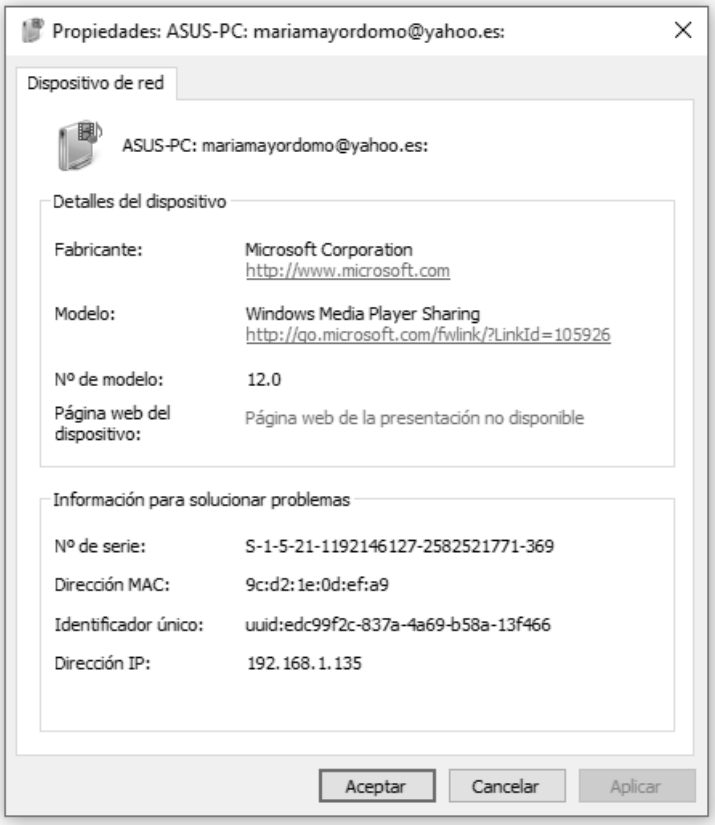


Figura C.8. Propiedades de uno de los PCs conectados a la red.

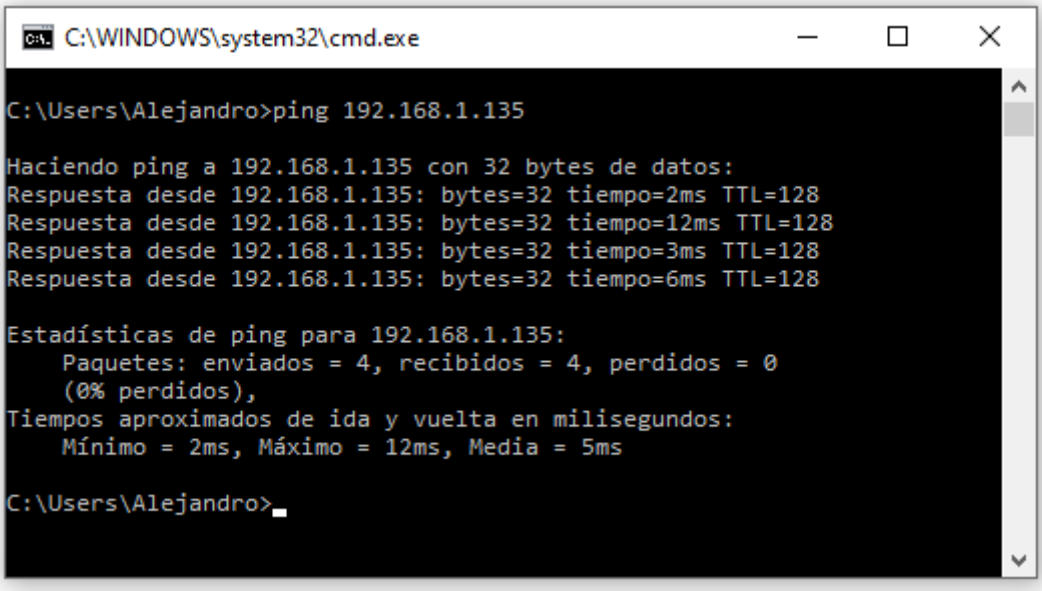


Figura C.9. Ping positivo a otro ordenador de la red.

2. ARQUITECTURA DE RED.

2.1. CAPAS DE PROTOCOLOS.

En el capítulo 1 ya definimos el concepto de protocolo. En comunicaciones de datos y redes, el **protocolo** define las reglas que el emisor, el receptor, y todos los dispositivos intermediarios deben seguir para poder comunicarse de forma efectiva. Cuando la comunicación es simple, puede que solo sea necesario un único protocolo sencillo. Pero cuando la comunicación es compleja, probablemente necesitaremos dividir la tarea entre varias **capas**, y en ese caso necesitaremos un protocolo en cada capa. A continuación vamos a presentar dos escenarios que nos permitirán entender la necesidad de dividir la comunicación en varias capas de protocolos.

PRIMER ESCENARIO.

En el primer escenario la comunicación es tan sencilla que puede darse en una sola capa. Supongamos que María y Ana son unas vecinas que comparten muchas ideas y aficiones comunes. La comunicación entre María y Ana puede desarrollarse en una sola capa, cara a cara, y en el mismo idioma (ver figura 2.1).

Figura 2.1. Protocolo de comunicación en una sola capa.

Incluso en este escenario tan simple, podemos ver que la comunicación necesita una serie de reglas que deben obedecerse. En primer lugar, María y Ana saben que deben saludarse cada vez que se ven. En segundo lugar, saben que deben limitar los temas de conversación al ámbito de su amistad (por ejemplo, no van a hablar de mecánica cuántica, la especialidad de María; ni de filosofía existencialista, la especialidad de Ana). En tercer lugar, cada una de ellas sabe que debe abstenerse de hablar cuando la otra está hablando. En cuarto lugar, ambas saben que la conversación debe ser un diálogo, no un monólogo: Ambas deben tener la oportunidad de hablar de sus cosas. Por último, las dos deben despedirse cuando se separan.

Como vemos, el protocolo de comunicación usado por María y Ana es diferente del protocolo de comunicación entre un profesor y sus alumnos en el transcurso de una clase. En el segundo caso, la comunicación es principalmente un monólogo: El profesor habla la mayor parte del tiempo, a no ser que un alumno tenga una pregunta, situación en la que el protocolo indica que el alumno debe levantar la mano y esperar a que el profesor le dé permiso para hablar. En este caso, la comunicación es muy formal y se limita al ámbito de la asignatura que el profesor está impartiendo.

SEGUNDO ESCENARIO.

En el segundo escenario vamos a imaginar que Ana se muda por trabajo a otra ciudad muy lejos de su vecina María. Las dos amigas quieren continuar hablando de sus ideas en común, porque han pensado en montar un negocio juntas. Para ello han decidido continuar sus conversaciones a través del correo postal ordinario. Sin embargo, no quieren que otras personas tengan acceso a las ideas de su futuro proyecto, en el caso de que las cartas sean interceptadas. En consecuencia, acuerdan usar un sistema de encriptado y desencriptado. La emisora de la carta encripta el mensaje para hacerlo ilegible a un hipotético intruso; la receptora de la carta lo desencripta para recuperar el contenido original.

En este caso podemos decir que la comunicación entre María y Ana ocurre en tres capas, como muestra la figura 2.2. En este modelo suponemos que María y Ana disponen de un ordenador con un software de reconocimiento de voz, un aparato codificador, y un secretario, que se encargan de realizar las tareas asignadas a cada una de las capas.

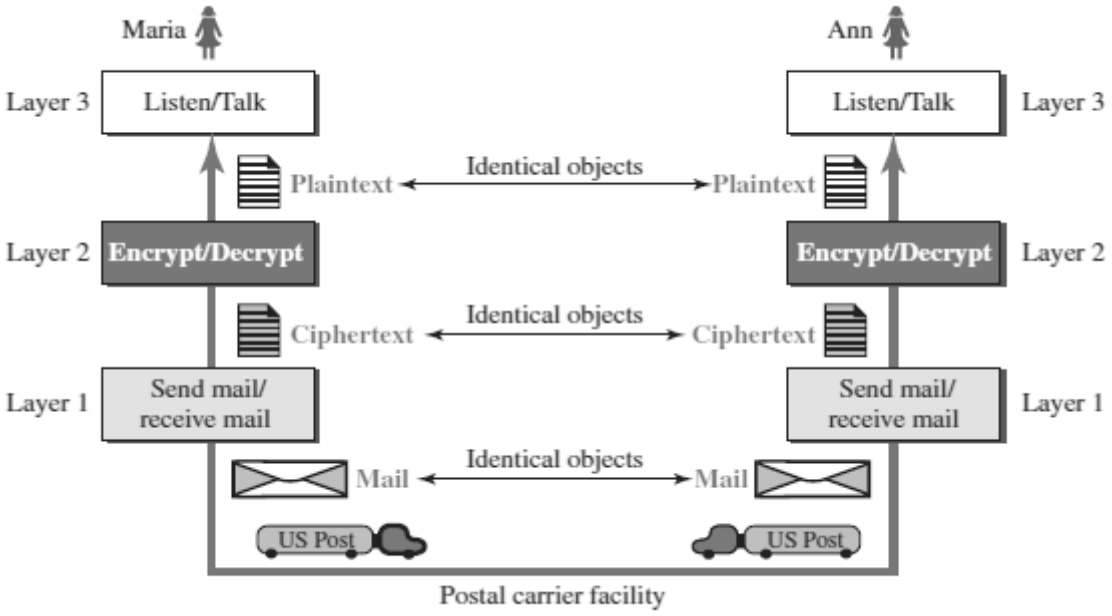


Figura 2.2. Protocolo de comunicación en tres capas.

Supongamos que María envía la primera carta a Ana. María habla con el ordenador en la tercera capa como si el ordenador fuese Ana y la estuviese escuchando. El ordenador en la tercera capa escucha lo que dice María, crea el texto plano (un texto en castellano), y se lo pasa al aparato codificador en la segunda capa. El codificador toma el texto plano, lo encripta para crear el texto cifrado, y se lo hace llegar al secretario de María en la primera capa. El secretario recibe el texto cifrado, lo imprime, lo mete en un sobre, añade las direcciones del destinatario y del remitente, le pone un sello, y lo envía por correo.

En el lado de Ana, el secretario en la primera capa recoge la carta del buzón de Ana, y a partir de la dirección del remitente, la reconoce como una carta de María. El secretario saca el texto cifrado del sobre y lo introduce en la máquina decodificadora en la segunda capa. Esta máquina decodifica el mensaje, crea el texto plano, y se lo envía al ordenador de Ana en la tercera capa. El ordenador recibe el texto plano y se lo lee a Ana como si fuese María la que está hablando (aunque con un extraño tono metálico).

UTILIDAD DEL MODELO EN CAPAS.

Las capas de protocolos nos permiten dividir una tarea compleja en varias tareas más pequeñas y sencillas. Por ejemplo, en la figura 2.2 podríamos tener una sola máquina que se encargase de las tareas del ordenador y del codificador. Pero si María y Ana decidiesen que el software de reconocimiento de voz del ordenador no es lo suficientemente bueno, se verían obligadas a cambiar toda la máquina. En el modelo en capas, solo necesitan cambiar el software de reconocimiento de voz en la tercera capa; el aparato codificador podría seguir usándose igual que antes. A esto se le denomina **modularidad**, y en este contexto significa tener capas independientes. Cada capa (o módulo) puede verse como una caja negra con puertos de entrada y de salida, que oculta los detalles de la forma en la que la caja procesa las entradas para producir las salidas. Si dos máquinas proporcionan las mismas salidas cuando se les dan las mismas entradas, una máquina puede reemplazar a la otra. Por ejemplo, Ana y María podrían comprar el aparato codificador a distintos fabricantes, pero si ambas máquinas crean el mismo texto cifrado a partir del mismo texto plano y viceversa, las dos máquinas son igual de válidas. Por otra parte, María podría decidir no contratar un secretario, y ocuparse del correo ella misma. Y Ana podría decidir escribir y leer ella misma los textos

planos, y no comprar un programa de reconocimiento de voz. Esto no afectaría a la comunicación: Mientras María y Ana asuman esas tareas en ambas direcciones, el sistema seguirá funcionando.

Otra ventaja del modelo en capas (que no podemos ver en este ejemplo tan sencillo, pero que entenderemos cuando discutamos el modelo en capas de Internet) es que la comunicación no siempre se limita a pasar por los dos sistemas en los extremos; ciertas comunicaciones también necesitan pasar por algunos sistemas intermediarios que solo implementan algunas de las capas más bajas, en lugar de implementarlas todas. Si no existiese este modelo en capas, los sistemas intermediarios tendrían que ser igual de complejos que los sistemas en los extremos, lo que implicaría que el sistema de comunicaciones en conjunto fuese mucho más costoso.

CONEXIONES LÓGICAS ENTRE CAPAS IGUALES.

De todo lo mencionado en los apartados previos vemos que existe una especie de comunicación directa entre capas iguales. En efecto, aunque dentro de cada componente del sistema la comunicación *real* se produce "en vertical" entre una capa y su capa inmediatamente inferior/superior, podemos considerar que existe una comunicación *virtual* "en horizontal" entre capas de igual nivel en distintos componentes del sistema (ver figura 2.3).

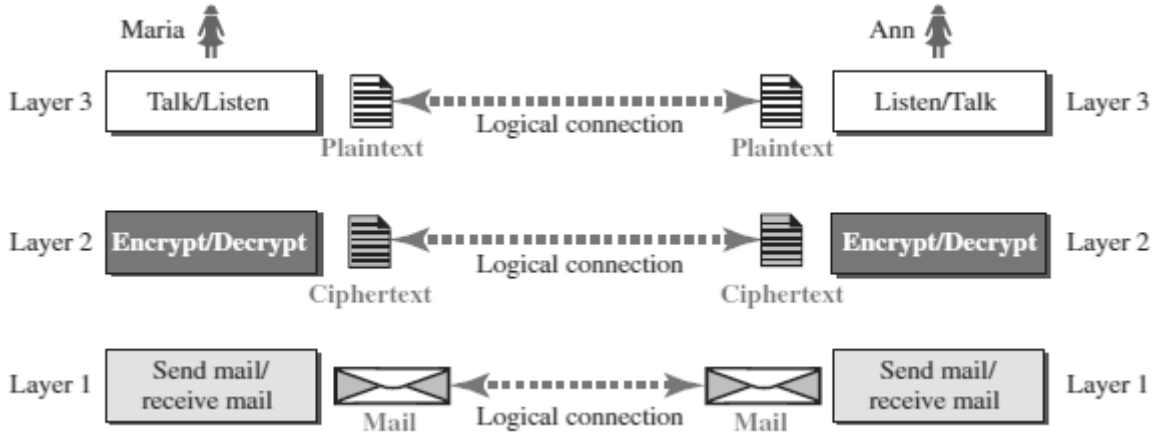


Figura 2.3. Comunicación virtual (conexión lógica) entre capas iguales.

Así, María y Ana pueden imaginar que hay una **conexión lógica** (imaginaria) entre capas iguales, a través de la cual una capa puede enviar directamente el objeto creado a la otra capa de igual nivel. En efecto, María y Ana perciben que su comunicación ocurre como si estuviesen cara a cara hablando directamente entre ellas. Los programas de reconocimiento de voz en la capa tres piensan que la comunicación entre ellos es directa, y que se produce intercambiando los textos planos escritos en castellano. Las máquinas de encriptado en la capa dos también creen que se comunican directamente intercambiando textos cifrados en el código común elegido. Por último, los secretarios también están en comunicación directa intercambiando cartas. Sin embargo, y como hemos visto, la comunicación ocurre realmente entre la capa superior y la inferior, y el objeto creado por una capa realmente se transfiere de un lado al otro pasándoselo a la capa de abajo.

2.2. LA PILA DE PROTOCOLOS TCP/IP.

¿QUÉ ES TCP/IP?

TCP/IP (Transmission Control Protocol/Internet Protocol) es el nombre de la arquitectura de red que se usa hoy día en Internet. Se trata de un conjunto de protocolos organizados en capas, donde cada capa define e implementa una parte del proceso general de transmitir la información a través de Internet.

Hoy día, TCP/IP es un modelo en cinco capas como el mostrado en la figura 2.4. Las dos capas superiores casi siempre están implementadas mediante software; por su parte, las capas inferiores son una combinación de hardware y software, excepto la capa física, que es principalmente hardware. Dentro de una cierta máquina, cada capa llama a los servicios de la capa inmediatamente inferior y proporciona sus servicios a la capa inmediatamente superior. Por ejemplo, la capa 3 usa los servicios proporcionados por la capa 2, y proporciona servicios a la capa 4. Entre dos máquinas distintas, la capa N en una máquina se comunica lógicamente con la misma capa N en otra máquina. Esta comunicación lógica está gobernada por el conjunto de reglas y convenciones dictadas por el protocolo de esa capa.

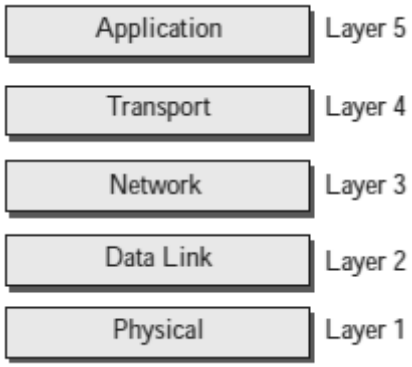


Figura 2.4. Capas de TCP/IP.

Las cinco capas del modelo TCP/IP son las capas de **aplicación**, **transporte**, **red**, **enlace de datos**, y **física**. Cada una de las capas implementa algunas de las tareas que implica el proceso de comunicación global. De forma resumida, las tareas asumidas por cada capa son los siguientes:

La **capa de aplicación** es la más alta del modelo y la más cercana al usuario y sus programas. Esta capa gestiona la comunicación entre los programas de aplicación que están ejecutándose en los hosts origen y destino. (A estos programas en ejecución se les denomina **procesos**). Por ejemplo, cuando un usuario se conecta a una página web, la capa de aplicación gestiona (con la ayuda del protocolo HTTP) la comunicación del programa navegador en el PC del usuario con el proceso servidor en el ordenador que aloja la página web. Si al mismo tiempo el usuario se está descargando un archivo, la capa de aplicación gobierna (en base al protocolo FTP) la comunicación entre el programa de descargas en el PC del usuario y el proceso en el servidor que aloja el archivo. Notar que en un mismo ordenador (en este caso, el PC del usuario) puede haber múltiples procesos simultáneos estableciendo comunicaciones independientes a través de Internet.

La **capa de transporte** garantiza la entrega extremo a extremo del mensaje completo entre un proceso origen y un proceso destino. Se ocupa de tareas como la identificación de los distintos procesos de un host mediante los **números de puerto** y de la segmentación y reensamblando de los mensajes. En las aplicaciones que requieren un servicio orientado a conexión también se encarga del establecimiento y la gestión de la conexión, y del control del flujo, de la congestión, y de los errores.

La **capa de red** es la responsable de llevar los paquetes entre un host origen y un host destino. Las tareas que asume esta capa son la identificación unívoca de los hosts origen y destino (en base a las **direcciones IP**), y el encaminamiento de los paquetes dentro del laberinto de redes que es Internet, a través de la mejor ruta posible. La ruta entre el host origen y el host destino probablemente pasará por varias redes intermedias, cada una de ellas con una tecnología distinta. Aunque los rúters son los encargados de seleccionar la siguiente red de la ruta hacia el destino, cuando los paquetes llegan a una cierta red se necesita otra capa que mueva los paquetes dentro de esa red, en base a la tecnología empleada. Esa es precisamente la tarea de la capa de enlace de datos.

La **capa de enlace de datos** se ocupa de mover los paquetes dentro de las distintas redes por las que pasa la ruta hasta el destino. Las funciones que tiene asignadas son el entramado de los datos, la definición de

las **direcciones físicas** para identificar a los distintos equipos conectados a una misma red, y el control del acceso al medio en los enlaces compartidos.

La **capa física** es la más baja del modelo, ya que se sitúa justo por encima del medio físico de transmisión. Esta capa se ocupa de coordinar las funciones requeridas para transmitir los bits de datos a través del medio físico, en función de la tecnología de red empleada y del tipo de medio de transmisión. Estas tareas incluyen la especificación del mecanismo de representación los bits (codificación); la conversión de los bits en señales eléctricas, ópticas, u ondas; y la definición de las características físicas de los interfaces y de los cables/fibras, de la velocidad de transmisión, del modo de transmisión (half – dúplex o full - dúplex), de la topología física, etc.

FUNCIONAMIENTO DEL MODELO TCP/IP.

Para mostrar la forma en la que funciona el modelo en capas TCP/IP, vamos a usar como ejemplo la pequeña interred mostrada en la figura 2.5. Esta interred está compuesta por tres redes (en este caso, tres LANs), a las que en la jerga de Internet denominamos **enlaces** (links). Cada uno de los enlaces tiene un conmutador (switch), al que podría haber conectados más ordenadores aparte de los mostrados en la figura. Los tres enlaces están interconectados entre sí mediante un rúter.

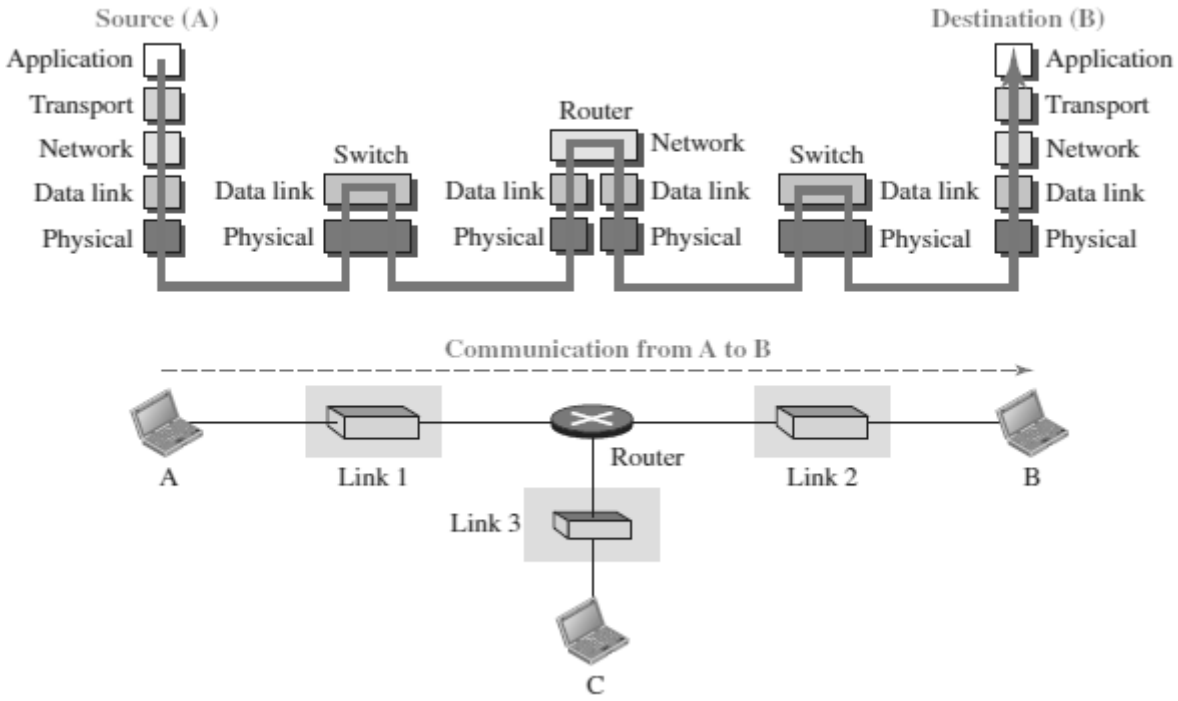


Figura 2.5. Comunicación a través de una interred.

Supongamos que el ordenador A se comunica con el ordenador B. Como vemos en la figura, en esta comunicación intervienen cinco dispositivos: El host de origen (ordenador A), el conmutador en el enlace 1, el rúter, el conmutador en el enlace 2, y el host destino (ordenador B). Cada dispositivo implementa un cierto número de capas del modelo, dependiendo del papel que desempeña dentro de la interred. Los dos hosts son los equipos finales que desean comunicarse, y están implicados en las cinco capas del modelo: El **host origen** necesita crear el mensaje en la capa de aplicación, y enviarlo hacia abajo capa a capa para poder transmitirlo físicamente al host destino. El **host destino** necesita recibir los datos en la capa física y enviarlos hacia arriba capa a capa hasta la capa de aplicación.

Los **rúters** son dispositivos que permiten interconectar distintas redes, y operan encaminando los datos de una red a otra. Por lo tanto, los rúters son dispositivos más sencillos que los hosts, y solo necesitan implementar las tres capas más bajas (física, enlace, y red); no incluyen la capa de transporte ni la de aplicación. Aunque todo rúter está implicado en una sola capa de red, también se ve involucrado en n capas

de enlace y físicas, siendo n el número de enlaces a los que el rúter está conectado. La razón para ello es que cada enlace (cada red LAN) podría usar sus propios protocolos de capa física y de capa de enlace. Por ejemplo, en la figura 2.5 el rúter está conectado a tres enlaces, pero el mensaje enviado desde el host A hasta el host B solo atraviesa dos enlaces. Cada enlace podría usar diferentes tecnologías en sus capas de enlace y física, y el rúter tendría que recibir el paquete del enlace 1 basándose en la tecnología del primer enlace (por ejemplo, Ethernet), y entregarlo al enlace 2 en base a la tecnología de ese segundo enlace (por ejemplo, Token Ring).

Los **conmutadores** son dispositivos cuya misión es permitir la interconexión de los distintos equipos conectados a una misma red. Por consiguiente, un conmutador es un dispositivo más simple que un rúter, y solo se ve implicado en dos capas: La capa física y la capa de enlace. Aunque cada conmutador de la figura tiene dos conexiones diferentes, las conexiones están dentro del mismo enlace (la misma red LAN), donde solo opera una tecnología de red. Esto significa que, al contrario que un rúter, un conmutador de capa de enlace solo está implicado en una capa física y en una capa de enlace.

LAS CAPAS DEL MODELO TCP/IP.

A continuación vamos a describir las distintas capas de la pila TCP/IP con más detalle. Para ello, nos resultará más fácil usar como ejemplo la interred privada de la figura 2.6, en lugar de utilizar la Internet mundial. Esta interred está compuesta por varios enlaces (redes), cada uno de ellos con un conmutador de capa de enlace al que se conectan todos los hosts de esa red. La figura solo muestra los dos hosts finales, a saber, los ordenadores A y B. Los distintos enlaces de la interred están interconectados entre sí mediante rúters. La interred incluye un total de seis enlaces y cuatro rúters.

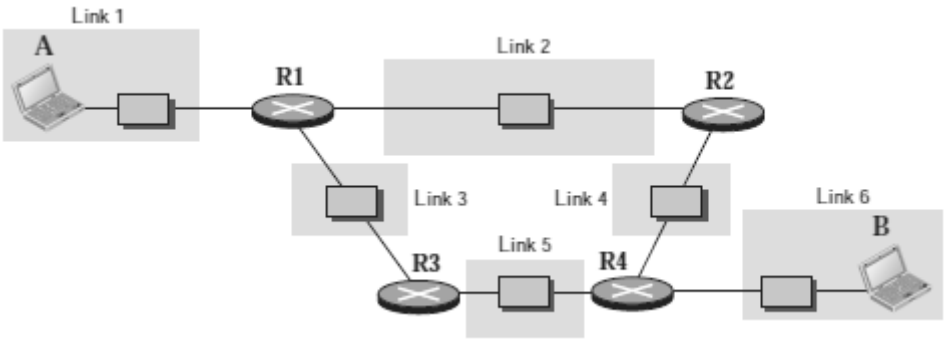


Figura 2.6. Interred de ejemplo.

Capa física.

La **capa física** es la responsable de transportar los bits individuales a través del enlace, y se encarga de definir aspectos como los voltajes de las señales eléctricas empleadas para representar esos bits, la velocidad y el modo de transmisión, la estructura de los cables y los conectores, etc. Aunque la capa física es la más baja de la pila TCP/IP, todavía hay otra capa oculta bajo ella: El **medio de transmisión** (el cable, la fibra, el aire, etc.). Sin embargo, el medio de transmisión no transporta bits, sino las señales eléctricas u ópticas que los representan. Por consiguiente, la capa física del nodo emisor se encarga de recibir los bits de datos de la capa superior (la capa de enlace), y de trasformarlos en señales para enviarlas a través del medio de transmisión. Por su parte, la capa física del nodo destino se ocupa de recibir las señales eléctricas del medio, y de recuperar los bits enviados para pasárselos a su capa de enlace.

TCP/IP no define ningún protocolo para la capa física, y utiliza la capa física de la tecnología de red usada en cada enlace. Algunos ejemplos son las capas físicas de tecnologías de red LAN como **Ethernet**, Token Ring, FDDI, y **WiFi**; las capas físicas de tecnologías de red WAN punto a punto como DSL, SONET, y PPP; y las capas físicas de tecnologías de red WAN conmutada como X.25, Frame Relay, y ATM.

La figura 2.7 muestra la comunicación entre los hosts A y B a nivel de capa física. Como vemos, la comunicación es de **nodo a nodo**. Aquí supondremos que la red ya ha determinado la ruta más eficiente para comunicar a los ordenadores A y B, a través de los rúters R1, R3, y R4. Notar que si un rúter está conectado a n enlaces, necesita n protocolos de capa física, porque cada enlace podría usar una tecnología diferente. Sin embargo, los hosts A y B solo están conectados a un enlace, y solo necesitan un protocolo de capa física. Como vemos en la figura, el viaje de los bits entre los ordenadores A y B está compuesto por cuatro trayectos más pequeños, a saber, los enlaces 1, 3, 5, y 6. El ordenador A envía los bits al rúter R1 en el formato dictado por el protocolo del enlace 1. A su vez, el rúter 1 envía los bits al rúter 3 en el formato dictado por el protocolo usado en el enlace 3, y así sucesivamente. El rúter R1 tiene tres capas físicas (aunque la figura solo muestra las dos capas físicas involucradas en la comunicación). La capa física conectada al primer enlace recibe los bits en el formato del protocolo usado por el enlace 1, mientras que la capa física conectada al enlace 3 envía los bits según el formato del protocolo del enlace 3. Y lo mismo ocurre en los otros rúters implicados en la comunicación.

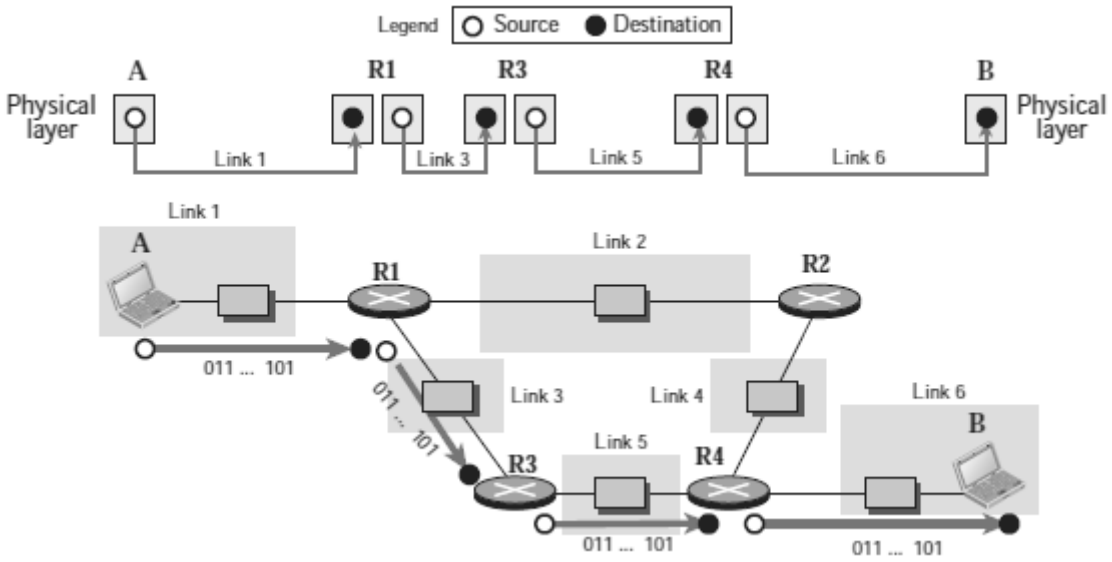


Figura 2.7. Comunicación en la capa física.

Capa de enlace de datos.

Ya hemos visto que una interred está compuesta por varios enlaces (redes LAN y/o WAN) interconectados mediante rúters. Un paquete podría tener que pasar por varios enlaces en su viaje desde el host origen al host destino. Los rúters se encargan de elegir el siguiente enlace de la ruta hacia el destino. Sin embargo, después de que un rúter haya seleccionado el siguiente enlace, la **capa de enlace de datos** es la responsable de tomar el paquete y de moverlo a través de ese enlace. El enlace podría ser una LAN cableada con un conmutador, una LAN inalámbrica, una WAN punto a punto, etc. También podríamos tener diferentes protocolos en cada enlace. En cualquier caso, la capa de enlace es la encargada de mover el paquete a través del enlace. Para cumplir con este cometido, la capa de enlace utiliza un esquema de **direcciones físicas** (también denominadas *direcciones de capa de enlace* o *direcciones MAC*) que permite identificar de forma unívoca a cada uno de los nodos de un enlace.

TCP/IP tampoco define ningún protocolo específico para la capa de enlace; de hecho, soporta todos los protocolos de las tecnologías de red empleadas en los enlaces. Cualquier protocolo que sea capaz de tomar un datagrama de capa de red y de llevarlo a través del enlace es igual de válido. La capa de enlace de datos recibe un datagrama de la capa de red, y lo *encapsula* en un paquete llamado **trama**.

La figura 2.8 muestra la comunicación entre A y B a nivel de capa de enlace. Como vemos, la comunicación también es de **nodo a nodo**. Como puede haber múltiples dispositivos conectados a un mismo enlace, todas las tramas que circulan por un enlace incorporan las direcciones físicas de los nodos emisor y receptor. En el primer enlace, las tramas que envía el host A deben llegar al rúter 1, que se encargará de pasarlas al

siguiente enlace de la ruta. Por consiguiente, esas tramas llevarán como dirección de origen la dirección física del host A, y como dirección de destino la dirección física del rúter 1. En el segundo enlace, las tramas que envía el rúter 1 van destinadas al rúter 3, por lo que esas tramas llevarán como dirección de origen la dirección física del rúter 1 y como dirección de destino la dirección física del rúter 3, etc.

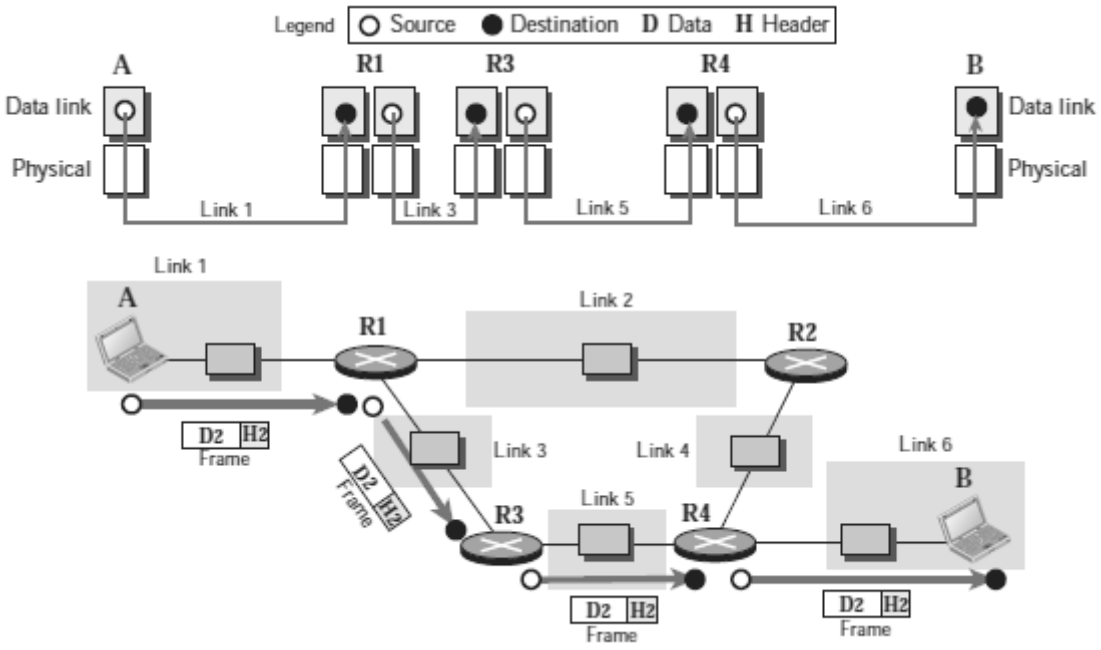


Figura 2.8. Comunicación en la capa de enlace.

De nuevo, cada rúter necesita tantos protocolos de capa de enlace como enlaces tiene conectados, porque cada enlace podría usar una tecnología distinta. Ello implica que las tramas que viajan por distintos enlaces pueden tener formatos diferentes. Por ejemplo, el rúter R1 necesita tres capas de enlace, porque está conectado a tres enlaces. (La figura solo muestra las dos capas involucradas en la comunicación). Así, cuando el rúter R1 recibe una trama desde el enlace 1, se la pasa a la capa de enlace correspondiente al enlace 1. Aquí abre la trama, desencapsula los datos, y se los pasa a la capa de enlace correspondiente al enlace 3. Esta capa vuelve a encapsular los datos para generar una nueva trama adaptada a la tecnología del enlace 3, que reenvía al rúter R3.

Notar también que la figura no muestra el movimiento físico de las tramas: La transmisión física solo ocurre en la capa física. A nivel de capa de enlace, la comunicación entre dos nodos es lógica, no física. Esta situación ilustra el concepto de **conexión lógica** del que hablamos antes: La capa de enlace del host A *cree* que las tramas que envía llegan directamente a la capa de enlace del rúter R1. En realidad, lo que ocurre es que la capa de enlace de A entrega la trama a su capa física, para que la transforme en un flujo de bits que transmite a través del medio físico con destino al rúter R1. Los bits llegan a la capa física de R1, la cual se los pasa a su capa de enlace, donde recupera la trama enviada por A. Ello da la impresión de que la trama se transfiere de forma directa entre las dos capas de enlace, aunque en la práctica no sea así.

Capa de red.

La **capa de red** es la responsable de garantizar la entrega de los datos entre el host origen y el host destino. Por consiguiente, la comunicación en esta capa es de host a host. Sin embargo, como podría haber varios enlaces entre origen y destino, los rúters se encargan de elegir la ruta óptima para cada paquete. Se puede decir que la capa de red se encarga de la comunicación de host a host y del encaminamiento del paquete por la mejor ruta posible. Para cumplir con esta tarea, la capa de red utiliza un esquema de **direcciones lógicas** denominadas *direcciones de capa de red*, o de forma más habitual, **direcciones IP**. Estas direcciones lógicas identifican de forma conjunta al host y a la red a la que pertenece ese host.

Tal vez nos preguntemos por qué necesitamos una capa de red. Como la comunicación en la capa de transporte también es de host a host, podríamos asignar la tarea del encaminamiento a la capa de transporte y ahorrarnos una capa. Una de las razones es la separación de las diferentes tareas en distintas capas. Otra razón es que los rúters no necesitan las capas de transporte ni de aplicación. El hecho de separar las tareas nos permite tener menos protocolos en los rúters.

En Internet, el protocolo principal en la capa de red es el **protocolo IP** (IP = Internet Protocol), el cual define el formato del paquete de capa de red, al que se le denomina **datagrama**. IP también especifica el formato y la estructura de las direcciones que se usan en esta capa (direcciones IP). Por último, IP es responsable de encaminar los paquetes desde su origen a su destino, lo que se consigue haciendo que cada rúter reenvíe los datagramas al siguiente rúter de la ruta. IP es un protocolo sin conexión que no proporciona control de flujo, ni control de errores, ni de control de congestión. Esto significa que si una aplicación necesita alguno de estos servicios, debe obtenerlos del protocolo de la capa de transporte.

La capa de red también incluye varios protocolos (por ejemplo, los protocolos **RIP**, **IGRP**, **OSPF**, etc.) para que los rúters puedan decidir cuál es el siguiente enlace de la ruta, tanto en el encaminamiento unicast (uno a uno) como en el multicast (uno a muchos). Estos protocolos no se encargan del encaminamiento propiamente dicho (eso es responsabilidad de IP); su misión es crear las **tablas de encaminamiento** que usan los rúters para elegir el enlace de salida al que reenviar los paquetes recibidos. Además del protocolo IP y de los protocolos de encaminamiento, la capa de red también utiliza algunos protocolos auxiliares que ayudan a IP a realizar sus tareas: El protocolo **ICMP** (Internet Control Message Protocol) ayuda a IP a reportar los problemas que puedan surgir al encaminar un paquete. El protocolo **DHCP** (Dynamic Host Configuration Protocol) ayuda a obtener automáticamente la dirección IP de un host (en vez de configurarla manualmente). El protocolo **ARP** (Address Resolution Protocol) ayuda a IP a obtener la dirección física de un nodo a partir de su dirección lógica. El protocolo **NAT** (Network Address Translation) permite establecer la correspondencia entre las direcciones IP privadas de los equipos conectados a una red LAN con las direcciones IP públicas de esa red en Internet, etc.

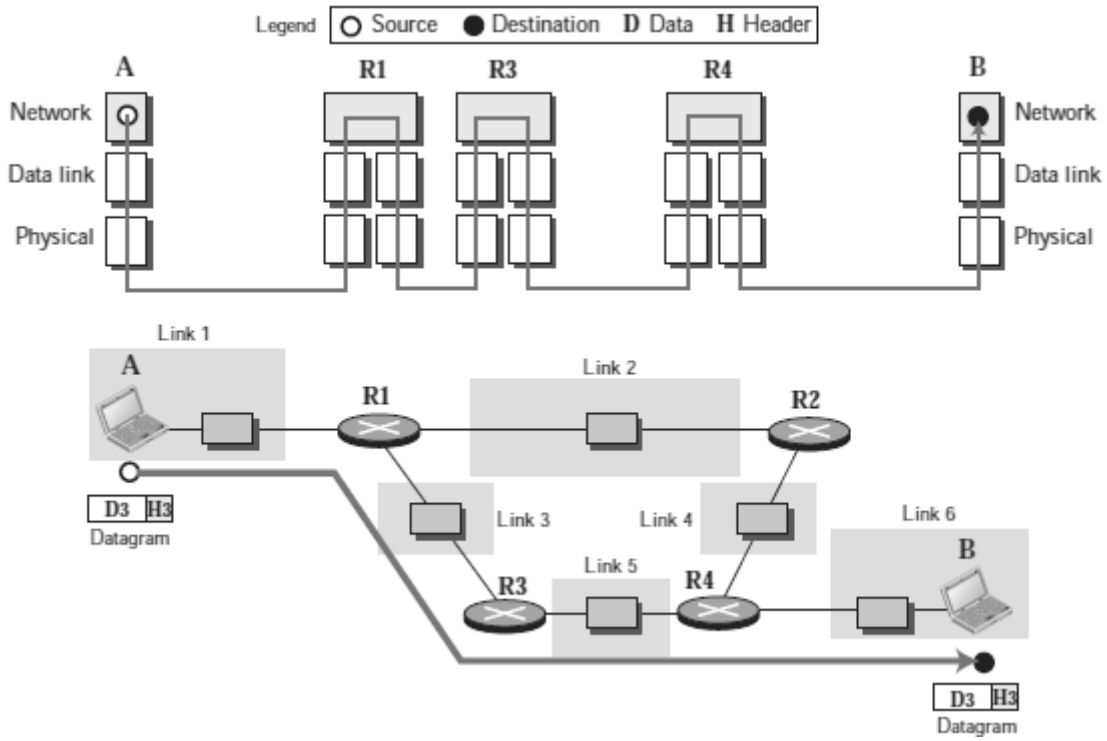


Figura 2.9. Comunicación en la capa de red.

La figura 2.9 muestra la comunicación a nivel de capa de red. Como vemos, la comunicación en la capa de red es **extremo a extremo**, mientras que en las capas de enlace y física era de nodo a nodo. El datagrama que envía el ordenador A es exactamente el mismo datagrama que recibe el ordenador B (a menos que necesite fragmentarlo en el caso de que sea demasiado grande para pasárselo al siguiente enlace). Las capas de red

de los rúters pueden inspeccionar las direcciones IP de origen y de destino del datagrama para determinar la mejor ruta, pero no pueden cambiar los contenidos ni el formato del datagrama.

De nuevo, la comunicación extremo a extremo a nivel de capa de red es lógica, no física. Aunque la capa de red del ordenador A cree que está enviando los datagramas directamente a la capa de red del ordenador B, la comunicación real se produce pasando los datos a las capas inferiores, y transmitiéndolos a través del medio físico.

Capa de transporte.

La conexión lógica en la **capa de transporte** también es extremo a extremo. En el host de origen, la capa de transporte recibe el mensaje creado en la capa de aplicación, lo encapsula en un paquete de capa de transporte (denominado **segmento** o **datagrama de usuario**, según el protocolo), y lo envía a través de la conexión lógica (imaginaria) a la capa de transporte del host de destino. En otras palabras, la capa de transporte se encarga de dar servicio a la capa de aplicación, lo que consiste en obtener el mensaje del programa de aplicación que se está ejecutando en la máquina origen, y enviarlo al programa de aplicación correspondiente en la máquina destino. Para identificar a los distintos programas de aplicación que podrían estar ejecutándose simultáneamente tanto en el host origen como en el host destino, la capa de aplicación utiliza un esquema de direccionamiento basado en los **números de puerto**, de los que hablaremos más detenidamente en secciones posteriores.

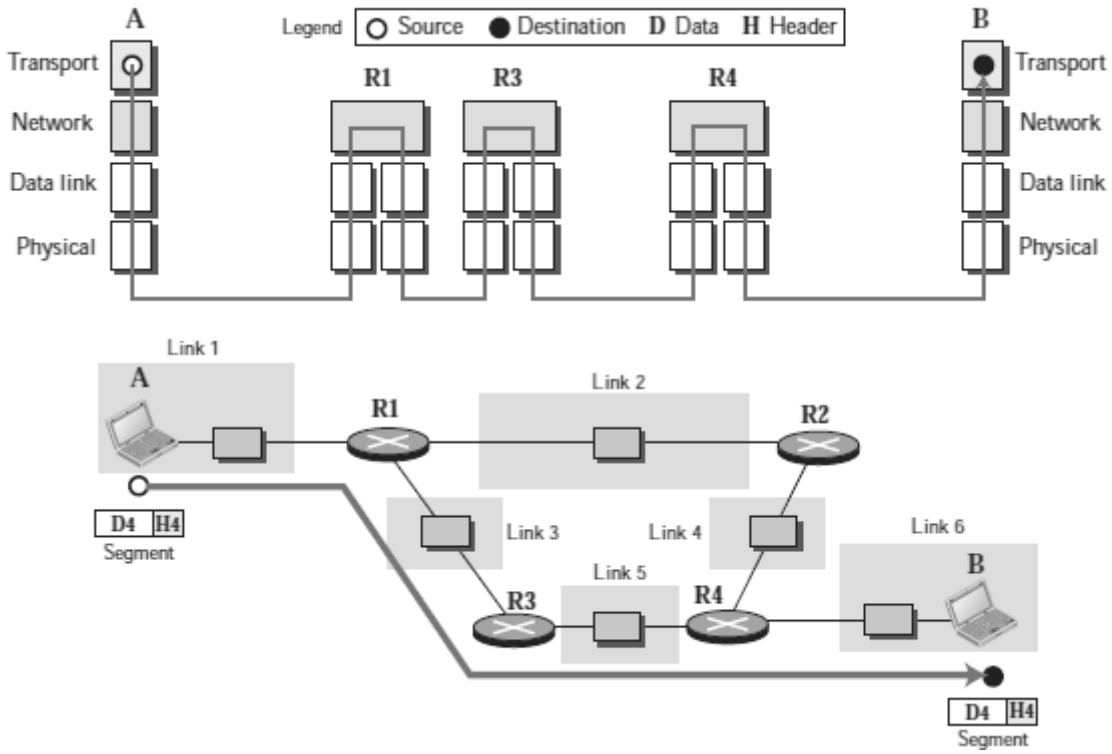


Figura 2.10. Comunicación en la capa de transporte.

La pila TCP/IP define varios protocolos de capa de transporte, cada uno de ellos diseñado para ofrecer un cierto tipo de servicio. Dependiendo de las necesidades de los programas de aplicación, el protocolo seleccionado será uno u otro. El protocolo principal, llamado **TCP** (**Transmission Control Protocol**), es un protocolo **orientado a conexión** que establece una conexión lógica inicial entre las capas de transporte de los dos hosts antes de transferir los datos. Esta conexión es como una tubería lógica entre las dos capas de transporte para transferir los datos a través de ella. TCP proporciona **control de flujo** (esto es, la equiparación de la velocidad de transmisión de datos del host de origen con la velocidad de recepción de datos del host de destino para evitar la sobrecarga del destino), **control de errores** (para garantizar que los segmentos llegan al destino sin errores, y para reenviar aquellos que se reciban con errores), y **control de congestión** (para reducir la pérdida de segmentos debida a la congestión de la red). Gracias a estas características, TCP garantiza que los paquetes enviados llegan al destino en el orden adecuado y libres de

errores. El otro protocolo de uso común es **UDP** (**Usar Datagram Protocol**), un protocolo **sin conexión** que transmite datagramas de usuario sin crear una conexión lógica previa. En UDP cada datagrama de usuario es una entidad independiente sin relación con el datagrama previo ni con el siguiente. UDP es un protocolo sencillo que no proporciona control de flujo, ni control de errores, ni control de congestión. Su simplicidad es muy interesante para programas de aplicación donde sea asumible una cierta tasa de paquetes perdidos, desordenados, o erróneos. También es adecuado para aplicaciones que necesiten enviar mensajes cortos y que no puedan permitirse la retransmisión que ocurre en TCP cuando un paquete se pierde o llega corrupto.

La figura 2.10 muestra la comunicación a nivel de capa de transporte. Como en el caso de la capa de red, la comunicación es **extremo a extremo**. La principal diferencia entre la capa de transporte y la de red es que, aunque todos los nodos (hosts y rúters) de la interred deben incluir una capa de red, solo los dos extremos de la comunicación (los dos hosts finales) necesitan una capa de transporte. La capa de red es responsable de enviar los datagramas del ordenador A al ordenador B, mientras que la capa de transporte se encarga de entregar el mensaje completo (en la forma de un segmento o un datagrama de usuario) del programa de aplicación en el host A al programa de aplicación en el host B. Un segmento puede consistir en unos pocos datagramas o en varios cientos de ellos. La capa de red debe ocuparse de dividir el segmento en datagramas, y de entregárselos a la red. Como en Internet cada datagrama se envía por una ruta distinta, los datagramas pueden llegar desordenados o se pueden perder por el camino. La capa de transporte en el ordenador B debe esperar a que lleguen todos los datagramas para volver a construir el segmento.

Capa de aplicación.

De nuevo, la conexión lógica entre dos capas de aplicación es extremo a extremo. Las dos capas de aplicación en ambos extremos intercambian **mensajes**, como si hubiese una conexión directa entre las dos capas. Sin embargo, y como ya sabemos, la comunicación real se produce a través de todas las capas. Ahora bien, al contrario que en las capas de red y de transporte, la comunicación en la **capa de aplicación** no se establece entre el host origen y el host destino, sino entre dos *procesos* en los hosts origen y destino. (Un **proceso** en un programa de aplicación que está ejecutándose en el host origen o en el host destino). Para poder comunicarse, uno de los procesos envía una solicitud al otro proceso, y recibe una respuesta. La tarea de la capa de aplicación es gestionar esta comunicación proceso a proceso.

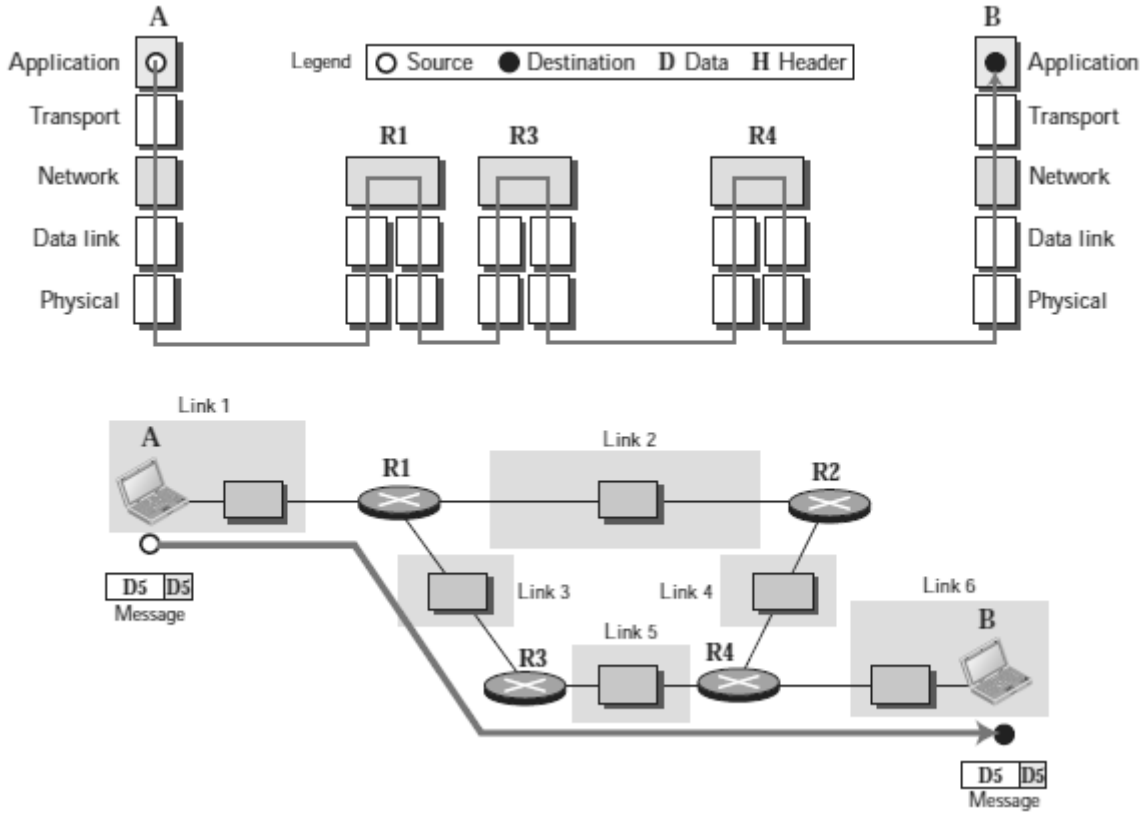


Figura 2.11. Comunicación en la capa de aplicación.

En Internet, la capa de aplicación incluye muchos protocolos predefinidos: El protocolo **HTTP** (**Hypertext Transfer Protocol**) sirve para acceder a la World Wide Web. **SMTP** (**Simple Mail Transfer Protocol**) es el protocolo principal usado en el servicio de correo electrónico. El protocolo **FTP** (**File Transfer Protocol**) se usa para transferir archivos de un host a otro. Los protocolos **TELNET** (**Terminal Network**) y **SSH** (**Secure Shell**) sirven para acceder remotamente a una máquina. El protocolo **DNS** (**Domain Name System**) ayuda a otros protocolos a obtener la dirección lógica de un ordenador a partir de su nombre, etc.

Notar que los programas de aplicación como Google Chrome, Outlook, Bit Torrent, Whatsapp, etc., no residen en la capa de aplicación. Estos programas (y sus usuarios) están fuera del modelo, por encima de la capa de aplicación. Por ejemplo, un usuario podría usar el navegador Google Chrome para acceder a una página web en un servidor. Este programa no está en la capa de aplicación, pero está soportado por el protocolo HTTP que sí reside en esta capa.

La figura 2.11 muestra la comunicación a nivel de capa de aplicación. Como vemos, la comunicación es **extremo a extremo**. El proceso origen en el ordenador A genera un mensaje, que la red envía al proceso destino en el ordenador B sin que ningún dispositivo lo cambie durante la transmisión.

DIRECCIONAMIENTO.

Un concepto muy importante relacionado con el modelo en capas de Internet es el **direccionamiento**. Como ya hemos discutido antes, en la pila TCP/IP existe una comunicación lógica entre capas iguales. Ahora, cualquier comunicación entre dos partes necesita dos direcciones: La dirección de origen y la dirección de destino. Aunque podríamos pensar que necesitamos cinco parejas de direcciones (una por cada pareja de capas iguales), de hecho tenemos solo cuatro, porque la capa física no necesita direcciones. (En efecto, la unidad de datos que intercambian dos capas físicas es el bit, el cual obviamente no puede incluir una dirección). La figura 2.12 muestra la unidad de datos intercambiada y el direccionamiento empleado en cada capa del modelo TCP/IP.

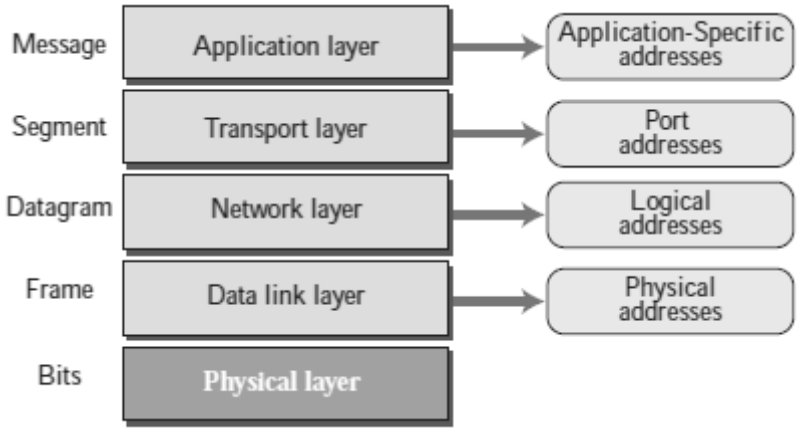


Figura 2.12. Direccionamiento en TCP/IP.

Direcciones físicas.

Ya hemos visto que, en su viaje desde el host origen al host destino, los paquetes podrían tener que pasar por varios enlaces (varias redes LAN y/o WAN). Las direcciones IP permiten identificar de forma unívoca los host origen y destino de los paquetes, y también les sirven a los rúters para decidir cuál es la siguiente red de la ruta hacia el destino. Sin embargo, después de que el rúter haya elegido la siguiente red de la ruta, la capa de enlace utiliza las direcciones físicas de los dispositivos conectados a la red para mover los paquetes dentro de esa red. Por consiguiente, las **direcciones físicas** (también denominadas **direcciones de capa de enlace** o **direcciones MAC**) son las direcciones que usa la capa de enlace para identificar a los distintos dispositivos conectados a una cierta red, y para mover los paquetes desde el nodo de entrada hasta el nodo de salida de esa red. Estas direcciones se incluyen en las tramas que viajan por una cierta

red para definir a los nodos emisor y receptor dentro de esa red. Las direcciones físicas solo tienen vigencia dentro de cada red individual. El tamaño y el formato de estas direcciones dependen de la tecnología de red usada en la red en cuestión. Por ejemplo, las redes **LAN Ethernet** utilizan direcciones físicas estáticas de 6 bytes (48 bits) que están grabadas en la circuitería de las **tarjetas de interfaz de red** (**NIC** = Network Interface Card) de los nodos del enlace. Otros protocolos, como AppleTalk (Apple), usan direcciones dinámicas de 1 byte que se asignan de forma automática.

La figura 2.13 muestra un ejemplo de red LAN en la que un nodo con dirección física 10 envía un paquete a otro nodo con dirección física 87. La capa de enlace del nodo emisor recibe los datos de la capa superior (capa de red), y los encapsula en una trama añadiéndoles un encabezado y un finalizador. El **encabezado** incluye, entre otros datos de control, las direcciones físicas de los nodos origen y destino. El **finalizador** suele contener bits adicionales para la detección de errores. Notar que, en la mayoría de protocolos de capa de enlace, la dirección de destino (87 en este caso) viene antes que la dirección de origen (10). La trama se difunde a través de la LAN a todas las estaciones de la red. Las estaciones con una dirección física distinta de la 87 descartan la trama recibida, porque la dirección física de destino no coincide con su propia dirección física. Sin embargo, el ordenador destinatario encontrará una coincidencia entre la dirección física de destino de la trama y su propia dirección física. Ese ordenador copiará la trama, comprobará errores, desencapsulará los datos, y se los pasará a las capas superiores.

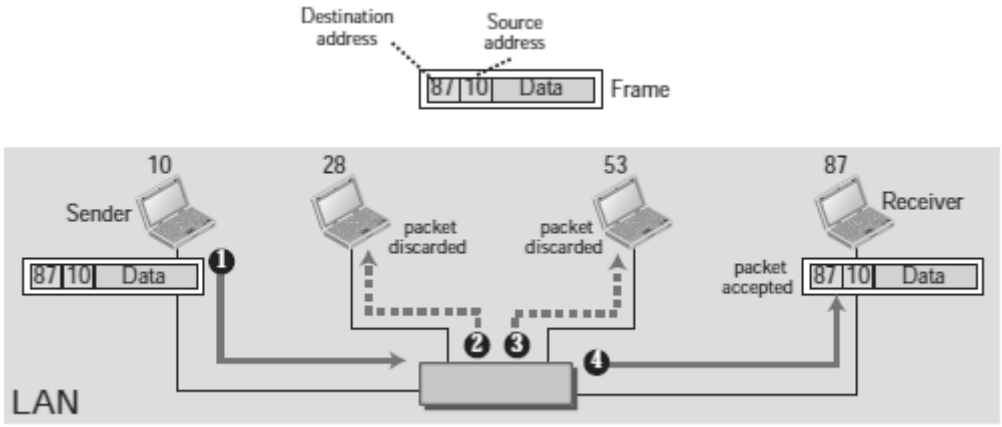


Figura 2.13. Direcciones físicas.

Direcciones lógicas.

Las direcciones físicas no son adecuadas para permitir las comunicaciones en una interred compuesta por múltiples redes que usen diferentes formatos para las direcciones físicas. Por consiguiente, se necesita un sistema de direccionamiento universal en el que cada host de una interred pueda identificarse de forma unívoca, independientemente de las tecnologías de las redes subyacentes. Las **direcciones lógicas**, o direcciones de capa de red, proporcionan este esquema de direccionamiento.

En Internet, las direcciones lógicas se denominan **direcciones IP**, y consisten en direcciones de 4 bytes (32 bits) que le permiten a la **capa de red** identificar de forma conjunta a un host conectado a Internet, y a la red a la que pertenece. Los protocolos de TCP/IP no permiten que dos hosts visibles en Internet puedan tener la misma dirección IP.

La figura 2.14 muestra una parte de una interred con dos rúters que conectan tres LANs. Cada host y cada rúter de la interred tienen asignadas dos direcciones (una lógica y una física) a cada una de sus conexiones. Los hosts solo tienen una conexión, por lo que únicamente disponen de una dirección lógica y una dirección física. Pero en este ejemplo los rúters están conectados a tres enlaces, y necesitan tres parejas de direcciones, esto es, tres direcciones lógicas y tres direcciones físicas.

Supongamos que el ordenador con dirección lógica A y dirección física 10 necesita enviar un paquete al ordenador con dirección lógica P y dirección física 95. El ordenador de origen encapsula los datos en un datagrama de capa de red, al que añade las direcciones lógicas de origen y destino (A y P, respectivamente). Notar que, en la mayoría de protocolos de capa de red, la dirección lógica de origen viene antes que la dirección lógica de destino (esto es, al contrario que las direcciones físicas). Sin embargo, la capa de red del ordenador de origen necesita determinar en primer lugar la dirección física del primer **salto** (hop) de la ruta, esto es, la dirección del rúter 1. Para ello, la capa de red consulta su **tabla de encaminamiento**, y obtiene que la dirección lógica del rúter 1 es F. A continuación, otro protocolo llamado **ARP** (**Address Resolution Protocol** = Protocolo de Resolución de Direcciones) utiliza la dirección lógica del rúter 1 (F) para determinar su dirección física (20). Ahora, la capa de red puede pasar esta dirección a la capa de enlace, que encapsula el paquete con la dirección física de destino (20) y con la dirección física de origen (10).

Esta trama se difunde a todos los dispositivos de la LAN 1, pero la descartan todos excepto el rúter 1, que es el único que encuentra una coincidencia entre la dirección física de destino de la trama y su propia dirección física. El rúter 1 desencapsula el paquete fuera de la trama y lee la dirección lógica de destino, que es P. Como esta dirección lógica no coincide con su dirección lógica (F), el rúter 1 sabe que debe reenviar el paquete. El rúter 1 consulta su tabla de encaminamiento, y usa el protocolo ARP para hallar la dirección física del siguiente salto (que es el rúter 2). A continuación, crea una nueva trama donde encapsula el paquete, y se la envía al rúter 2. Prestar atención a las direcciones físicas de esta nueva trama. La dirección física del emisor cambia de 10 (ordenador emisor) a 99 (rúter 1), y la dirección física del receptor cambia de 20 (rúter 1) a 33 (rúter 2). Pero las direcciones lógicas de origen y de destino siguen siendo las mismas; en caso contrario, el paquete nunca llegaría a su destino final.

En el rúter 2 tenemos un escenario similar. El rúter 2 cambia las direcciones físicas, y crea una nueva trama que envía al ordenador de destino. Cuando la trama llega al destino, el ordenador desencapsula el paquete fuera de la trama. Como la dirección lógica de destino (P) coincide con la dirección lógica del ordenador, la capa de red del ordenador desencapsula los datos y los entrega a las capas superiores. Notar que, aunque las direcciones físicas cambian al pasar de un enlace a otro, las direcciones lógicas son las mismas durante todo el trayecto de origen a destino.

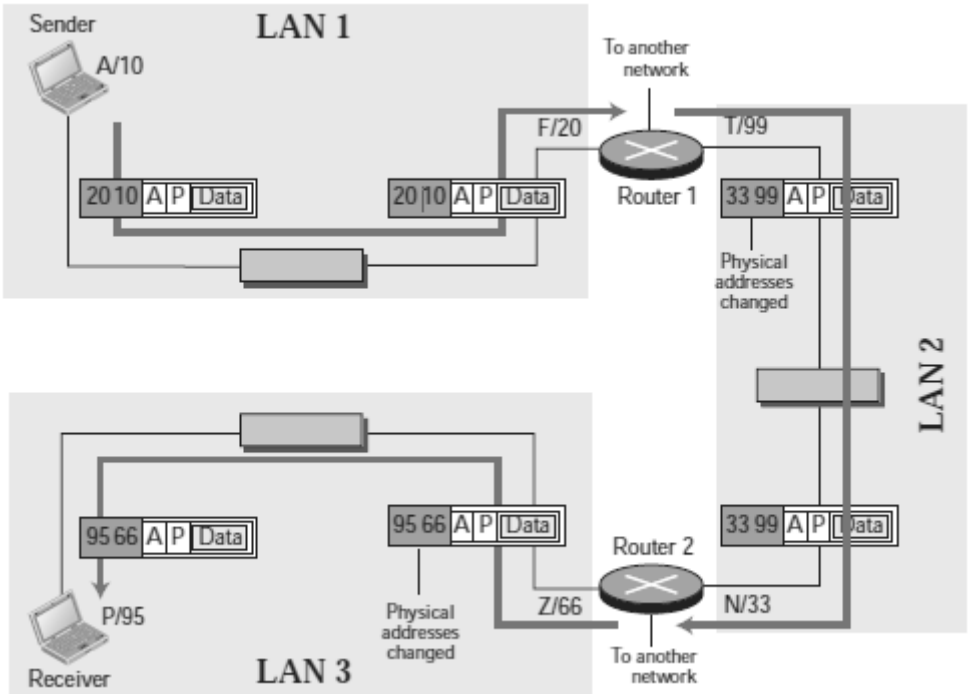


Figura 2.14. Direcciones lógicas.

Números de puerto.

Las direcciones IP y las direcciones físicas son necesarias para que los datos viajen desde el host origen hasta el host destino. Sin embargo, el objetivo final de los datos no es solo llegar al host destino. Hoy día, los ordenadores son dispositivos multitarea, y pueden ejecutar múltiples procesos al mismo tiempo. Por consiguiente, el objetivo final de las comunicaciones a través de Internet es que un proceso en el host origen se comunique con otro proceso en el host destino. Por ejemplo, el ordenador A puede estar comunicándose con el ordenador C mediante FTP. Al mismo tiempo, el ordenador A puede estar comunicándose con el ordenador B mediante HTTP. Para que los procesos en A puedan recibir datos simultáneamente, necesitamos un mecanismo para identificar a los diferentes procesos. En TCP/IP, la etiqueta que usa la **capa de transporte** para identificar a los distintos procesos que están ejecutándose en una máquina es el **número de puerto**.

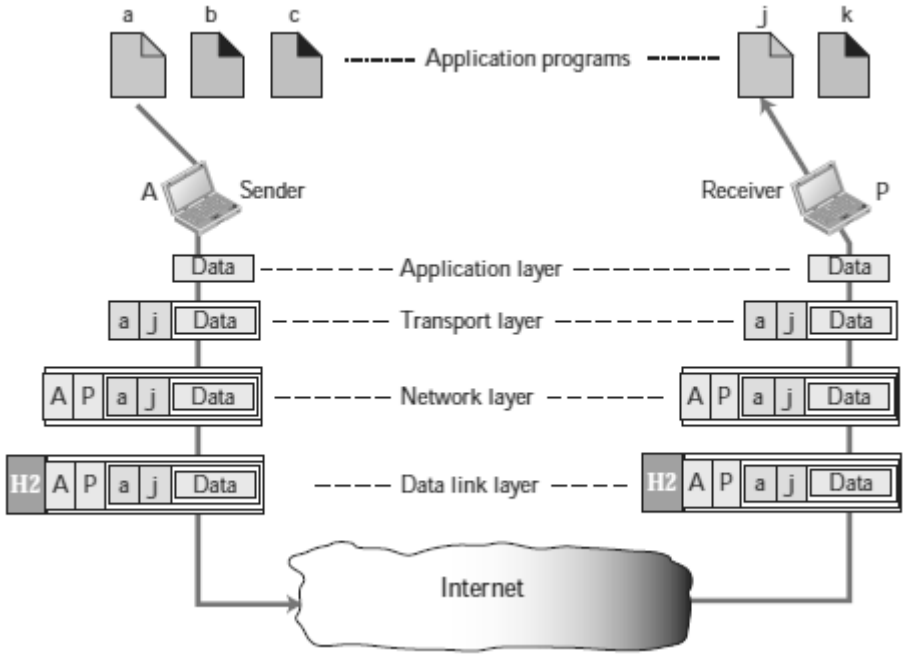


Figura 2.15. Puertos.

La figura 2.15 muestra dos ordenadores comunicándose a través de Internet. En el instante mostrado, el ordenador de origen está ejecutando tres procesos, con números de puerto a, b, y c. El ordenador de destino está ejecutando dos procesos, identificados con los números de puerto j y k. Supongamos que el proceso a en el ordenador origen desea comunicarse con el proceso j en el ordenador destino. Aunque ambos ordenadores estén usando la misma aplicación (digamos, FTP), los números de puerto son distintos porque uno es un proceso cliente y el otro es un proceso servidor. Para indicar que los datos del proceso a deben entregarse al proceso j, y no al k, la capa de transporte encapsula los datos de la capa de aplicación en un paquete, al que añade los números de puerto del proceso origen y del proceso destino (a y j, respectivamente). A continuación, el paquete de la capa de transporte se encapsula en otro paquete en la capa de red, que incluye las direcciones lógicas de origen y de destino (A y P). Finalmente, este paquete se encapsula en una trama con las direcciones físicas de los nodos emisor y receptor para el siguiente enlace. La figura no muestra las direcciones físicas porque cambian de enlace a enlace dentro de la nube que representa a Internet. Notar que, aunque las direcciones físicas cambian en cada enlace de la ruta, las direcciones lógicas y los números de puerto permanecen inalterados durante todo el trayecto entre origen y destino.

Direcciones específicas de las aplicaciones.

Como ya hemos indicado, el destino final de los datos viene definido por una dirección lógica (que identifica de forma unívoca al host de destino dentro de Internet) y por un número de puerto (que identifica al

proceso receptor de los datos en el host de destino). Un ejemplo de este tipo de direccionamiento sería 200.23.56.8 y 69, donde 200.23.56.8 es la dirección IP del host destino, y 69 el número de puerto del proceso destino.[13] Por supuesto, esta forma de identificar un destino es poco amigable para los usuarios, razón por la que algunas aplicaciones utilizan direcciones más intuitivas que están diseñadas para esa aplicación específica. Algunos ejemplos son las direcciones de correo electrónico (por ejemplo, alexmartinezpadilla@yahoo.es) y las **URL** (Uniform Resource Locator = Localizador Uniforme de Recursos), como por ejemplo www.facebook.com. La primera identifica al destinatario de un correo electrónico; la segunda se usa para encontrar una página en la World Wide Web. Sin embargo, el ordenador emisor debe traducir estas direcciones específicas a las direcciones lógicas y a los números de puerto correspondientes.

En resumen, las redes que usan los protocolos TCP/IP utilizan cuatro tipos de direcciones: Las direcciones físicas, las direcciones lógicas de interred (direcciones IP), los números de puerto, y las direcciones específicas de las aplicaciones. Las direcciones físicas, también llamadas direcciones de capa de enlace, permiten identificar a los distintos nodos de una red LAN o WAN. Las direcciones IP identifican de forma conjunta y unívoca tanto a los hosts de Internet como a las redes a las que pertenecen. Los números de puerto identifican los distintos procesos de un host. Y las direcciones específicas se usan en algunas aplicaciones para proporcionar un acceso más intuitivo.

ENCAPSULADO Y DESENCAPSULADO.

Uno de los conceptos más importantes de la arquitectura en capas es el **encapsulado** y **desencapsulado** de los datos. La figura 2.16 ilustra este concepto en el contexto de una interred compuesta por dos enlaces (con sus correspondientes conmutadores) y un rúter. La figura muestra el encapsulado en el host de origen, el desencapsulado en el host de destino, y el encapsulado y desencapsulado en el rúter. La figura no muestra las capas de los conmutadores, porque estos dispositivos no cambian los paquetes de datos, y por consiguiente, no realizan encapsulado ni desencapsulado de los datos.

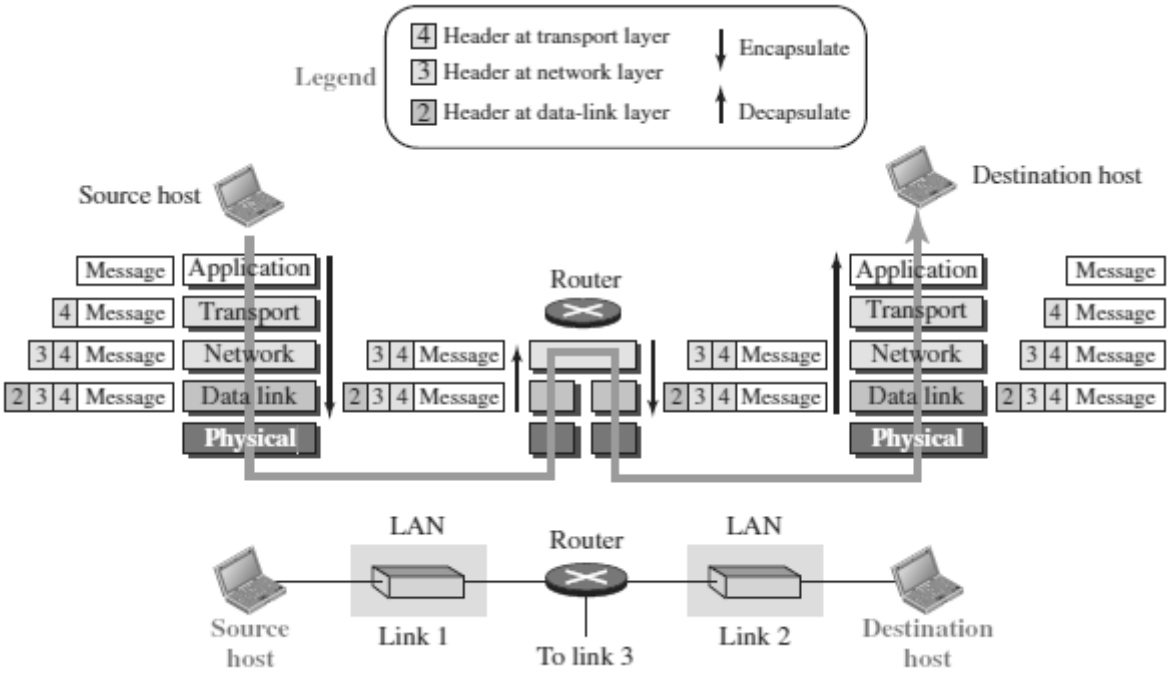


Figura 2.16. Encapsulado y desencapsulado.

[13] Por cierto, que a la combinación de la dirección IP y del número de puerto se le denomina **dirección de socket**.

Encapsulado en el host origen.

En el host origen, solo hay encapsulado:

1) En la capa de aplicación, a los datos a intercambiar con la capa de aplicación del host remoto se les denomina **mensajes**. Los mensajes no suelen incluir ni encabezado ni terminador, pero de tenerlos, les seguiríamos llamando mensajes. Los mensajes generados por la capa de aplicación se envían a la capa de transporte.
2) La capa de transporte recibe un mensaje como carga útil, y le añade un encabezado que contiene los identificadores (números de puerto) de los procesos origen y destino que desean comunicarse, más algunas informaciones adicionales para garantizar la entrega extremo a extremo del mensaje, como información para el control de flujo, para el control de errores, y para el control de la congestión. El resultado es un paquete de capa de transporte, al que se denomina **segmento** (en TCP) o **datagrama de usuario** (en UDP). La capa de transporte le pasa el paquete resultante a la capa de red.
3) La capa de red recibe el paquete de la capa de transporte como carga útil, y le añade su propio encabezado. El encabezado contiene las direcciones lógicas (direcciones IP) de los hosts origen y destino, más alguna información adicional para el control de errores en el encabezado, fragmentación de la información, etc. El resultado es un paquete de capa de red llamado **datagrama**. A continuación, la capa de red le pasa este paquete a la capa de enlace.
4) La capa de enlace recibe el paquete de capa de red como carga útil y le añade su propio encabezado, que incluye las direcciones de capa de enlace del nodo emisor (el propio host de origen) y del nodo receptor (el rúter que conecta con el siguiente enlace). La capa de enlace también añade un terminador (no mostrado en la figura) que incluye bits para la detección de errores. El resultado es un paquete de capa de enlace al que se denomina **trama**. La trama se le pasa a la capa física para transmitirla como un flujo de **bits** en serie.

Conmutador del primer enlace.

Como hemos mencionado antes, los conmutadores no encapsulan ni desencapsulan las tramas que reciben. Cuando una trama llega a la capa de enlace del conmutador, ésta simplemente examina las direcciones físicas del nodo emisor (el host de origen) y del nodo receptor (el rúter) para mover la trama a través del enlace. (Notar que los conmutadores no cambian los paquetes de datos. En concreto, los conmutadores no modifican las direcciones, ni fragmentan las tramas, ni adaptan el formato de los paquetes a otro protocolo, etc.).

Encapsulado y desencapsulado en el rúter.

En el rúter tenemos tanto encapsulado como desencapsulado, porque el rúter interconecta dos o más enlaces:

1) Después de que los bits lleguen a la capa de enlace del rúter, esta capa reconstruye la trama, desencapsula el datagrama contenido, y se lo pasa a la capa de red. (Notar que en este paso el rúter elimina las direcciones de capa de enlace correspondientes al primer enlace). La capa de red solo inspecciona las direcciones IP de origen y de destino del encabezado del datagrama, y consulta su tabla de encaminamiento para determinar el siguiente enlace al que se debe enviar el datagrama. La capa de red del rúter no altera los datos contenidos por el datagrama (en concreto, no cambia las direcciones IP de origen y destino), a menos que necesite fragmentarlo en el caso de que sea demasiado grande para pasárselo al siguiente enlace. A continuación, la capa de red pasa el datagrama a la capa de enlace del siguiente enlace.
2) La capa de enlace del siguiente enlace encapsula el datagrama en una trama. Notar que, en este paso, el rúter añade al encabezado de la trama las nuevas direcciones de capa de enlace del nodo emisor (el

rúter) y del nodo receptor (el host de destino). A continuación, le pasa la trama resultante a la capa física para su transmisión.

Conmutador del segundo enlace.

De nuevo, el conmutador del segundo enlace no encapsula ni desencapsula los datos, y únicamente examina las direcciones de capa de enlace del nodo emisor (el rúter) y del nodo receptor (el host de destino) para moverlas a través de este enlace.

Desencapsulado en el host de destino.

En el host de destino, cada capa solo desencapsula el paquete recibido, le extrae la carga útil, y se la pasa al protocolo de la capa inmediatamente superior hasta que el mensaje llega a la capa de aplicación. El proceso de desencapsulado en el host destino también implica la comprobación de posibles errores.

EJEMPLO DE COMUNICACIÓN.

Para ilustrar y poner en contexto todos los conceptos estudiados a lo largo de este capítulo, consideremos el siguiente ejemplo de comunicación: Un navegador web (un cliente web) en un host con nombre "pc1.ies.edu" (de ahora en adelante, simplemente "pc1") establece una conexión con un servidor web ubicado en un host con nombre "webserver.empresa.com" (al que llamaremos sencillamente "webserver"). Ambos hosts están conectados a Internet. El "pc1" solicita acceder a la página web raíz del servidor "webserver", cuya URL es http://webserver.empresa.com/index.html.

En este ejemplo vamos a explorar qué ocurre en la red cuando el host cliente realiza una solicitud de acceso web al host servidor. Analizaremos los pasos que los distintos protocolos deben efectuar hasta que el primer paquete enviado por el host cliente llega al host servidor. (La tabla al final de la sección muestra un resumen de todos los pasos involucrados en esta comunicación).

Protocolo de aplicación HTTP.

Los procesos cliente y servidor son dos programas en ejecución que se comunican entre sí usando el **protocolo de aplicación HTTP** (HTTP = HyperText Transfer Protocol). HTTP es un protocolo de tipo cliente – servidor: El host cliente ejecuta un **proceso cliente HTTP**, y el host servidor ejecuta un **proceso servidor HTTP**. Imaginemos que el usuario del ordenador "pc1" escribe la URL de la página que desea consultar en la barra de direcciones de su navegador web. Esta acción implica que el proceso cliente envíe un mensaje de solicitud "HTTP Request" con la URL de la página web al proceso servidor. En el momento en el que encuentra la página, el proceso servidor responde con un mensaje "HTTP Reply" al proceso cliente, que incuye los contenidos de la página web solicitada. Cuando el proceso cliente recibe este mensaje, le pasa los contenidos al navegador web para que muestre por pantalla la página web recibida.

Protocolo de transporte TCP.

Para enviar los datos entre los procesos cliente y servidor, el protocolo de aplicación HTTP usa los servicios del **protocolo de transporte TCP** (TCP = Transmission Control Protocol). Recordemos que TCP es un **protocolo orientado a conexión**. Esto implica que, cuando el proceso cliente en el host "pc1" quiere enviar un mensaje "HTTP Request", primero debe establecer una conexión TCP con el proceso servidor en el host "webserver".

Pero antes de que el cliente HTTP en "pc1" pueda pedirle al protocolo TCP que establezca una conexión, debe resolver un asunto relacionado con el direccionamiento: El protocolo TCP espera que las direcciones estén escritas en términos de una **dirección IP** y de un **número de puerto**. La dirección IP es un

identificador de 32 bits que define de forma unívoca tanto al host como a la red a la que pertenece. Por ejemplo, la dirección IP del host "pc1" es 128.143.137.144. Notar que cada byte (esto es, cada grupo de 8 bits) de la dirección IP se escribe como un número decimal, que puede tomar un valor cualquiera entre 0 y 255. Los cuatro números decimales separados por puntos constituyen la forma más habitual de representar una dirección IP. Por su parte, el número de puerto es un número de 16 bits (lo que permite $2^{16} = 65536$ números de puerto posibles) que sirve para identificar un proceso en ejecución en una máquina. Conjuntamente, la dirección IP y el número de puerto (combinación a la que a veces se denomina **dirección de socket**) permiten identificar unívocamente una aplicación en Internet.

Protocolo DNS.

Veamos la forma en la que el cliente HTTP en "pc1" obtiene la dirección IP y el número de puerto del servidor HTTP en "webserver". La dirección IP se obtiene usando un **servicio global de traducción de direcciones** llamado **DNS** (DNS = Domain Name System), que se encarga de traducir **nombres de dominio** simbólicos (esto es, direcciones específicas como "webserver.empresa.com") a direcciones IP, y viceversa. Para que un host pueda llamar al servivio DNS, debe enviar una solicitud a su **servidor DNS preferido**. La localización del servidor DNS preferido de un host forma parte de la configuración IP de ese host (ver práctica B). Dejando de lado los detalles de esta configuración, "pc1" envía una consulta a su servidor DNS sobre el nombre de dominio "webserver.empresa.com", y obtiene como respuesta la dirección IP de ese host, que resulta ser 128.143.71.21.

Puertos bien conocidos.

Una vez conocida la dirección IP del host servidor, determinar el número de puerto del servidor HTTP es fácil: En Internet, los procesos servidores de las aplicaciones más comunes tienen números de puerto previamente asignados, a los que se denomina **números de puerto bien conocidos**. Estos puertos bien conocidos están almacenados en un archivo de configuración de los hosts. El puerto bien conocido de un servidor HTTP siempre es el 80. Cuando el cliente HTTP en "pc1" contacta con el servidor HTTP en "webserver", asume que este servidor web siempre estará disponible a través del puerto 80.

Establecimiento de la conexión TCP.

Ahora, el cliente HTTP en "pc1" dispone de la información necesaria para establecer una conexión TCP con el servidor HTTP en "webserver", y solicita una conexión TCP con el puerto 80 en la dirección IP 128.143.71.21. Al igual que HTTP, el protocolo TCP también utiliza un modelo cliente – servidor. La parte que inicia la conexión se denomina **cliente TCP**, y la parte que permanece a la espera de recibir solicitudes de conexión se llama **servidor TCP**. Cuando un servidor web arranca, el proceso servidor establece un servidor TCP que queda a la espera (a través del puerto 80) de recibir solicitudes de conexión TCP. No vamos a discutir los detalles del proceso para establecer una conexión TCP, y solo mencionaremos que consta de tres pasos: (1) El cliente TCP envía una solicitud de conexión al servidor TCP. (2) El servidor TCP responde a esta solicitud. (3) El cliente TCP confirma esta respuesta y comienza a enviar datos. Aquí solo vamos a analizar el primer paso del establecimiento de una conexión TCP, en el que el protocolo TCP del host cliente envía una solicitud de conexión al puerto 80 de la dirección IP 128.143.71.21. Es importante puntualizar que esta solicitud de conexión no contiene datos. El mensaje de solicitud HTTP con la URL de la página web se enviará en el tercer paso del procedimiento de conexión (la fase de transferencia de datos a través de la tubería TCP previamente establecia).

Protocolo de red IP.

Ahora vamos a discutir los pasos involucrados en la entrega del mensaje de solicitud de conexión TCP desde "pc1" a "webserver". En primer lugar, el cliente TCP en "pc1" le pide a su **protocolo de red IP** (IP = Internet Protocol) que entregue la solicitud de conexión TCP a la dirección IP 128.143.71.21. El protocolo

IP toma esa solicitud de conexión, la encapsula en un **datagrama IP** (recordemos que a los paquetes de la capa de red se les denominaa **datagramas**), y envía ese datagrama con destino al host "webserver".

La red ilustrada en la figura 2.17 muestra un esquema global de los dispositivos involucrados en la entrega del datagrama IP desde el host "pc1" al host "webserver". La figura muestra que el host "pc1" y el host "webserver" están conectados cada uno a una LAN Ethernet distinta. La figura también muestra que la ruta del datagrama entre "pc1" y "webserver" pasa a través de un **rúter** que interconecta ambas LANs. Recordemos que un rúter es un dispositivo de conexión que permite interconectar redes. Los rúters disponen de tantos interfaces como redes interconectan, cada uno con su propio nombre de dominio y dirección IP. Los rúters se encargan de tomar los datagramas IP que le llegan a través de alguno de sus interfaces de entrada para reenviarlos a través de alguno de sus interfaces de salida, con objeto de encaminar el datagrama hacia su detino final. Para decidir a qué interfaz de salida debe reenviar cada datagrama, los rúters consultan una **tabla de encaminamiento** que indica a qué interfaz de salida encaminar ciertas direcciones IP de destino.

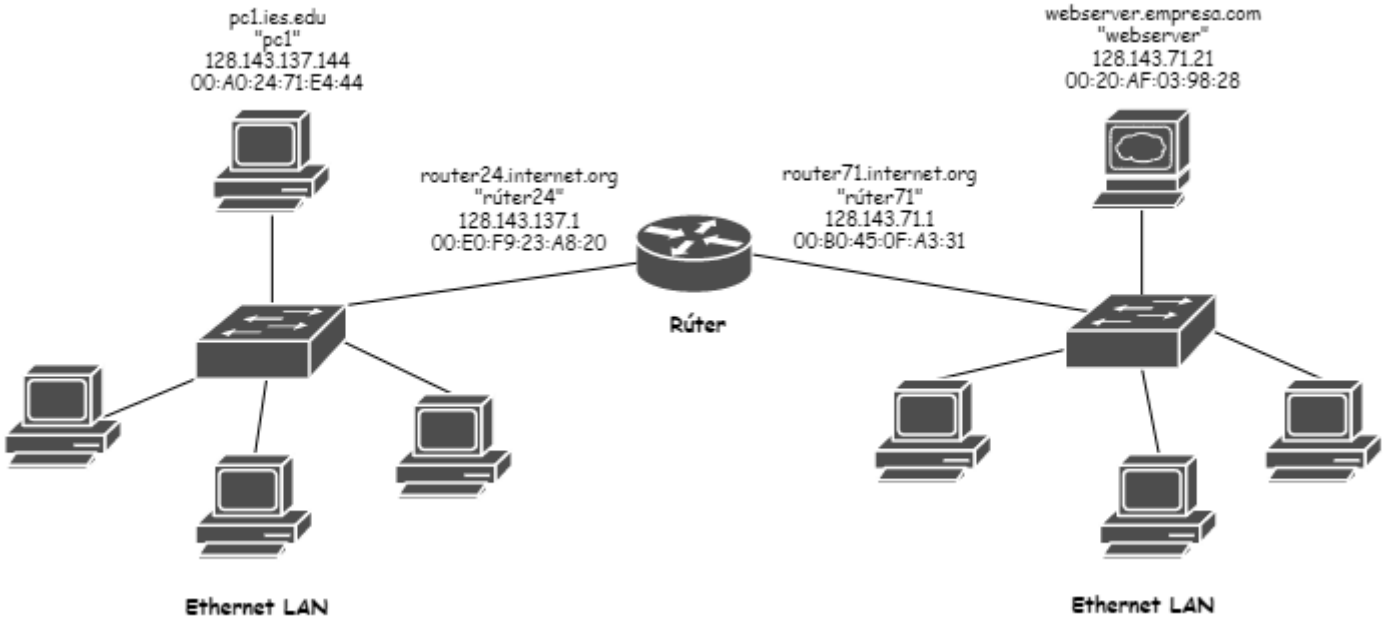


Figura 2.17. La interred del ejemplo de comunicación.

Cuando "pc1" dispone del datagrama IP que debe enviar a la dirección IP 128.143.71.21 (el host "webserver"), desconoce la ruta que deberá tomar el datagrama para llegar a su destino final. Analizando la dirección de destino, el protocolo IP del host "pc1" sabe si puede entregar ese datagrama directamente a un host de su propia red, o si debe reenviarlo a un rúter para encaminarlo hacia otra red. Para determinar si el host de destino pertenece a su misma red, el host "pc1" compara la dirección IP de "webserver" (128.143.71.21) con su propia dirección IP (128.143.137.144). En este caso, "pc1" verifica que el host "webserver" no pertenece a su misma red, y reenvía el datagrama a su **puerta de enlace predeterminada**. La puerta de enlace predeterminada de un host es la dirección IP del rúter que le conecta a Internet, y forma parte de la configuración IP del host en cuestión. La puerta de enlace de "pc1" es 128.143.137.1, un rúter con nombre de dominio "router24.internet.org" (a partir de ahora, simplemente "rúter24"). Por consiguiente, "pc1" reenvía el datagrama a "rúter24".

Protocolo de enlace Ethernet.

Ahora, para enviar el datagrama a la puerta de enlace 128.143.137.1, el host "pc1" le pasa el datagrama IP al driver Ethernet de su **tarjeta de interfaz de red** (**NIC** = Network Interface Card). Este driver implementa la **capa de enlace de datos** del host "pc1", y se encarga de insertar el datagrama IP en una **trama Ethernet** (recordemos que a los paquetes de la capa de enlace de datos se les denamina **tramas**), y de transmitir esa trama a través de la red Ethernet a la que está conectado "pc1". Sin embargo, las tramas

Ethernet no usan direcciones IP, sino direcciones de 48 bits llamadas **direcciones físicas**. Las direcciones físicas suelen escribirse en notación hexadecimal, donde cada byte está saparado del siguiente por dos puntos. (Por ejemplo, la dirección física de "pc1" es 00:A0:24:71:E4:44). Cada trama Ethernet contiene las direcciones físicas de los nodos origen y destino de la trama en esa red. Pero antes de que el protocolo IP pueda pasarle el datagrama a la tarjeta de red (a la capa de enlace), primero debe determinar la dirección física del nodo destino (el rúter) a partir de su dirección IP (128.143.137.1). La traducción de direcciones IP a direcciones físicas es tarea del **protocolo de traducción de direcciones ARP** (ARP = Address Resolution Protocol).

Protocolo ARP.

¿Cómo opera el protocolo ARP? En primer lugar, el nodo "pc1" envía un mensaje ARP de la forma "¿Cuál es la dirección física asociada a la dirección IP 128.143.137.1?". Este mensaje ARP se difunde a los intefaces de red de todos los dispositivos conectados a la red LAN de "pc1". Cuando el nodo "rúter24" (cuya dirección IP es 128.143.137.1) recibe este mensaje ARP, responde con otro mensaje ARP que dice "La dirección IP 128.143.137.1 está asociada a la dirección física 00:E0:F9:23:A8:20". Finalmente, cuando el "pc1" recibe este mensaje, le pasa al driver Ethernet de su tarjeta de red la dirección física de la puerta de enlace junto con el datagrama a enviar, y transmite la trama. Notar que se transmiten tres tramas Ethernet en total. Notar también que las subsiguientes transmisines de datagramas IP desde el "pc1" al "rúter24" no necesitan intercambiar más mensajes ARP: El "pc1" mantiene una lista de las direcciones físicas ya conocidas en una tabla ARP local.

Entrega del mensaje de solicitud de conexión TCP.

Posteriormente, cuando recibe la trama Ethernet del "pc1", el "rúter24" extrae el datagrama IP y se lo pasa a su capa de red. Una vez analizada la dirección IP de destino del datagrama recibido, el rúter toma una decisión similar a la que tomó el "pc1": El rúter determina si debe reenviar el datagrama directamente el host "webserver", o consulta su tabla de encaminamiento para seleccionar el siguiente rúter de la ruta hasta el host "webserver". Como el "rúter24" dispone de múltiples interfaces de salida hacia varias redes, debe comprobar si el destino final del datagrama IP está directamente accesible a través de alguno de sus interfaces de salida. Para determinar si un host está directamente accesible, el rúter compara la dirección IP de destino final (la dirección IP de "webserver", 128.143.71.21) con las direcciones IP de todos sus interfaces de salida (128.143.137.1 y 128.143.71.1). En este caso, el rúter detecta una coincidencia con la dirección IP 128.143.71.1, y decide reenviar el datagrama IP directamente a "webserver" a través de ese interfaz de salida.

La transmisión del datagrama IP desde el rúter hasta el host "webserver" pasa por los mismos pasos que la entrega del datagrama IP desde el host "pc1" hasta el rúter: En primer lugar, el rúter obtiene la dirección física de "webserver" enviando un mensaje de solicitud ARP a la red local Ethernet conectada a su interfaz de salida 128.143.71.1. El host "webserver" recibirá este mensaje, y responderá enviando su dirección física, 00:20:AF:03:98:28, en un mensaje de respuesta ARP. Tan pronto como el rúter recibe la dirección física de "webserver", envía una trama Ethernet que contiene el datagrama IP con destino a "webserver".

Cuando "webserver" recibe la trama Ethernet del "rúter24", primero extrae el datagrama IP, y a continuación, entrega el paquete TCP al puerto 80. En este momento se completa la entrega del primer datagrama IP entre "pc1" y "webserver", que recordemos, contiene la solicitud de establecimiento de la conexión TCP. Lo que ocurre a continuación es que el servidor TCP en "webserver" responde a la solicitud de conexión TCP, lo que implica la transmisión de otro datagrama IP desde "webserver" hacia "pc1". Los pasos involucrados en la entrega de este datagrama son los mismos que vimos antes. Cuando el "pc1" recibe esta respuesta, puede comenzar a transmitir el mensaje de solicitud HTTP.

La descripción previa nos da una idea de la complejidad que conlleva transmitir datos entre dos hosts interconectados a través de una interred. La tabla a continuación resume los pasos que hemos detallado:

PASO	DESCRIPCIÓN.
1	El cliente web en "pc1" genera un mensaje "HTTP Request" para acceder a la página web con URL http://webserver.empresa.com/index.html.
2	"pc1" contacta con su servidor DNS preferido para traducir el nombre de dominio "webserver.empresa.com" a una dirección IP.
3	El cliente HTTP en "pc1" solicita una conexión TCP al puerto 80 de la dirección IP 128.143.71.21 (la IP del host "webserver" devuelta por el servidor DNS).
4	El cliente TCP en "pc1" solicita a su protocolo IP que entregue el datagrama IP con la solicitud de conexión TCP al host con dirección IP 128.143.71.21.
5	La capa de red en "pc1" determina que no puede entregar el datagrama IP directamente, y decide enviar el datagrama a su puerta de enlace predeterminada (el rúter), cuya dirección IP es 128.143.137.1.
6	El protocolo ARP en "pc1" envía un mensaje ARP para solicitar la dirección física asociada a la dirección IP 128.143.137.1.
7	La solicitud ARP se difunde a todos los dispositivos de la red LAN Ethernet a la que pertenece "pc1".
8	El rúter con dirección IP 128.143.137.1 responde a esta solicitud con una respuesta ARP que incluye su dirección física, que es 00:E0:F9:23:A8:20.
9	El protocolo IP en "pc1" le pide a su adaptador de red que envíe el datagrama IP en una trama Ethernet a la dirección física 00:E0:F9:23:A8:20.
10	El adaptador de red del rúter con dirección física 00:E0:F9:23:A8:20 extrae el datagrama IP de la trama, y se lo pasa a su capa de red.
11	El protocolo IP del rúter determina que puede entregar el datagrama IP con destino a 128.143.71.21 directamente (esto es, sin necesidad de pasar por otros rúters intermediarios).
12	El protocolo ARP del rúter envía una solicitud ARP para obtener la dirección física asociada a la dirección IP 128.143.71.21.
13	El mensaje de solicitud ARP se difunde a todas las máquinas de la red LAN Ethernet a la que pertenece "webserver".
14	El host "webserver" (cuya dirección IP es 128.143.71.21) responde con un mensaje ARP que incluye su dirección física, a saber, 00:20:AF:03:98:28.
15	El protocolo IP en el rúter le pide a su adaptador de red que envíe el datagrama IP en una trama Ethernet a la dirección física 00:20:AF:03:98:28.
16	El adaptador de red Ethernet de "webserver" extrae el datagrama contenido en la trama Ethernet recibida, y se la pasa a su capa de red.
17	El protocolo IP extrae la solicitud de conexión TCP contenida en el datagrama IP y se la pasa al servidor TCP de "webserver" en el puerto 80.
18	El servidor TCP de "webserver" procesa la solicitud de conexión TCP.
19	El servidor TCP responde a esta solicitud de conexión. Cuando el "pc1" recibe esta respuesta, comienza transmitir el mensaje de solicitud HTTP.

PRÁCTICA D. ENCAPSULADO Y DESENCAPSULADO CON PYTHON.

D.1. PLANTEAMIENTO DEL PROBLEMA.

El objetivo de esta práctica es escribir un programa en Python que encapsule un mensaje escrito por el usuario, y que a continuación lo desencapsule para recuperar el mensaje inicial. La salida del programa debería ser similar a la siguiente:

```
Por favor, escriba el mensaje a enviar (máx. 10 caracteres):
Hola Alex!

Mensaje a enviar: Hola Alex!

Host de origen enviando el mensaje...
Capa de Aplicación: ['Hola Alex!']
Capa de Transporte: ['H4', 'Hola Alex!']
Capa de Red: ['H3', 'H4', 'Hola Alex!']
Capa de Enlace de datos: ['H2', 'H3', 'H4', 'Hola Alex!', 'T2']

Host de destino recibiendo la trama...
Capa de Enlace de datos: ['H2', 'H3', 'H4', 'Hola Alex!', 'T2']
Capa de Red: ['H3', 'H4', 'Hola Alex!']
Capa de Transporte: ['H4', 'Hola Alex!']
Capa de Aplicación: ['Hola Alex!']

Mensaje recibido: Hola Alex!
>>>
```

Notar que `'H4'`, `'H3'`, y `'H2'` son cadenas que representan los encabezados (Headers) añadidos al mensaje por las capas de transporte (4), red (3), y enlace de datos (2), respectivamente. (Recordemos que la capa de aplicación no añade ningún encabezado al mensaje, razón por la que no vemos la cadena `'H5'`). La cadena `'T2'` representa el terminador que añade al mensaje la capa de enlace de datos (2).

Observar que el programa procesa el encapsulado y el desencapsulado utilizando listas. Así, la lista del mensaje encapsulado incluirá como objetos los encabezados de las capas de transporte, red, y enlace de datos; el mensaje inicial; y el terminador de la capa de enlace.

D.2. FUNCIÓN ENCAPSULADORA.

Para empezar, vamos a escribir una función que reciba la cadena del mensaje a enviar, y que la encapsule en una lista con los encabezados `'H4'`, `'H3'`, y `'H2'`, y el terminador `'T2'`. La función devolverá la lista del mensaje encapsulado (esto es, la trama a enviar).

```
def encapsular (mensaje):
    trama = [mensaje] # Convertimos la cadena a lista.
    print ('Capa de aplicación: ' + str(trama))
```

Tu tarea es completar el código de esta función. Usa un bucle `for` para hacer que cada capa añada su encabezado como primer elemento de la lista. Además, la capa de enlace debe añadir su terminador como último elemento de la lista. La función también debe imprimir por pantalla el proceso de encapsulado capa a capa, como muestra la salida del programa.

D.3. FUNCIÓN DESENCAPSULADORA.

A continuación, vamos a escribir una función que reciba la lista con el mensaje encapsulado, y que extraiga capa a capa los encabezados y el terminador hasta obtener el mensaje original. La función debe devolver la cadena del mensaje original.

```
def desencapsular (trama):
    print ('Capa de Enlace de datos: ' + str(trama))
```

De nuevo, tu tarea es completar el código de esta función. Primero, debes eliminar el terminador `'T2'` de capa de enlace, que siempre será el último objeto de la lista. A continuación usa un bucle `for` para eliminar los encabezados de las capas de enlace, de red, y de transporte. (Notar que el siguiente encabezado a eliminar siempre será el primer objeto de la lista). La función también debe imprimir por pantalla el proceso de desencapsulado capa a capa, como muestra la salida del programa.

D.4. PROGRAMA PRINCIPAL.

Por último, escribe el programa principal. En primer lugar, el programa debe pedir al usuario el texto a enviar (cuya longitud no puede superar los 10 caracteres). Si el usuario escribe un texto más largo el programa se lo indica, y le pide de nuevo el texto a enviar.

A continuación, el programa debe llamar a la función encapsuladora, pasándole como parámetro la cadena de texto a enviar. La función devolverá una lista cuyos objetos serán los encabezados, el mensaje, y el terminador; esta lista representa la trama enviada.

Por último, el programa llama a la función encapsuladora, pasándole como parámetro la lista que representa la trama enviada. La función devolverá una cadena con el mensaje original, que debemos imprimir por pantalla.

D.5. IMPLEMENTAR LA CAPA FÍSICA.

A modo de **ampliación**, vamos a implementar la capa física del modelo. Como sabemos, la capa física no introduce ningún tipo de encabezado, y solo se ocupa de transmitir los bits de datos a través del medio físico. En esta práctica programaremos una capa física que se encargue de convertir a bits todas las cadenas de la lista que representa la trama.

Existen diversas formas de codificar cadenas como bits, y una de ellas es el código ASCII, del que ya hablamos en el capítulo 1. Para convertir cadenas en bits en base al código ASCII, podemos usar la clase integrada `Bytes`. Esta clase incluye el método constructor `bytes()`, al que le pasamos como primer argumento la cadena a codificar, y como segundo argumento el esquema de codificación (`'ascii'`, en este caso). Veamos un ejemplo:

```
>>> cadena =  'Alejandro'
>>> cadenaBytes = bytes(cadena, 'ascii')
>>> for byte in cadenaBytes:
     print(byte)

65
108
101
106
97
110
```

```
100
114
111
>>>
```

Como vemos, el método `bytes()` devuelve el código ASCII en decimal de los caracteres de la cadena. Por ejemplo, 65 es el código ASCII decimal del carácter `'A'`, 108 es el código ASCII decimal del carácter `'l'`, etc. Para convertir el código ASCII de cada carácter de decimal a binario, podemos usar la función `bin()`:

```
>>> bin(65)
'0b1000001'
>>> bin(108)
'0b1101100'
```

, donde `0b` indica que la cadena representa un número escrito en binario.

El objetivo de esta sección es escribir una función llamada `codificaASCII()` que reciba la trama (lista) construida por el host de origen, y que convierta cada una de las cadenas que contiene a binario. La función debería recorrer cada una de las cadenas de la trama, y para cada cadena, recorrer todos sus caracteres para convertirlos a binario. Una vez escrita la función, el programa principal debería llamarla después de que el host de origen haya encapsulado en el mensaje hasta la capa de enlace, para convertir la trama en bits. La salida del programa debería ser similar a la siguiente:

```
Por favor, escriba el mensaje a enviar (máx. 10 caracteres):
Hola!

Mensaje a enviar: Hola!

Host de origen enviando mensaje...
Capa de Aplicación: ['Hola!']
Capa de Transporte: ['H4', 'Hola!']
Capa de Red: ['H3', 'H4', 'Hola!']
Capa de Enlace de datos: ['H2', 'H3', 'H4', 'Hola!', 'T2']
Capa física: ['0b10010000b110010', '0b10010000b110011', '0b10010000b110100',
'0b10010000b11011110b11011000b11000010b100001', '0b10101000b110010']

Host de destino recibiendo bits...
Capa física: ['0b10010000b110010', '0b10010000b110011', '0b10010000b110100',
'0b10010000b11011110b11011000b11000010b100001', '0b10101000b110010']
Capa de Enlace de datos: ['H2', 'H3', 'H4', 'Hola!', 'T2']
Capa de Red: ['H3', 'H4', 'Hola!']
Capa de Transporte: ['H4', 'Hola!']
Capa de Aplicación: ['Hola!']

Mensaje recibido: Hola!
>>>
```

PRÁCTICA E: TCP/IP EN LA VENTANA DE COMANDOS DE WINDOWS.

E.1. INTRODUCCIÓN.

En el capítulo 2 presentamos las bases teóricas del proceso global de comunicación entre dos ordenadores conectados a Internet (o a otra interred cualquiera). En esta práctica vamos a poner en contexto algunos de los conceptos introducidos en el capítulo previo, utilizando la consola de comandos de Windows en nuestro ordenador. (Aquí asumiremos que el Sistema Operativo (SO) instalado en nuestro equipo es Windows 10; para otras versiones el proceso debería ser muy parecido).

E.2. LA VENTANA DE COMANDOS DE WINDOWS.

Todos los comandos que usaremos en esta práctica se ejecutan en la ventana de comandos de Windows. El manejo de la ventana de comandos es una habilidad esencial para cualquier administrador de redes. El objetivo de esta sección es proporcionar una breve introducción al uso básico de la ventana de comandos de Windows.

La **ventana de comandos**, también llamada **línea de comandos**, **consola de comandos** o **símbolo del sistema**, es el intérprete de comandos de Windows. Hoy día, la ventana de comandos existe en la forma de un archivo ejecutable, llamado *cmd.exe*, que se localiza en la carpeta C:\Windows\system32. Para abrir la ventana de comandos podemos acudir a la carpeta indicada y pinchar dos veces en el icono del archivo ejecutable. Otra opción más rápida es pulsar las teclas WINDOWS + R para abrir la ventana de ejecución de programas, y escribir `cmd`. La figura E.1 muestra la ventana de comandos una vez abierta.

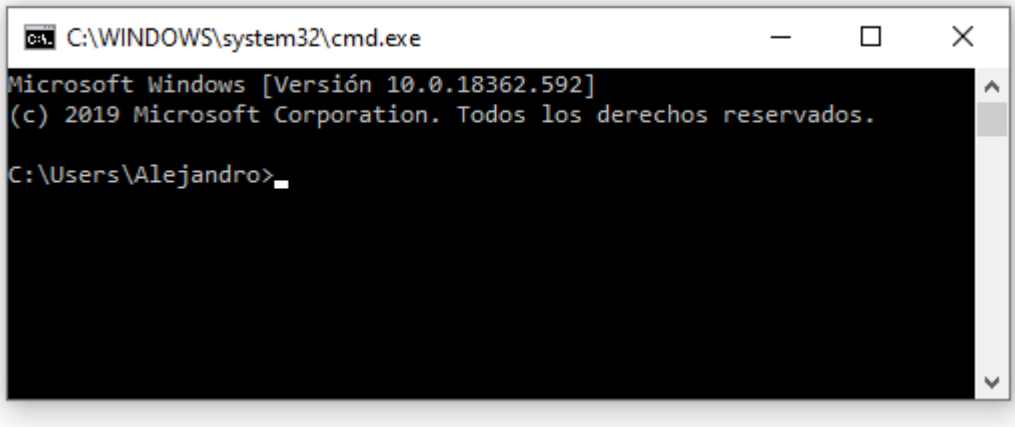


Figura E.1. La ventana de comandos de Windows.

A continuación, vamos a aprender a trabajar con la ventana de comandos.

Para ejecutar un comando, escribimos el nombre del comando y pulsamos ENTER. Por ejemplo, escribimos el comando `cd\` y pulsamos ENTER. El **prompt** de la ventana de comandos debería cambiar de C:\Users\Alejandro> a C:\>, sin ninguna carpeta adicional.[14] Esto puede resultar útil cuando necesitemos ejecutar comandos muy largos, porque así el prompt ocupa menos espacio. A continuación, podemos volver a usar el comando `cd` para cambiar a otra carpeta: Por ejemplo, si escribimos `cd C:\Windows\System32`, haremos que el prompt cambie de la carpeta raíz C:\> a la carpeta C:\Windows\System32>. Con el prompt

[14] Se llama **prompt** al carácter o conjunto de caracteres que se muestran en una línea de comandos para indicar que está a la espera de órdenes.

ubicado en la carpeta que nos interese, podemos ejecutar el comando `dir` para mostrar los contenidos (archivos y subcarpetas) dentro de esa carpeta.

A continuación ejecutamos el comando `cls`. Este comando sirve para limpiar la pantalla y todo el historial de comandos. Sin embargo, todavía podemos volver a traer los comandos que escribimos previamente pulsando las teclas de las flechas arriba y abajo, o usando F3 y F7.

Ahora, ejecutamos el comando `cls /?`. Con esto mostramos el archivo de ayuda del comando `cls`, que nos explica para qué sirve el comando. Este archivo de ayuda contiene muy poca información, pero los comandos más complejos que `cls` tienen archivos de ayuda mucho más extensos. A continuación, ejecutamos el comando `dir /?` para obtener el archivo de ayuda de este comando, que incluye mucho más contenido que el del comando `cls` (ver figura E.2). En general, podemos usar la opción `/?` con cualquier comando para mostrar la información de uso de dicho comando.

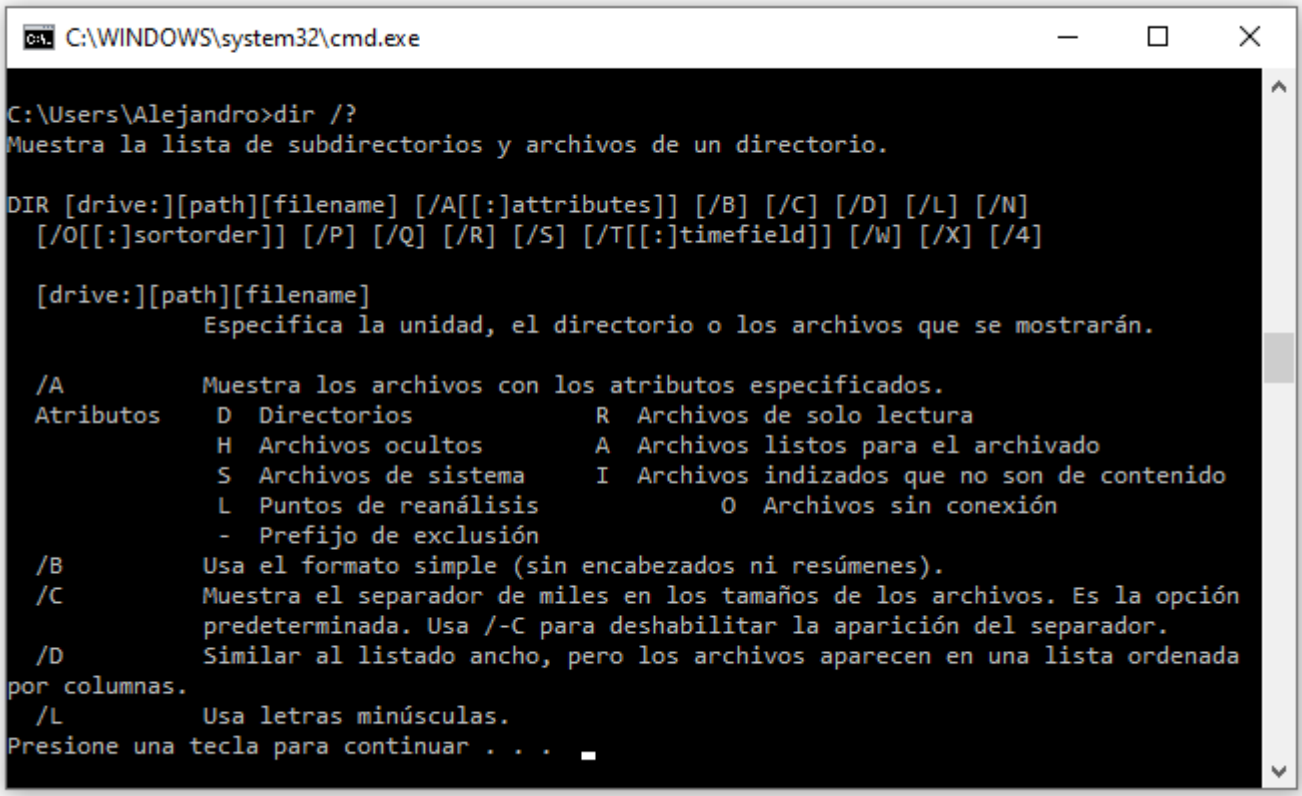


Figura E.2. Resultado del comando `dir /?`.

A veces, el archivo de ayuda o los resultados de un comando serán demasiado extensos como para caber en una sola pantalla de la ventana de comandos. En algunas ocasiones deberemos pulsar una tecla para ver más. En otras, podemos añadir la opción `| more` al final del comando para mostrar los resultados hasta llenar la página; para mostrar la siguiente página debemos presionar la tecla espaciadora (o la tecla ENTER para mostrar la información línea a línea). A modo de ejemplo, vamos a la carpeta raíz C:\ escribiendo el comando `cd\`. A continuación, cambiamos a la carpeta C:\windows\system32 ejecutando el comando `cd\windows\system32`. Una vez allí, ejecutamos un `dir`. Esto inundará la pantalla con cientos de líneas de información. Para ver la información pantalla a pantalla, basta con escribir `dir | more`.

E.3. COMANDOS BÁSICOS.

Los comandos `ipconfig` y `ping` son unos de los más útiles para los administradores de redes. Estos comandos básicos pueden ayudarnos a analizar y detectar problemas en nuestras redes. Como todos los comandos que estudiaremos en esta práctica, se ejecutan a través de la ventana de comandos.

COMANDO IPCONFIG.

El comando `ipconfig` muestra la información relacionada con las **tarjetas de red** (o **adaptadores de red**) de nuestro ordenador. (A esta información también se la denomina "configuración TCP/IP"). Para probar este comando, ejecutamos `ipconfig` en la ventana de comandos. Los resultados deberían ser parecidos a los mostrados en la figura E.3.

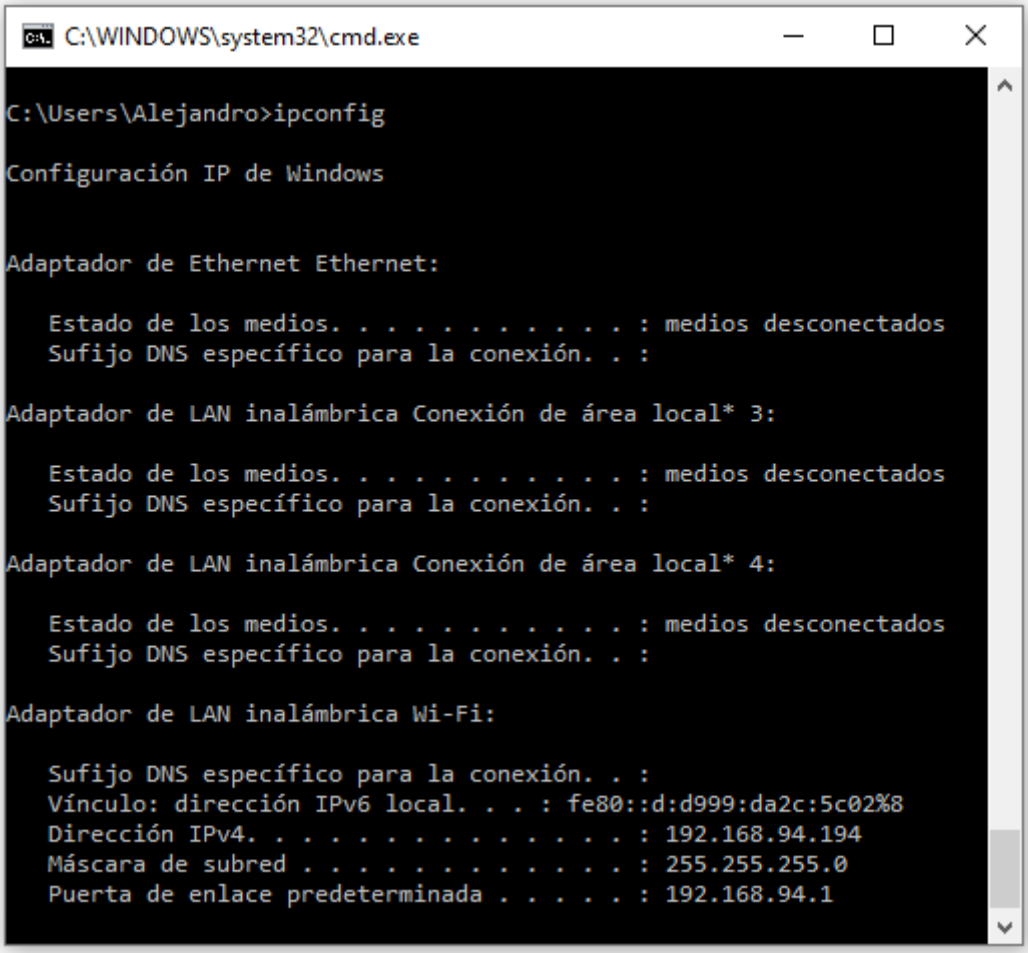


Figura E.3. Resultados del comando `ipconfig`.

Como vemos, aquí es donde podemos encontrar la dirección IP de nuestros adaptadores activos, y otras informaciones útiles como la máscara de subred, y la dirección IP de la puerta de enlace predeterminada (esto es, el rúter que nos conecta a Internet). Notar que lo habitual es que un ordenador disponga de dos adaptadores de red distintos: Un adaptador de red Ethernet para conectarnos a redes de cable, y un adaptador de red WiFi para conectarnos a redes inalámbricas. Los rúters disponen de tantos adaptadores como redes tengan interconectadas.

Ahora bien, ésta no es toda la información que `ipconfig` puede mostrar. Por ejemplo, si queremos conocer la dirección física de nuestros adaptadores de red, debemos usar una de las múltiples opciones disponibles para `ipconfig`. Ejecutamos el comando `ipconfig /all`. Esta opción arroja muchos más resultados que antes, incluyendo las direcciones físicas de los adaptadores de red de nuestro equipo:

```
C:\Users\Alejandro>ipconfig /all

Configuración IP de Windows

   Nombre de host. . . . . : Lenovo-PC
   Sufijo DNS principal  . . . . . :
   Tipo de nodo. . . . . : híbrido
   Enrutamiento IP habilitado. . . : no
   Proxy WINS habilitado . . . . . : no
```

```
Adaptador de Ethernet Ethernet:

   Estado de los medios. . . . . . . . . . . : medios desconectados
   Sufijo DNS específico para la conexión. . :
   Descripción . . . . . . . . . . . . . . . : Realtek PCIe GBE Family Controller
   Dirección física. . . . . . . . . . . . . : 28-D2-44-FC-EC-BC
   DHCP habilitado . . . . . . . . . . . . . : sí
   Configuración automática habilitada . . . : sí

Adaptador de LAN inalámbrica Wi-Fi:

   Sufijo DNS específico para la conexión. . :
   Descripción . . . . . . . . . . . . . . . : Realtek RTL8723BE Wireless LAN 802.11n PCI-E NIC
   Dirección física. . . . . . . . . . . . . : 10-08-B1-7B-5B-4B
   DHCP habilitado . . . . . . . . . . . . . : sí
   Configuración automática habilitada . . . : sí
   Vínculo: dirección IPv6 local. . . : fe80::d:d999:da2c:5c02%8(Preferido)
   Dirección IPv4. . . . . . . . . . . . . . : 192.168.94.194(Preferido)
   Máscara de subred . . . . . . . . . . . . : 255.255.255.0
   Concesión obtenida. . . . . . . . . . . . : martes, 11 de febrero de 2020 8:34:13
   La concesión expira . . . . . . . . . . . : martes, 11 de febrero de 2020 20:43:04
   Puerta de enlace predeterminada . . . . . : 192.168.94.1
   Servidor DHCP . . . . . . . . . . . . . . : 192.168.94.1
   IAID DHCPv6 . . . . . . . . . . . . . . . : 135268529
   DUID de cliente DHCPv6. . . . . . . . . . : 00-01-00-01-1B-B5-DC-C1-28-D2-44-FC-EC-BC
   Servidores DNS. . . . . . . . . . . . . . : 8.8.8.8
                                       80.58.61.250
   NetBIOS sobre TCP/IP. . . . . . . . . . . : habilitado
```

Notar que al principio de los resultados mostrados, hay una sección llamada "Configuración IP de Windows". Aquí encontramos el **nombre del ordenador**, o **nombre de host**. (También podemos obtener este dato ejecutando el comando `hostname`).

El comando `ipconfig` acepta muchas más opciones (parámetros). Para conocer cuáles son estas opciones, ejecutamos `ipconfig /?`. Con esto mostramos el archivo de ayuda de `ipconfig`, que como vemos, es muy extenso. Este archivo describe qué es `ipconfig`, qué hace, y muestra las distintas opciones que podemos usar con él, junto con algunos ejemplos de uso. Por el momento, no necesitaremos usar ninguna de estas opciones, a excepción de la opción `/all`.

COMANDO PING.

El comando `ping` se usa para comprobar la conectividad con otro host de la red, y nos indica si ese otro host está "vivo" en la red. El comando `ping` utiliza el protocolo **ICMP** (Internet Control Message Protocol). Este protocolo de capa de red permite que un host pueda enviar paquetes de prueba a otro host remoto para obtener información acerca de posibles problemas de conexión con él.

Sin embargo, y antes de empezar a trabajar con el comando `ping`, debemos comprobar si nuestro ordenador está correctamente configurado para trabajar en red. (Cuidado: Esta configuración solo es necesaria si nuestro ordenador forma parte de una red de ordenadores como la del aula informática; si estamos trabajando desde casa, la podemos omitir). El proceso es el siguiente:

En primer lugar, nos aseguramos de que nuestro ordenador y el resto de ordenadores de la red están visibles. Para ello abrimos el explorador de archivos, y seleccionamos la carpeta llamada "Red". El explorador mostrará todos los dispositivos que están conectados a la red, ver figura E.4. Si nuestro ordenador no detecta ningún dispositivo en red, debemos activar el **uso compartido**. Para ello, vamos a "Configuración", seleccionamos la opción "Red e Internet", y a continuación elegimos "Opciones de uso compartido". Dentro de esta ventana, activamos todas las opciones para permitir la detección de redes y el uso compartido de archivos e impresoras.

A continuación debemos definir un nombre para nuestro ordenador, y asignarlo a un grupo de trabajo. Inicialmente, todos los ordenadores vienen con un nombre de host por defecto, asignado en el proceso

de fabricación. (Este nombre suele ser DESKTOP - <código>, LAPTOP - <código>, o similar). El nombre por defecto es poco intuitivo, y podría estar replicado en varios ordenadores de la red. Para cambiar el nombre del equipo acudimos al explorador de archivos, y con el botón derecho del ratón, pulsamos en "Este equipo", y seleccionamos la opción "Propiedades". En la ventana que muestra la información básica del equipo, en la zona de configuración de nombre, dominio, y grupo de trabajo, pulsamos en "Cambiar configuración". Después, en la ventana de Propiedades del Sistema, pulsamos el botón "Cambiar…" para modificar el nombre del equipo (ver figura E.5). Nosotros no cambiaremos el nombre del grupo de trabajo, y mantendremos su valor por defecto (WORKGROUP).

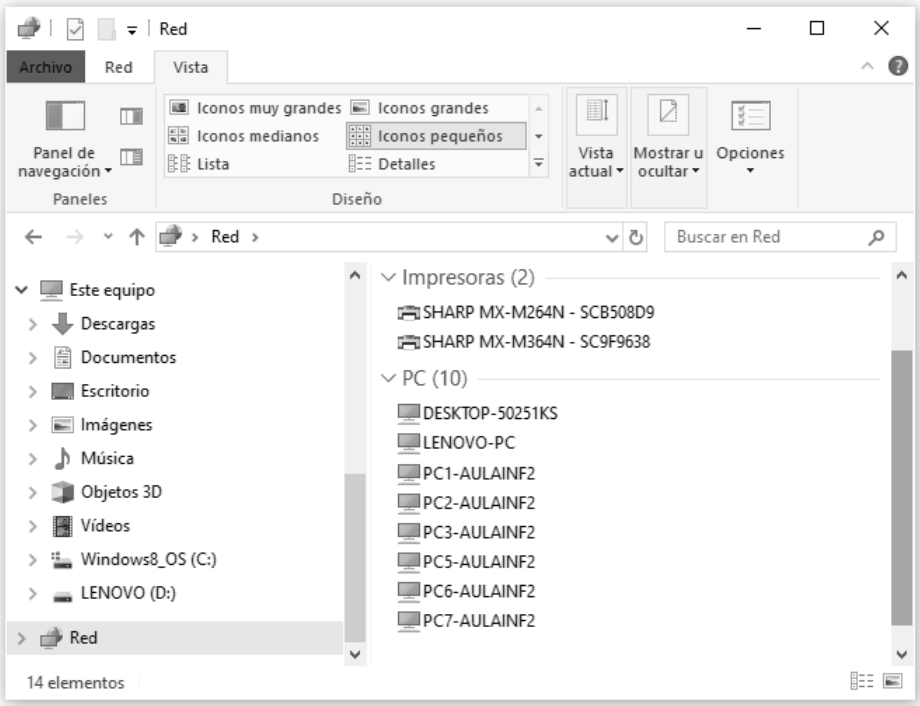


Figura E.4. Explorador de archivos mostrando los equipos en red.

Figura E.5. Cambio del nombre de host del ordenador.

Una vez tenemos todo configurado, vamos a probar el comando para hacer un `ping` a otros ordenadores de la red (o a la puerta de enlace predeterminada, si no estamos en red), a nuestro propio ordenador, y a un host remoto de Internet.

Primero ejecutamos el comando `ipconfig`. Nos apuntamos la dirección IP de nuestro ordenador (en mi caso, 10.127.96.147) y la dirección IP de otro ordenador de la red (en mi caso, 10.127.96.143; vosotros podéis apuntaros la dirección IP del ordenador de algún compañero). Si no estamos en red, en vez apuntarnos la dirección IP de otro ordenador, tomamos nota de la dirección IP de la puerta de enlace predeterminada (el rúter). Ahora, vamos a hacer un `ping` a la dirección IP del otro ordenador, por ejemplo, `ping 10.127.96.143`. Este es el comando `ping` básico, y comprueba si el otro host está vivo en la red. También podemos hacer un `ping` a otro ordenador usando su nombre de host en vez de su dirección IP. El comando `ping` envía una serie de paquetes a la otra dirección IP. (Los tamaños de los paquetes enviados son, por defecto, de 32 bytes). Si el otro ordenador está activo en la red, debería responder. Pero si el ordenador no está vivo o no está disponible, obtendremos uno de los posibles mensajes de error, como "host de destino inaccesible", "tiempo de espera agotado", etc. La figura E.6 muestra el aspecto de un `ping` positivo, en el que hemos recibido cuatro respuestas afirmativas. La figura E.7 muestra un `ping` a otro ordenador usando su nombre del host.

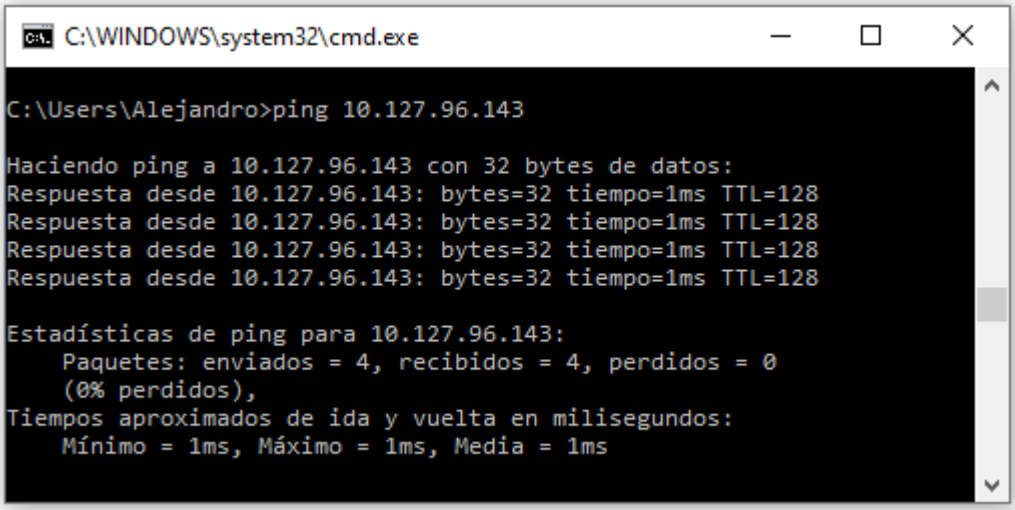


Figura E.6. Resultado de un `ping` a otro ordenador de la red.

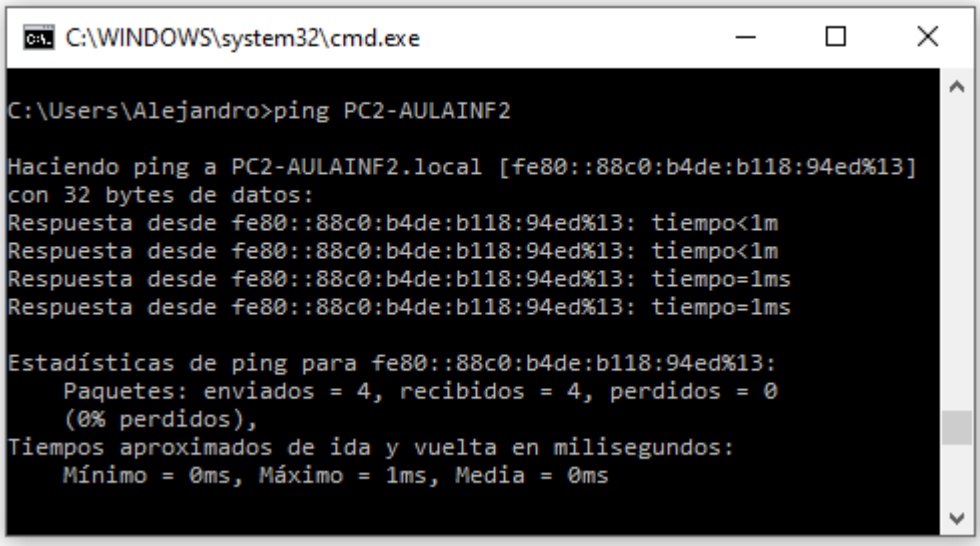


Figura E.7. Resultado de un `ping` por nombre de host.

Seguidamente, vamos a hacer un `ping` a nuestro propio ordenador. Para ello, ejecutamos `ping localhost`; después `ping loopback`; y finalmente `ping 127.0.0.1`. Los dos primeros comandos son

básicamente iguales, y ofrecen los mismos resultados. Por el contrario, el comando `ping 127.0.0.1` no pone tráfico en la red; el adaptador de solo envía paquetes a su propio SO, y todo el tráfico queda dentro del ordenador. Esta última es la mejor forma de comprobar si los protocolos TCP/IP están correctamente instalados en un adaptador de red, porque ni siquiera es necesario estar conectados físicamente a una red.

A la hora de comprobar problemas de conectividad en nuestra red con el comando `ping`, debemos comenzar con el host local, y seguir con el resto de hosts conectados a la red, terminando con el rúter. Después, podemos probar con hosts fuera de nuestra red (por ejemplo, podemos probar a ejecutar `ping www.facebook.com`).

Vamos a probar otras opciones del comando `ping`. Comenzamos haciendo un `ping` con paquetes de más de 32 bytes. Para ello, elegimos otro ordenador al que hacerle un `ping`, como el ordenador de un compañero, un segundo ordenador, o el rúter. Aquí vamos a usar la dirección IP 10.127.96.143. Ejecutamos el comando `ping -l 1500 10.127.96.143`. Notar que ahora enviamos paquetes de 1500 bytes, en lugar de hacerlo con los paquetes de 32 bytes de un `ping` básico. La opción `-l` nos permite modificar el tamaño de los paquetes ICMP que enviamos. El tamaño máximo es de 65500 bytes, pero con esto podríamos crear paquetes fragmentados. Esta opción nos puede ayudar a simular tráfico hacia un host en particular. Ahora ejecutamos el comando `ping -n 10 10.127.96.143`. Con esto enviamos 10 paquetes ICMP, en lugar de los 4 paquetes que se envían por defecto. La opción `-n` nos permite enviar tantos paquetes ICMP como deseemos. Esto sirve para comparar el funcionamiento de un ordenador a lo largo del tiempo. Por ejemplo, si ejecutamos `ping -n 1000 10.127.96.143` todos los días, podemos comparar los resultados y ver si el ordenador destino funciona mejor o peor de lo habitual. (También es posible combinar ambos comandos en una sola instrucción, como por ejemplo, `ping -l 500 -n 8 10.127.96.143`). Por último, hacemos un `ping` continuo a un ordenador ejecutando `ping -t 10.127.96.143`. Este comando envía sin parar paquetes ICMP a la dirección IP de destino. Solo podemos detener este envío continuo pulsando CTRL + C o cerrando la ventana de comandos. Esta opción es interesante para comprobar si hemos realizado una conexión correctamente. Por ejemplo, si no estamos seguros del puerto al que deberíamos conectar un ordenador, podemos iniciar un `ping` continuo a la IP de ese ordenador, y conectarlo a uno u otro puerto hasta recibir respuestas afirmativas. (También podemos hacer un `ping` continuo con paquetes de más de 32 bytes con la combinación `ping -t -l 500 10.127.96.143`).

E.4. COMANDOS AVANZADOS.

COMANDO ARP.

El comando `arp` sirve para consultar (y modificar) la tabla local de protocolo **ARP** (Address Resolution Protocol), que relaciona las direcciones lógicas de los equipos de una red con sus direcciones físicas, con objeto de mover las tramas a través de esa red.

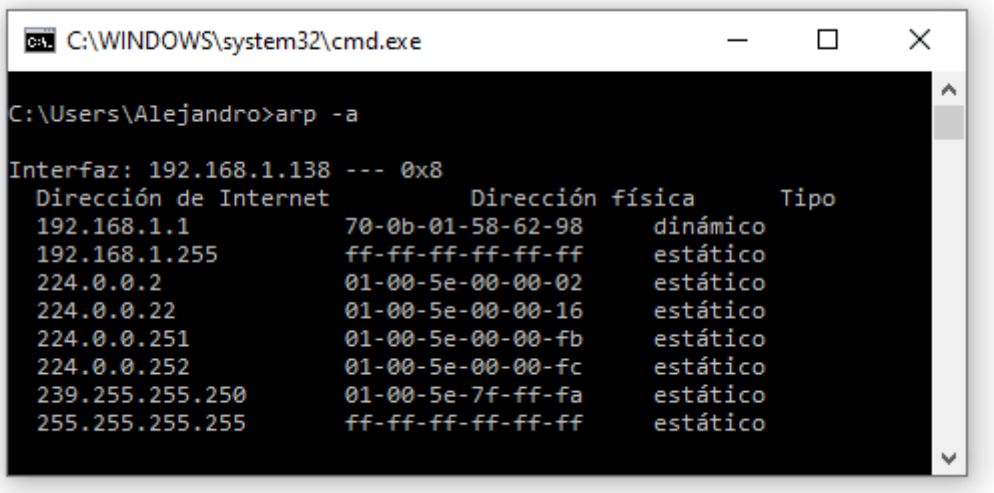


Figura E.8. Resultado del comando `arp -a`.

Para empezar, ejecutamos el comando `arp` en la ventana de comandos. Este comando muestra el archivo de ayuda de `arp`. A continuación, ejecutamos el comando `arp -a`. La figura E.8 muestra el resultado.

A modo de ejemplo, la primera fila de la tabla ARP muestra la dirección IP de la puerta de enlace (el rúter) al que estaba conectado (192.168.1.1), y su dirección física (70:0B:01:58:62:98).

El comando `arp -d [direccion IP]` nos permite borrar de la tabla ARP la entrada asociada a la IP indicada. Pero para poder usar este comando, necesitamos acceder a la ventana de comandos con **permisos de administrador**. Para abrir la ventana de comandos como administradores, accedemos a la carpeta C:\Windows\system32, y en el icono del ejecutable *cmd.exe*, pulsamos en el botón derecho del ratón y seleccionamos "Ejecutar como administrador". (Otra forma más rápida de hacerlo es pulsar en el botón de inicio de Windows y escribir "cmd". En los resultados mostrados, pulsamos con el botón derecho del ratón sobre el símbolo del sistema, y elegimos la opción "Ejecutar como administrador", ver figura E.9).

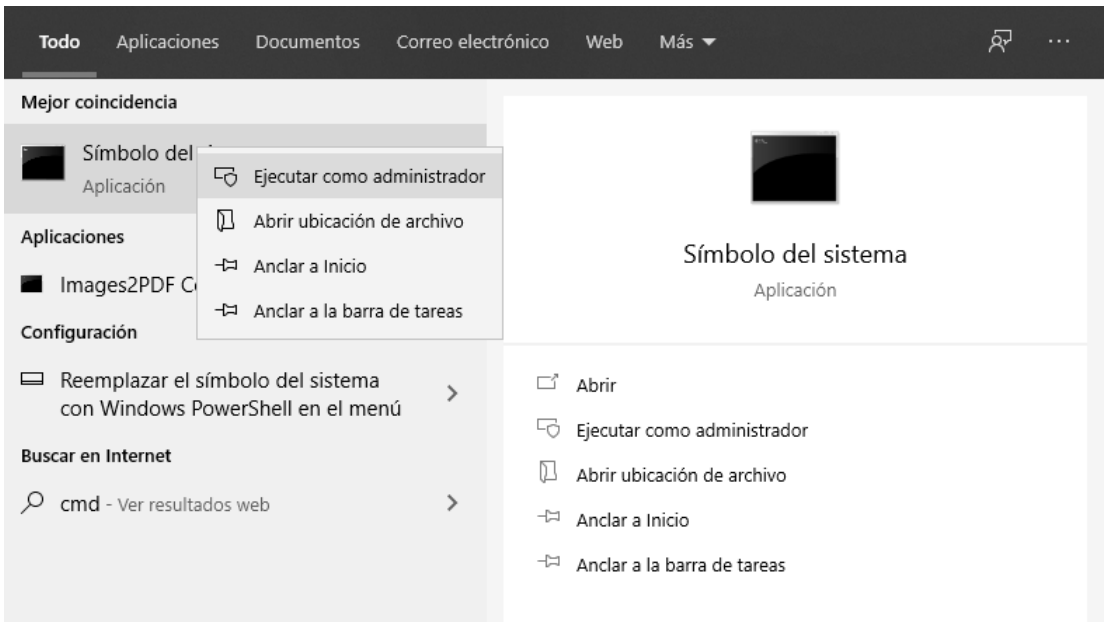


Figura E.9. Ejecutar la ventana de comandos como administrador.

Una vez dentro de la ventana de comandos como administradores, y a modo de ejemplo, ejecutamos el comando `arp -d` sobre una de las IP de la tabla ARP de nuestro ordenador, por ejemplo, `arp -d 192.168.1.1`. Si a continuación volvemos a ejecutar `arp -a`, veremos que la entrada asociada a la IP 192.168.1.1 ya no está en la tabla, ver figura E.10.

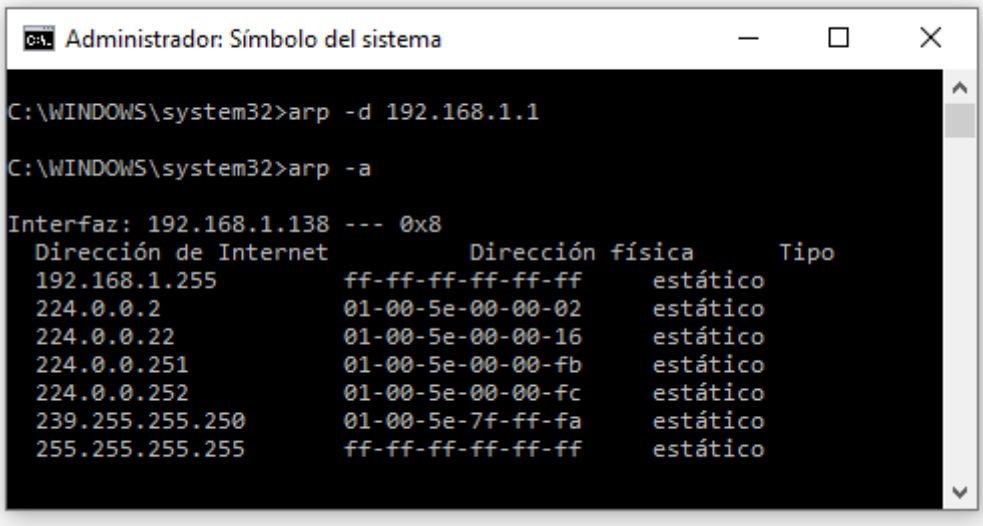


Figura E.10. Resultado del comando `arp -d`.

(Cuidado: Debemos ser rápidos a la hora de visualizar los resultados tras borrar una entrada, porque los ordenadores actualizan automáticamente su tabla ARP con bastante rapidez). Otra opción es usar el comando `arp -d *`. Al ejecutar este comando, eliminamos todas las entradas de la tabla ARP.

También podemos utilizar el comando `arp -s [direccion IP] [dirección física]` para añadir una nueva entrada a la tabla ARP. Por ejemplo, si queremos añadir una entrada al equipo cuya dirección IP es 192.168.1.2 para asociarle la dirección física 00:AA:00:62:C6:09, el comando que deberíamos usar es `arp -s 192.168.1.2 00-AA-00-62-C6-09`. Si tras hacer esto ejecutamos el comando `arp -a`, veremos que la tabla ARP incorpora ahora una entrada asociada a la IP 192.168.1.2, ver figura E.11.

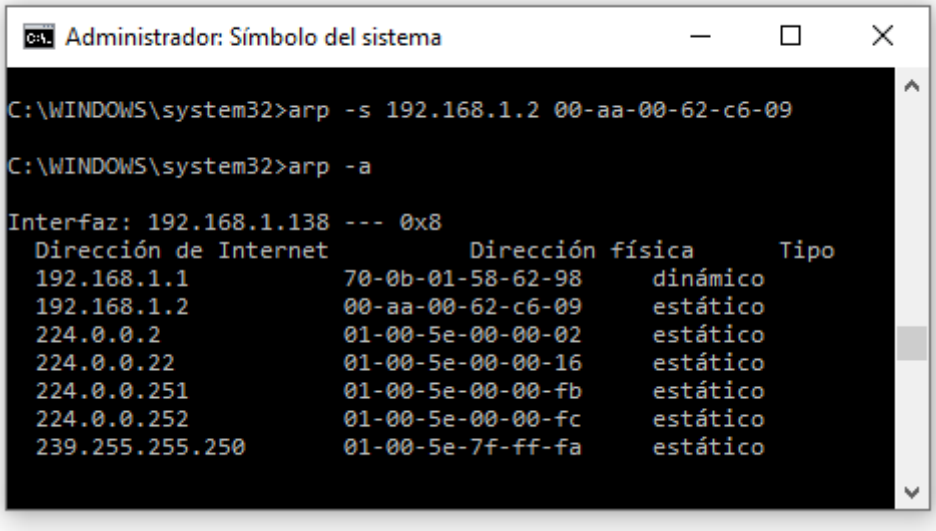


Figura E.11. Resultado del comando `arp -s`.

COMANDOS TRACERT Y PATHPING.

A continuación vamos a usar los comandos `tracert` y `pathping` para analizar el camino seguido por los paquetes a través de la red. Ambos comandos muestran la ruta a través de uno o más rúters hacia el host remoto especificado, aunque la sintaxis y los resultados ofrecidos son algo distintos. Por ejemplo, además de trazar la ruta, el comando `pathping` también efectúa un análisis posterior de la tasa de paquetes perdidos.

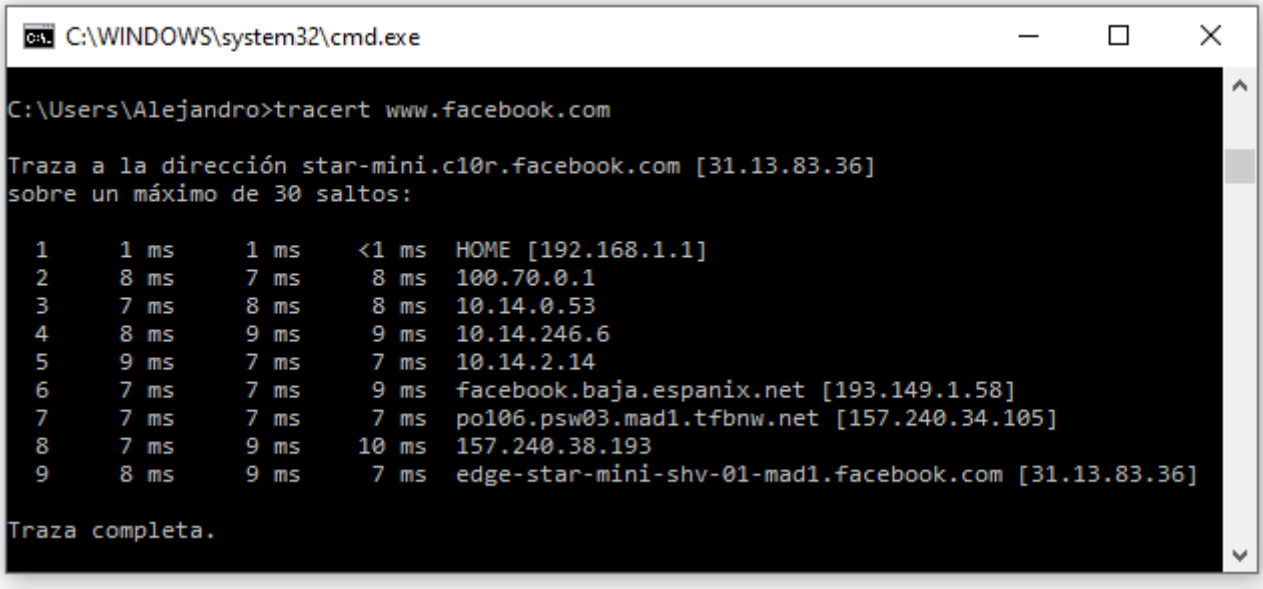


Figura E.15. Resultados del comando `tracert www.facebook.com`.

Para empezar, ejecutamos el comando `tracert` y observamos los resultados. Al igual que `tracert /?`, este comando muestra el archivo de ayuda del comando. Como vemos, el comando `tracert` muestra la ruta hacia un host destino o hacia otra red. Esto lo consigue realizando un triple `ping` en cada salto de la ruta hacia el destino.

Para probar el comando, vamos a trazar la ruta al sitio web de Facebook. Para ello, ejecutamos `tracert www.facebook.com`. Los resultados deberían ser similares a los mostrados en la figura E.15. A cada paso de la ruta se le denomina **salto** (hop). Cada línea de los resultados es una red por la que los paquetes han pasado hasta llegar a su destino. Observar el nombre de cada rúter y su correspondiente dirección IP. Mirando estos nombres podemos hacer un seguimiento geográfico de la ruta que han seguido los paquetes ICMP en su ruta hacia el destino. En este ejemplo, los paquetes han necesitado 9 saltos para alcanzar su destino. El primer salto (dirección IP 192.168.1.1) se corresponde con el rúter WiFi que, en el momento de realizar la traza, conectaba mi ordenador a Internet. El resto de saltos hasta la dirección IP 10.14.2.14 se repitieron en las trazas a destinos tan dispares como www.amazon.com, www.nasa.gov, etc., y probablemente se corresponden con los diversos rúters que los paquetes deben atravesar dentro del ISP que me daba servicio de Internet. Ahora ejecutamos el comando `tracert -d www.facebook.com` para hacer la misma traza numéricamente (y no por nombre, como en el ejemplo previo), ver figura E.16. Esto nos permite ahorrar muchísimo tiempo a la hora de obtener resultados, ya que no es necesario hacer la resolución de direcciones IP a nombres de equipo. El comando `tracert` es útil para descubrir si un rúter no está funcionando. Comparando los resultados del comando `tracert` con el diagrama de la red bajo prueba deberíamos ser capaces de detectar qué salto ha fallado, y en su caso, avisar a la persona responsable o corregir el problema nosotros mismos. Es muy habitual que el problema se solucione simplemente reiniciando el rúter que ha fallado.

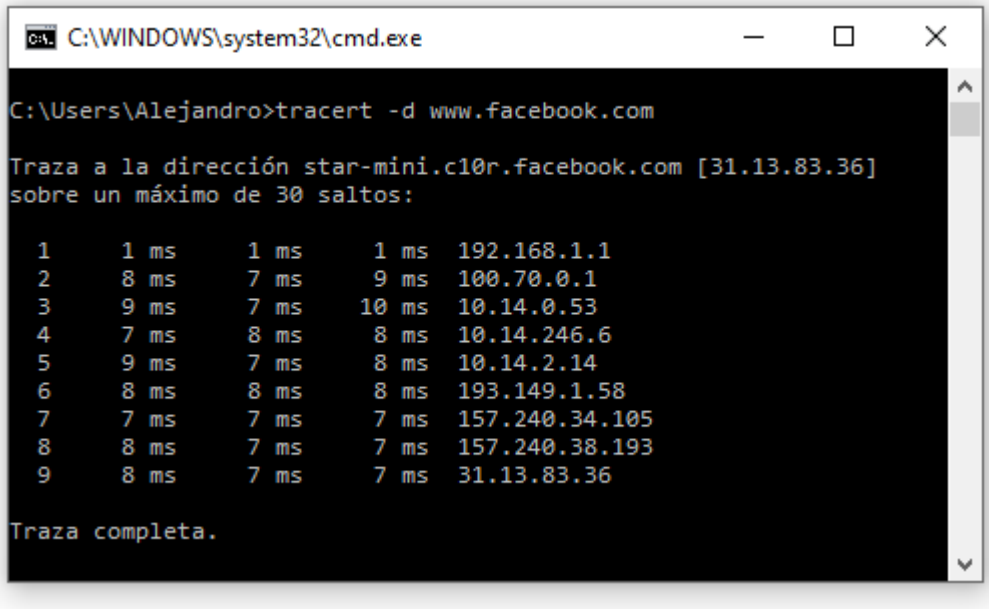


Figura E.16. Resultados del comando `tracert -d www.facebook.com`.

Seguidamente, vamos a ejecutar el comando `pathping www.facebook.com`. El comando `pathping` es similar a `tracert`, con la diferencia de que también calcula la tasa de pérdida de paquetes, como vemos en la figura E.17. Esta información aparecerá bajo la columna Perdidos/Enviados. Finalmente, ejecutamos `pathping -n www.facebook.com` para evitar la resolución de nombres, tal y como lo hace el comando `tracert -d`. Con esto podremos obtener resultados de forma mucho más rápida que mediante el comando `pathping` estándar.

NOTA IMPORTANTE: En algunas redes u ordenadores los comandos `tracert` y `pathping` podrían no funcionar o estar deshabilitados. En tales casos, todavía podemos trazar la ruta a un destino usando una herramienta online (como https://ping.eu/traceroute/ y http://en.dnstools.ch/visual-traceroute.html), o una aplicación de escritorio (como Visual Route, ver figura E.18).

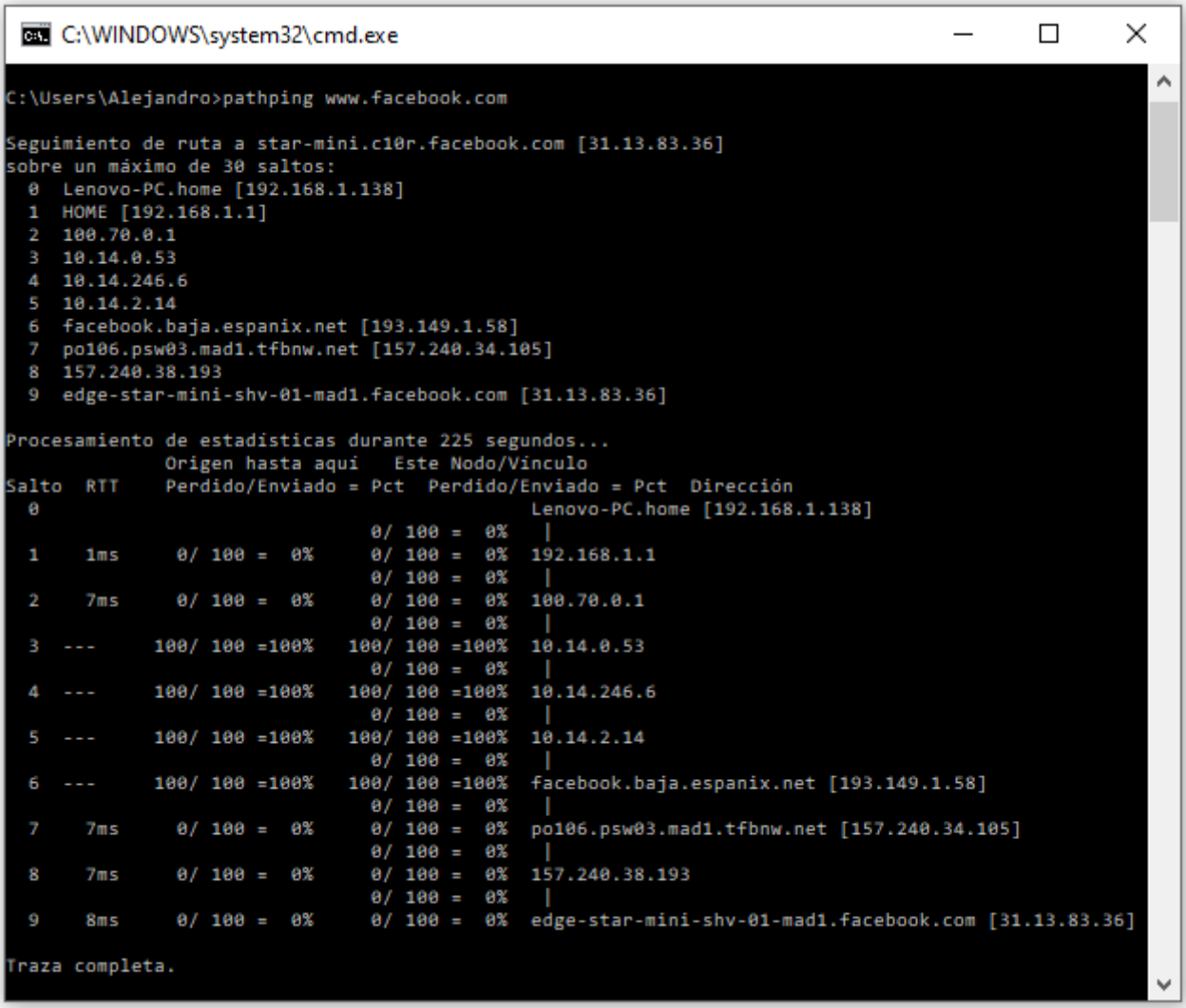


Figura E.17. Resultados del comando `pathping www.facebook.com`.

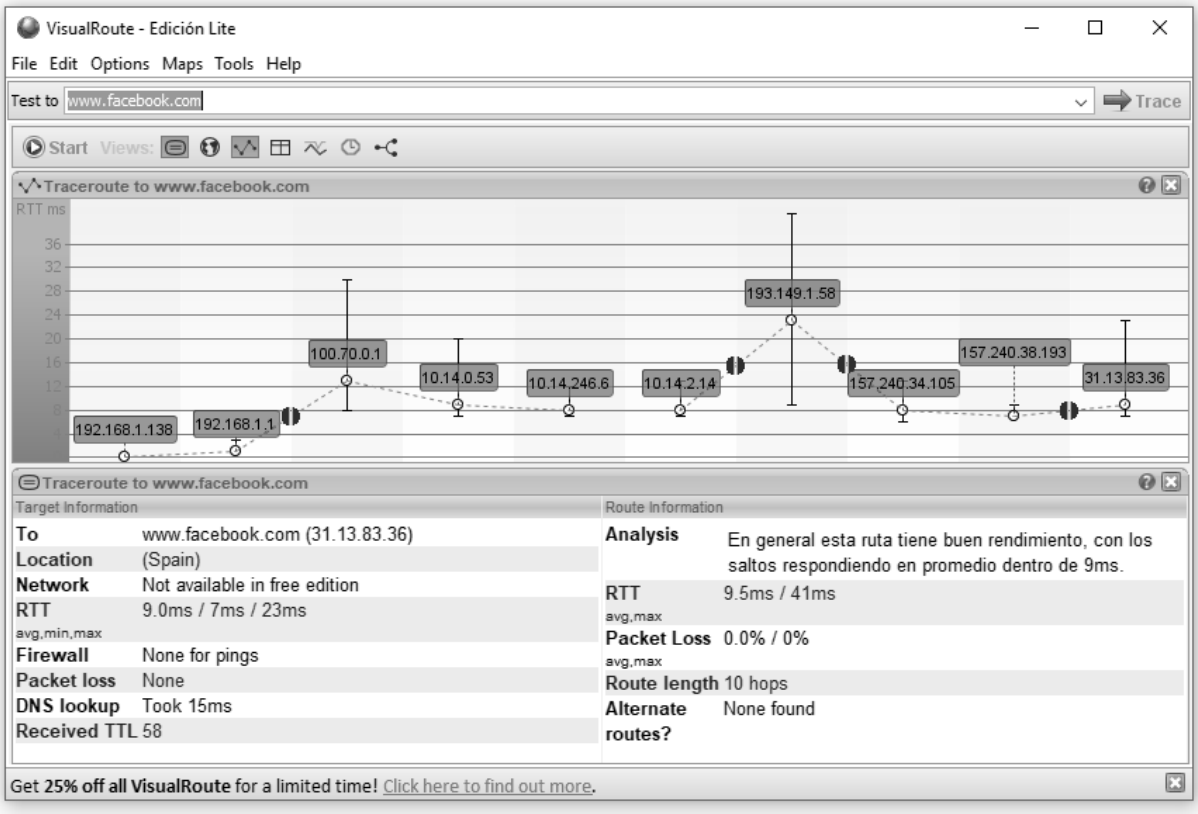


Figura E.18. Resultado de la traza a Facebook en la aplicación Visual Route.

COMANDO NSLOOKUP.

El comando `nslookup` muestra información acerca de los nombres de dominio de los hosts y sus correspondientes direcciones IP. Para probar este comando, ejecutamos `nslookup www.facebook.com` y observamos los resultados. Deberíamos ver la dirección IP correspondiente al nombre de dominio www.facebook.com (ver figura E.19). También podemos probar este comando con otros nombres de dominio bien conocidos (Amazon, Microsoft, NASA, Google, etc.).

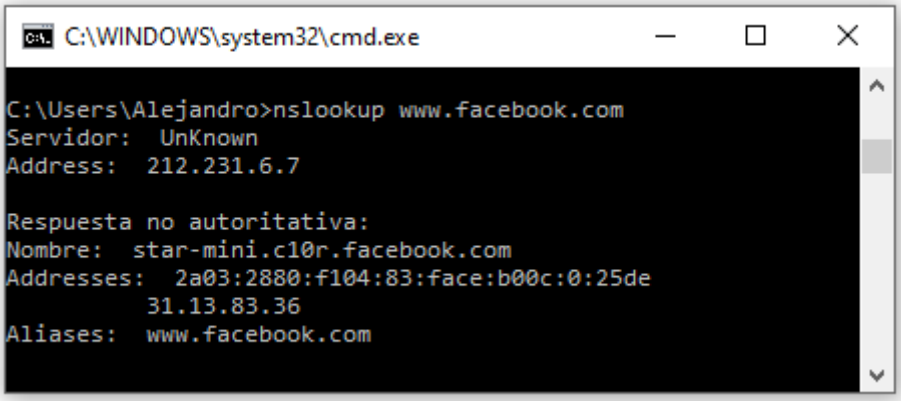


Figura E.19. Resultado del comando `nslookup www.facebook.com`.

COMANDO NETSTAT.

`netstat` es un comando que muestra un listado de las conexiones de capa de transporte que tiene activas un ordenador (entrantes y salientes). Para ver este comando en acción ejecutamos `netstat` en la ventana de comandos y observamos los resultados.

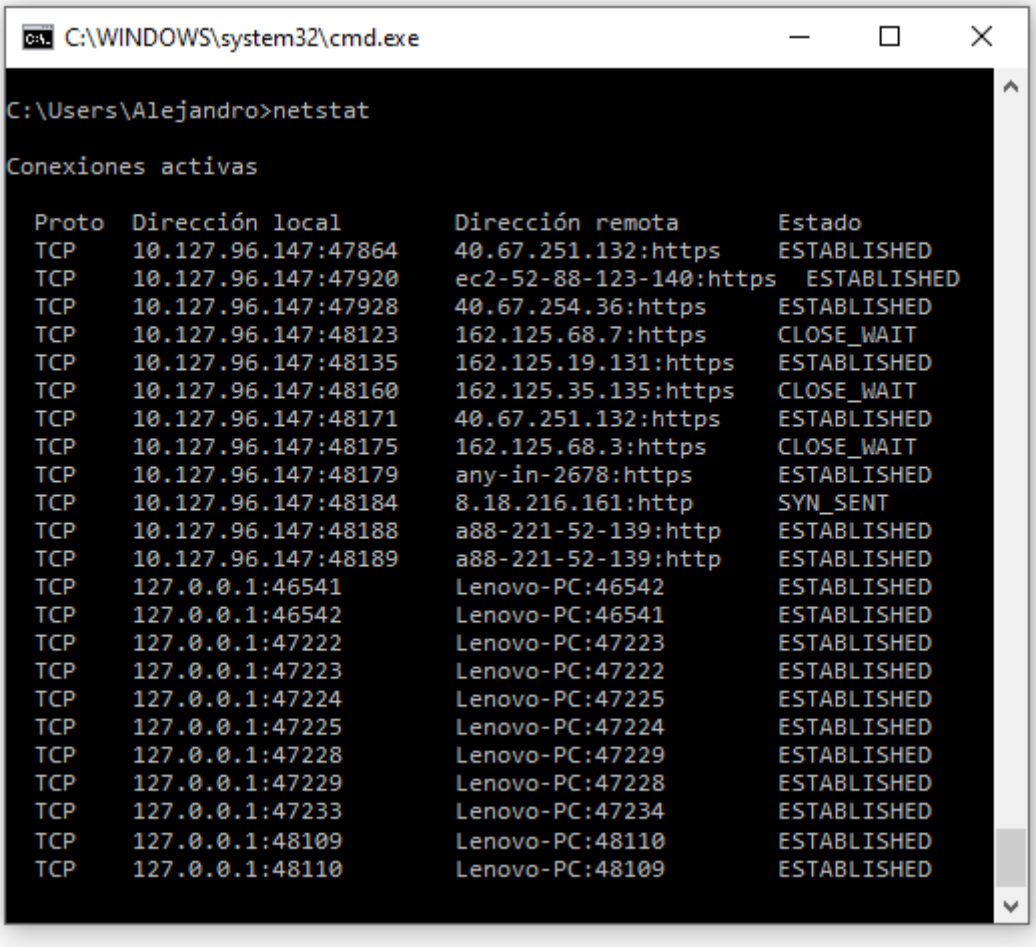


Figura E.12. Resultado del comando `netstat`.

Este comando puede tardar unos pocos minutos en responder, dependiendo de la configuración de la red y de la cantidad de conexiones existentes. Los resultados deberían ser similares a los mostrados en la figura E.12. Estos resultados muestran las conexiones TCP (o UDP) que están activas. Cada conexión aparece en una línea de la tabla de resultados. Notar que tenemos cuatro columnas de información para cada conexión. La columna "Proto" muestra el protocolo de capa de transporte usado por la conexión. (El comando `netstat` sin opciones solo muestra las conexiones TCP). La columna "Dirección local" muestra el identificador del ordenador local (nombre o dirección IP), seguido del número de puerto de salida. La columna "Dirección remota" muestra el identificador del ordenador remoto (nombre o dirección IP), junto con el puerto de entrada. En algunos casos, el ordenador remoto puede ser el propio ordenador local. La columna "Estado" muestra el estado de la conexión, y puede ser ESTABLISHED, CLOSE_WAIT, CLOSED, LISTEN, SYN_SENT, etc.

Vamos a mostrar otro ejemplo con una nueva conexión establecida: Abrimos un navegador de Internet y nos conectamos a una página web conocida, como www.elmundo.es. En la ventana de comandos, ejecutamos `nslookup www.elmundo.es` para averiguar la dirección IP de esta web (que en el momento de escribir estos apuntes era 151.101.121.50). Ahora volvemos a ejecutar `netstat`. En los resultados, deberíamos ver entradas adicionales, como las mostradas en la figura E.13. En concreto, debemos prestar atención a las entradas para las que la columna de dirección remota muestra la dirección IP 151.101.121.50. Vemos que hay dos conexiones con la dirección 151.101.121.50, que se crearon cuando el ordenador se conectó con la web del periódico El Mundo.

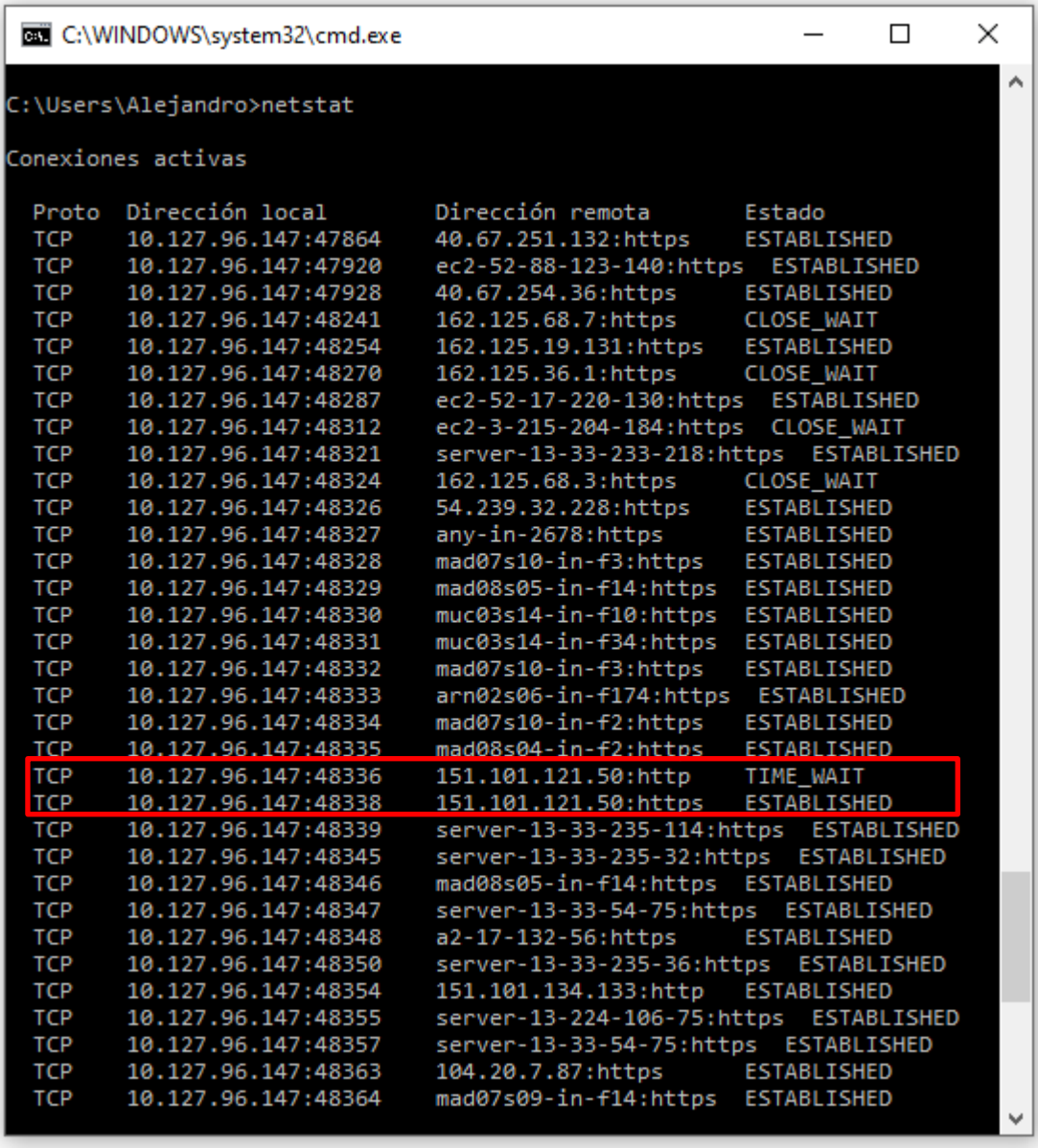


Figura E.13. Resultado del comando `netstat` con entradas adicionales.

El comando `netstat` es muy útil para controlar las conexiones de red que crean las aplicaciones.

A continuación, ejecutamos el comando `netstat -a`. Esto nos permite mostrar las conexiones TCP y UDP.

Ahora, ejecutamos el comando `netstat -an`. Esto muestra las conexiones TCP y UDP en formato numérico. En muchas ocasiones, es mejor ver las conexiones por direcciones IP y números de puerto que verlas por nombre.

Ejecutamos `netstat -e`. Con esto mostramos estadísticas Ethernet, como el número de paquetes y bytes enviados y recibidos, ver figura E.14.

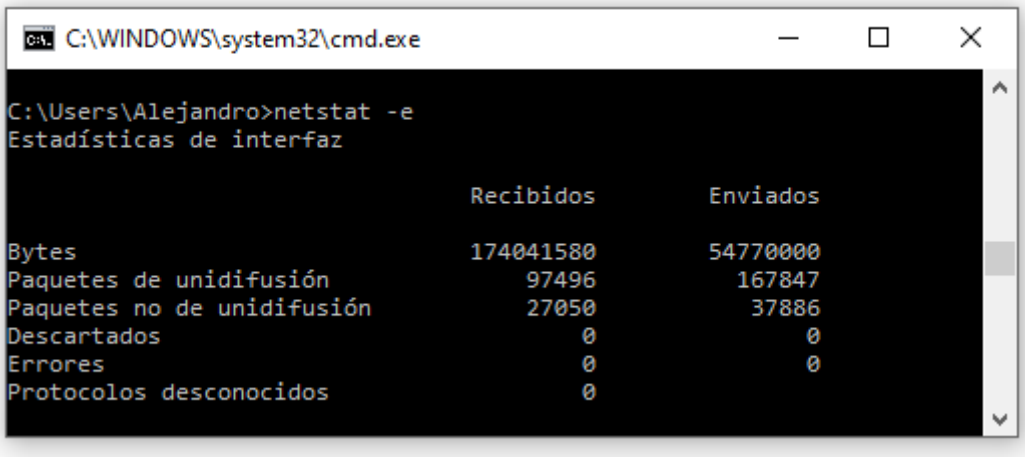


Figura E.14. Estadísticas Ethernet.

Ejecutamos `netstat -r` para obtener la tabla de encaminamiento de nustro equipo local. Este comando produce los mismos resultados que el comando `route`, del que hablaremos a continuación.

Finalmente, ejecutamos `netstat -s`. Este comando muestra estadísticas para cada protocolo, como TCP, UDP, ICMP, IP, etc.

COMANDO ROUTE.

El comando `route` nos permite mostrar y modificar la **tabla de encaminamiento** local del ordenador en cuestión. Este comando es especialmente útil para examinar las tablas de encaminamiento de los rúters, pero nosotros solo lo podremos ver en acción en nuestro ordenador personal. Normalmente, los ordenadores no tienen grabadas rutas hacia otras redes, porque estas máquinas no se ocupan de esas tareas.

Comenzamos ejecutando el comando `route print`. Deberíamos obtener unos resultados similares a los mostrados en la figura E.20. Este comando muestra la misma información que `netstat -r`, pero es mucho más utilizado. Los resultados muestran la lista de adaptadores de red (o interfaces) del ordenador local, incluyendo la dirección física y el nombre de cada uno de ellos. A continuación muestran la tabla de encaminamiento IPv4. La tabla de encaminamiento IPv4 muestra las conexiones IP a otras redes. Como vemos, hay varias conexiones de red. La columna "Destino de red" indica a qué destino está intentando conectarse el ordenador local. La columna "Máscara de red" es la máscara de subred del destino de red correspondiente, y sirve para identificar qué parte de la dirección IP se corresponde con la red, y qué parte se corresponde con el host. La columna "Puerta de enlace" es la dirección IP de la máquina (típicamente, un rúter) que proporciona acceso a la red remota. La columna "Interfaz" es la dirección IP del adaptador de red que está estableciendo la conexión con la red remota. La columna de "Métrica" es un entero entre 1 y 9999, y su valor depende de parámetros como la velocidad de la conexión, el número de saltos para llegar a la red remota, etc. La **métrica** es una especie de medida del coste de llegar a la red

destino, y lo habitual es seleccionar la ruta con la métrica más baja para establecer una conexión con una red remota.

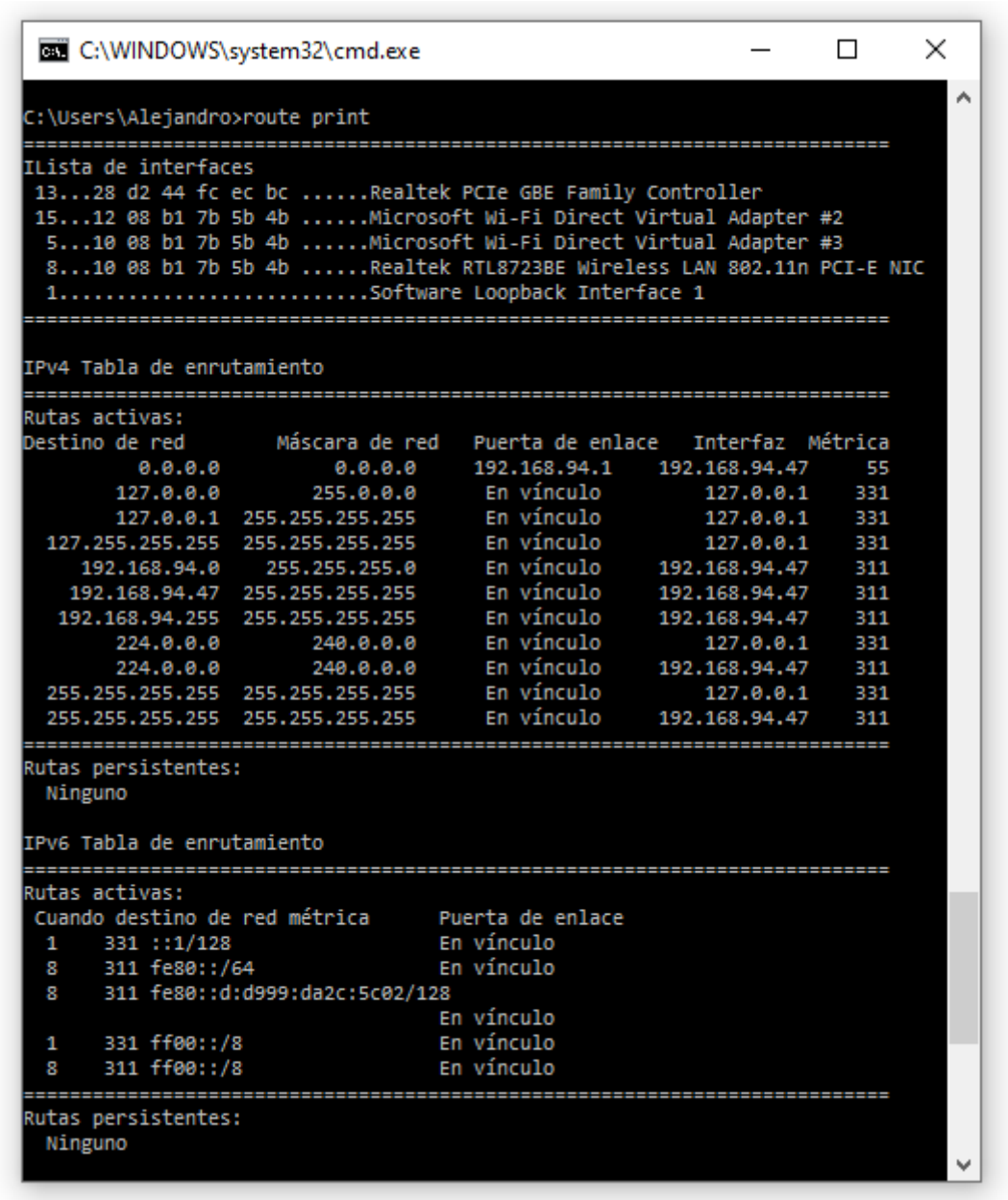


Figura E.20. Resultados del comando `route print`.

A modo de ejemplo, imaginar que una de las líneas de la tabla de encaminamiento fuese:

Destino de red	Máscara de red	Puerta de enlace	Interfaz	Métrica
10.254.254.0	255.0.0.0	10.254.254.112	192.168.94.47	331

Esto significa que un paquete con destino a cualquier máquina en la subred 10.254.254.0 debería reenviarse a la puerta de enlace 10.254.254.112, a través de la interfaz 192.168.94.47 del ordenador origen. Un destino de red igual a 0.0.0.0 identifica la puerta de enlace por defecto. Si en la tabla de encaminamiento no hay una entrada para un cierto destino de red, los paquetes se envían a la puerta de enlace por defecto a través del interfaz especificado.

Ahora vamos a añadir y a eliminar entradas en la tabla de encaminamiento de nuestro ordenador, pero para ello, debemos acceder a la ventana de comandos con **permisos de administrador**. Una vez dentro de la línea de comandos como administradores, vamos a añadir una ruta ficticia usando nuestra propia dirección IP local como interfaz que conecta con la red remota. Para ello, ejecutamos `route add 192.168.1.0`

`mask 255.255.255.0 [direcciónIPLocal]`. La figura E.21 muestra el resultado de ejecutar este comando.

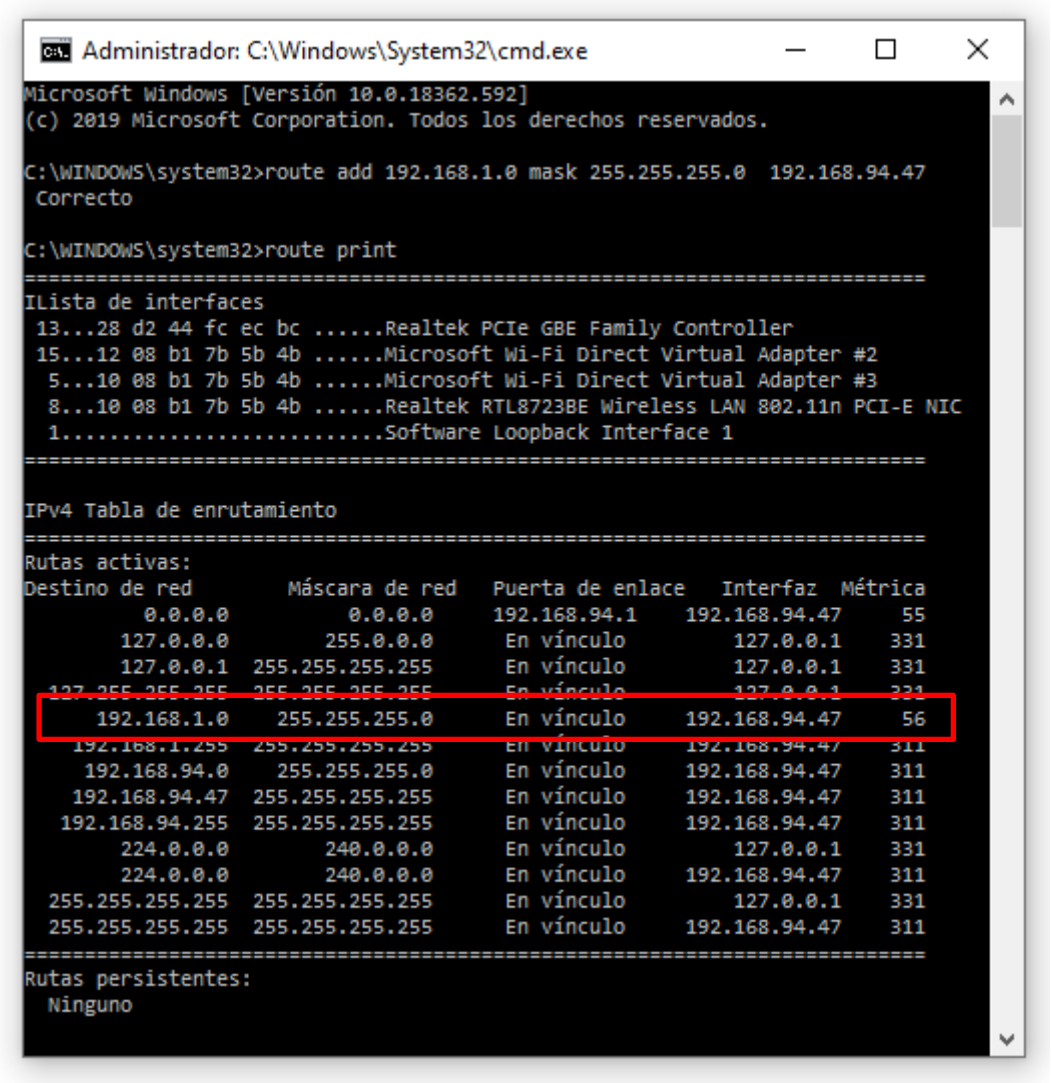


Figura E.21. Añadir una ruta a la tabla de encaminamiento.

Ahora ejecutamos el comando `route delete 192.168.1.0 mask 255.255.255.0`. Esto debería borrar la ruta que añadimos previamente. También podríamos borrar todas las rutas de la tabla con un solo comando, a saber, `route -f`. Pero debemos tener cuidado con este comando: Dependiendo del sistema operativo, de los protocolos usados, y de la configuración de red, esto podría detener todas las conexiones de red. Ejecutamos el comando `route print` para comprobar que, efectivamente, hemos eliminando la ruta indicada. Si tenemos problemas con la tabla de encaminamiento, deberíamos detener y reiniciar TCP/IP, o reiniciar el ordenador. Normalmente, cualquier ruta que añadamos se perderá si reiniciamos TCP/IP o el ordenador. Pero los rúters pueden añadir rutas de forma persistente usando la opción `-p` del comando `add`.

Recordemos que la finalidad última de las tablas de encaminamiento es permitir el encaminamiento de los paquetes hacia redes remotas. A modo de ejemplo más realista, consideremos la red ilustrada en el diagrama de la figura E.22. El diagrama muestra dos LANs: la red LAN A y la red LAN B. Inicialmente, los ordenadores en estas dos redes no podrán hablar entre sí porque están separados por rúters y por una nube (sea lo que sea lo que represente esa nube: una WAN, una interred, Internet, etc.). Para que dos ordenadores en esas LANs pueden comunicarse, debe especificarse una ruta a través de los rúters de ambas LANs. Cada rúter dispone de dos direcciones IP, una por cada interfaz. (A estas dos direcciones se las suele llamar dirección privada y dirección pública, respectivamente). Digamos que la máscara de subred en ambas LANs es 255.255.255.0.

En el rúter A deberíamos añadir una ruta hacia la LAN B escribiendo el comando `route add 10.253.253.0 mask 255.255.255.0 63.21.15.121`. Con esto definimos la ruta hacia la red LAN B 10.253.253.0 usando la dirección pública del rúter B, 63.21.15.121, que es la que se usa para conectar con otras redes remotas.

En el rúter B, deberíamos ejecutar el comando `route add 10.254.254.0 mask 255.255.255.0 63.21.15.128`. Con esto definimos la ruta a la red LAN A 10.254.254.0 usando la dirección pública del rúter A, 63.21.15.128.

Después de haber definido ambas rutas, los ordenadores en ambas redes podrán comunicarse.

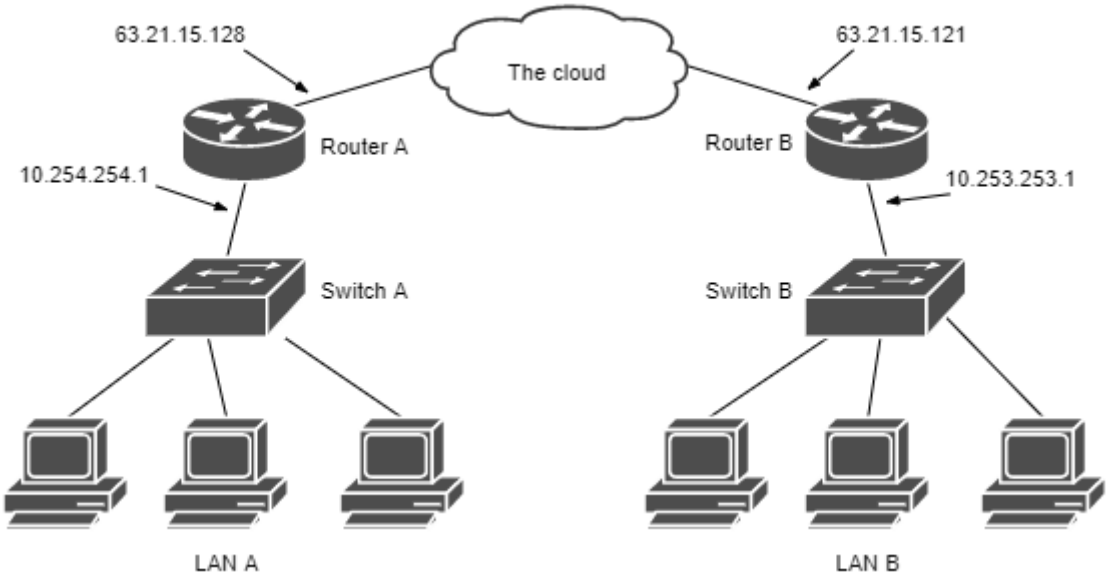


Figura E.22. Conexión de dos redes.

PRÁCTICA F (1): INTRODUCCIÓN A WIRESHARK.

F.1. ANALIZADORES DE PAQUETES.

En esta práctica vamos a ilustrar algunos de los conceptos estudiados en el capítulo 2 analizando los paquetes enviados y recibidos por nuestro ordenador. Para ello, podríamos proceder de dos formas: (1) La primera sería crear nuestra propia interred de laboratorio aislada, en la que podríamos enviar y recibir paquetes entre dos hosts cualesquiera. Aunque este método es el más instructivo, no siempre podremos permitirnos montar una red exclusiva para hacer pruebas con ella. (2) El segundo método consiste en usar la Internet mundial como nuestra red de pruebas. La ventaja de este método es que existen muchos analizadores de paquetes gratuitos que nos permiten capturar y examinar los paquetes intercambiados entre nuestro ordenador e Internet. Éste es el enfoque que usaremos aquí.

Un **analizador de paquetes** es un programa que debe ejecutarse en paralelo a la aplicación cuyos paquetes queremos analizar. Sin embargo, y antes de utilizar un analizador de paquetes, debemos entender adecuadamente el significado del término *paquete*. Como ya sabemos del capítulo 2, las comunicaciones a través de Internet se hacen en base a una arquitectura en cinco capas: La pila de protocolos TCP/IP. Podemos analizar los paquetes en las capas de aplicación, transporte, red, y enlace de datos. La capa física intercambia bits, no paquetes, y no podemos realizar este tipo de análisis aquí.

Los analizadores de paquetes no capturan paquetes en todas las capas de la pila TCP/IP, sino únicamente en la capa de enlace. Recordar que, cuando un ordenador emisor envía datos, los paquetes creados por todas las capas superiores se encapsulan en una trama a nivel de capa de enlace. Cuando el ordenador destino recibe los datos, los paquetes dirigidos a cualquier capa de nivel superior se desencapsulan de las tramas recibidas. Esto significa que un analizador de paquetes solo necesita capturar las tramas de capa de enlace para acceder a los paquetes de todas las capas superiores. Por esta razón, los analizadores de paquetes constan de dos componentes: El capturador de paquetes, y el analizador de paquetes propiamente dicho. El **capturador de paquetes** captura una copia de todas las tramas entrantes y salientes, y se las pasa al analizador de paquetes. A continuación, el **analizador de paquetes** extrae el mensaje y los diferentes encabezados contenidos en las tramas para su análisis.

Aunque en el capítulo 2 indicamos que el encapsulado (y el desencapsulado) comienza (y termina) en la capa de aplicación (ver figura 2.16), en realidad los paquetes pueden originarse en cualquier capa por encima de la capa de enlace. Como veremos, algunos protocolos de la capa de transporte y de la capa de red también necesitan intercambiar paquetes entre ellos. Estos paquetes se encapsulan en tramas de capa de enlace. Los analizadores de paquetes capturan todas las tramas entrantes y salientes, y muestran los encabezados de todos los protocolos usados en la comunicación. Pero el origen y el destino de los paquetes no es obligatoriamente la capa de aplicación. Las figuras E.1 (a) y E.1 (b) muestran dos ejemplos.

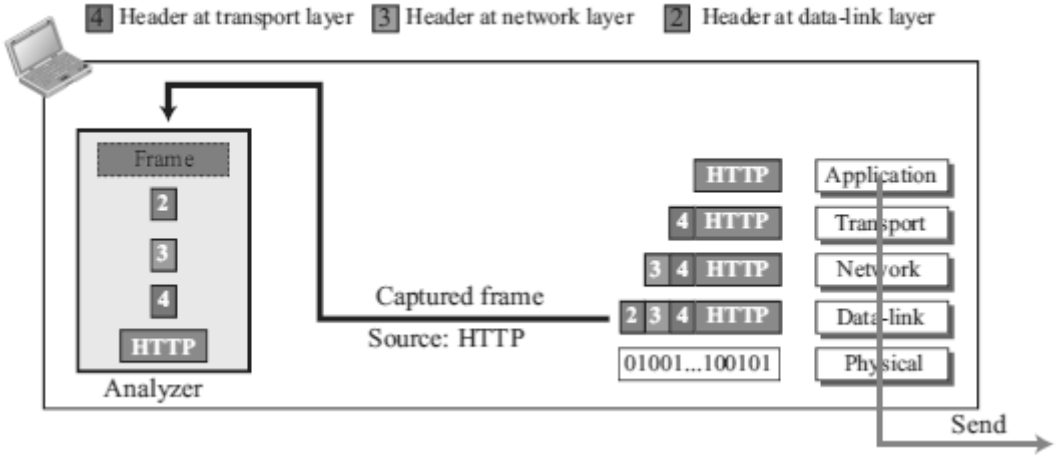


Figura F.1 (a). Una trama saliente con origen en la capa de aplicación.

En el ejemplo 1 el analizador captura una trama de salida (enviada). El origen de la trama es el protocolo HTTP en la capa de aplicación. El analizador se queda con una copia de la trama, y extrae de ella la información general de la trama (la caja punteada "Frame" en la figura), los encabezados de los protocolos de las capas 2, 3, y 4, y el mensaje HTTP para su posterior análisis. En el ejemplo 2 el analizador captura una trama de entrada (recibida). El destino final de la trama es el protocolo ARP en la capa de red. El analizador se queda con una copia de la trama, y extrae de ella la información general de la trama (la caja punteada "Frame" en la figura), el encabezado del protocolo de capa 2, y el mensaje ARP para su posterior análisis.

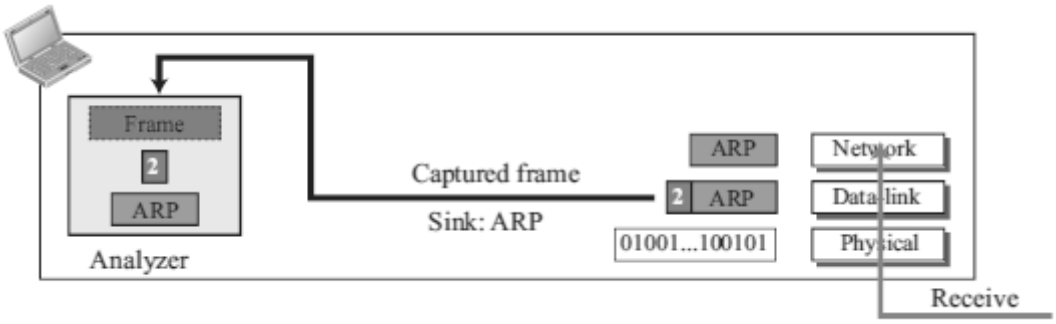


Figura F.1 (b). Una trama entrante con destino en la capa de red.

F.2. WIRESHARK.

¿QUÉ ES WIRESHARK?

En esta práctica vamos a usar un analizador de paquetes llamado **Wireshark** (https://www.wireshark.org/). Wireshark (antes conocido como Ethereal) es un analizador gratuito que captura paquetes de datos de un interfaz de red (de una tarjeta de red) y muestra la información detallada de los protocolos involucrados. Sin embargo, Wireshark es un analizador pasivo: Únicamente captura paquetes, sin manipularlos. Wireshark no es capaz de enviar paquetes ni de realizar otras operaciones activas.

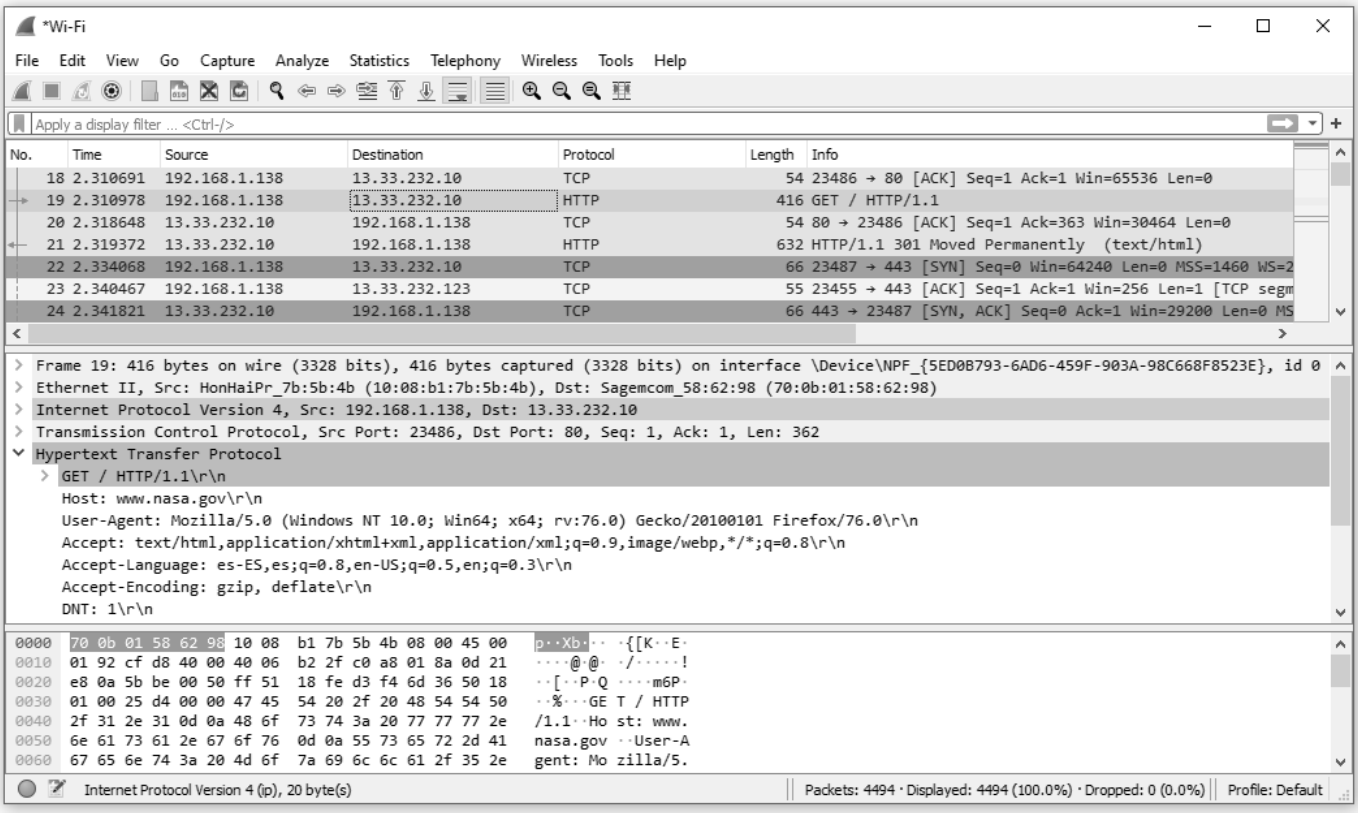


Figura F.2. La ventana principal de Wireshark.

LA VENTANA PRINCIPAL DE WIRESHARK.

La ventana principal de Wireshark consta de seis secciones importantes (ver figura F.2): La barra de menús, la barra de filtros, el panel de la lista de paquetes, el panel de los detalles del paquete, el panel de los bytes del paquete, y la barra de estado.

La **barra de menús** está formada por varios menús desplegables y por las barras de herramientas presentes en la mayoría de programas, ver figura F.3. El menú "File" sirve para realizar las típicas acciones de guardar, abrir, cerrar, imprimir, etc. El menú "Capture" se usa para empezar a capturar tramas. El menú "View" es útil para mostrar u ocultar algunas de las secciones de la ventana. Algunas de las acciones más importantes están disponibles en los botones de acceso directo bajo la barra de menús.

La **barra de filtros** nos permite mostrar los paquetes en los que estamos interesados y ocultar el resto. Cuando empieza a capturar paquetes, Wireshark captura y analiza cualquier trama entrante o saliente, independientemente del protocolo que la origine o la reciba. En ocasiones no querremos ver todas las tramas, y solo necesitaremos restringir el análisis a un protocolo específico. Por ejemplo, puede que solo queramos analizar los paquetes enviados o recibidos por el protocolo HTTP de la capa de aplicación, o los paquetes del protocolo ARP de la capa de red. En el lenguaje de los analizadores de paquetes, a esto se le llama **filtrado**. Para establecer un filtro antes o después de hacer una captura, basta con escribir el nombre del protocolo *en minúsculas* y pulsar en el botón "Apply" azul con la flecha blanca, ver figura F.3.

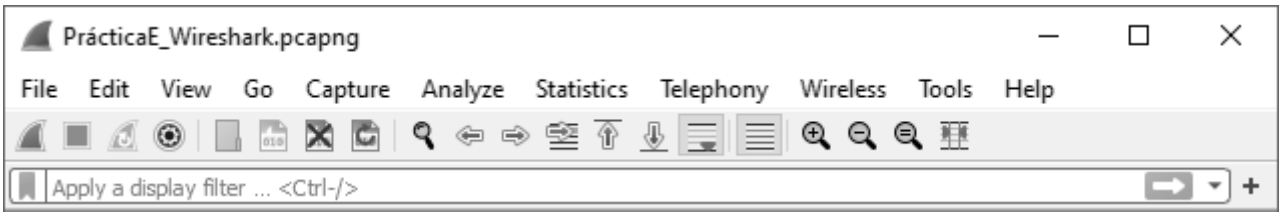


Figura F.3. Barra de menús y barra de filtros.

El **panel de la lista de paquetes** es un resumen de cada paquete capturado. Este resumen incluye el número del paquete (un número añadido por Wireshark que no forma parte del paquete), el instante en el que se capturó el paquete, las direcciones IP de origen y de destino del paquete, el protocolo origen y destino del paquete, e información adicional acerca de los contenidos del paquete, ver figura F.4. En otras palabras, este panel muestra las tramas capturadas que se pasarán al analizador para su análisis detallado.

No.	Time	Source	Destination	Protocol	Length	Info
7	2.256662	192.168.1.138	212.231.6.7	DNS	72	Standard query 0x505d A www.nasa.gov
8	2.280110	212.231.6.7	192.168.1.138	DNS	214	Standard query response 0x505d A www.nasa.gov CNAME www.nasawestprime.com CNAME
9	2.281243	192.168.1.138	13.33.232.10	TCP	66	23485 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
10	2.281647	192.168.1.138	212.231.6.7	DNS	89	Standard query 0xf170 A d30etcnkn29cv0.cloudfront.net
11	2.288653	13.33.232.10	192.168.1.138	TCP	66	80 → 23485 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=256
12	2.288753	192.168.1.138	13.33.232.10	TCP	54	23485 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
13	2.295524	212.231.6.7	192.168.1.138	DNS	153	Standard query response 0xf170 A d30etcnkn29cv0.cloudfront.net A 13.33.232.46 A
14	2.296654	192.168.1.138	212.231.6.7	DNS	89	Standard query 0x3f41 AAAA d30etcnkn29cv0.cloudfront.net
15	2.302854	192.168.1.138	13.33.232.10	TCP	66	23486 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
16	2.305646	212.231.6.7	192.168.1.138	DNS	313	Standard query response 0x3f41 AAAA d30etcnkn29cv0.cloudfront.net AAAA 2600:9000
17	2.310578	13.33.232.10	192.168.1.138	TCP	66	80 → 23486 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=256
18	2.310691	192.168.1.138	13.33.232.10	TCP	54	23486 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
19	2.310978	192.168.1.138	13.33.232.10	HTTP	416	GET / HTTP/1.1
20	2.318648	13.33.232.10	192.168.1.138	TCP	54	80 → 23486 [ACK] Seq=1 Ack=363 Win=30464 Len=0
21	2.319372	13.33.232.10	192.168.1.138	HTTP	632	HTTP/1.1 301 Moved Permanently (text/html)
22	2.334068	192.168.1.138	13.33.232.10	TCP	66	23487 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
23	2.340467	192.168.1.138	13.33.232.123	TCP	55	23455 → 443 [ACK] Seq=1 Ack=1 Win=256 Len=1 [TCP segment of a reassembled PDU]
24	2.341821	13.33.232.10	192.168.1.138	TCP	66	443 → 23487 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=256
25	2.341931	192.168.1.138	13.33.232.10	TCP	54	23487 → 443 [ACK] Seq=1 Ack=1 Win=65536 Len=0
26	2.343925	192.168.1.138	13.33.232.10	TLSv1…	571	Client Hello
27	2.347282	13.33.232.123	192.168.1.138	TCP	66	443 → 23455 [ACK] Seq=1 Ack=2 Win=127 Len=0 SLE=1 SRE=2
28	2.352060	13.33.232.10	192.168.1.138	TCP	54	443 → 23487 [ACK] Seq=1 Ack=518 Win=30464 Len=0
29	2.352758	13.33.232.10	192.168.1.138	TLSv1…	1514	Server Hello

Figura F.4. Panel de la lista de paquetes.

El **panel de los detalles del paquete** muestra el análisis detallado de cada trama capturada (ver figura F.5). Este panel solo muestra la información detallada de la trama que hayamos seleccionado en el panel de la lista de paquetes. Para elegir una trama, basta con pinchar sobre ella en la lista de paquetes. Al seleccionar una trama, el panel de detalles mostrará la información de la trama en la forma de una estructura en árbol. Para expandir cada rama, basta con pinchar en la flecha (>) al inicio de cada rama. Notar que el analizador muestra en primer lugar la información general de la trama. A continuación muestra la información contenida en los encabezados de todos los protocolos desde la capa de enlace hasta la capa origen o destino de la trama.

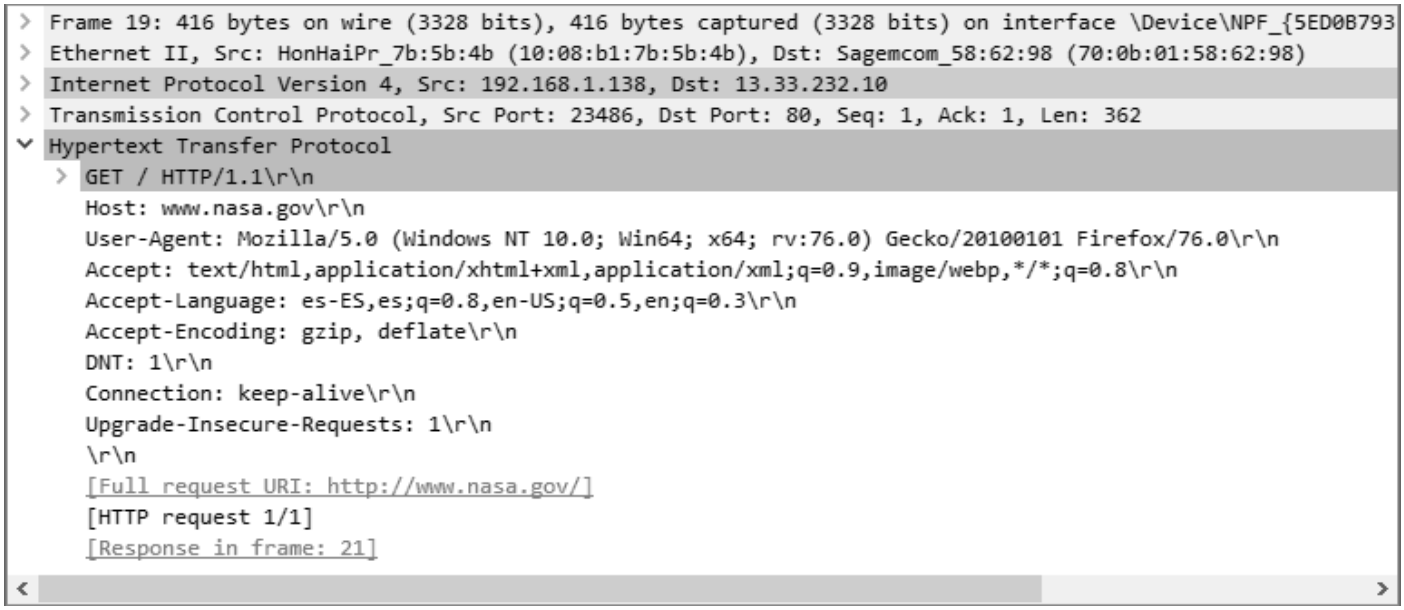


Figura F.5. Panel de detalles del paquete.

El **panel de los bytes del paquete** muestra la trama completa (aquella que hayamos elegido en la lista de tramas capturadas), en formato hexadecimal y ASCII (ver figura F.6). Si necesitamos el byte (o su equivalente ASCII) de una línea cualquiera del panel de detalles del paquete, basta con pinchar en la línea de interés en el panel de detalles, y la zona correspondiente en el panel de bytes se iluminará de azul.

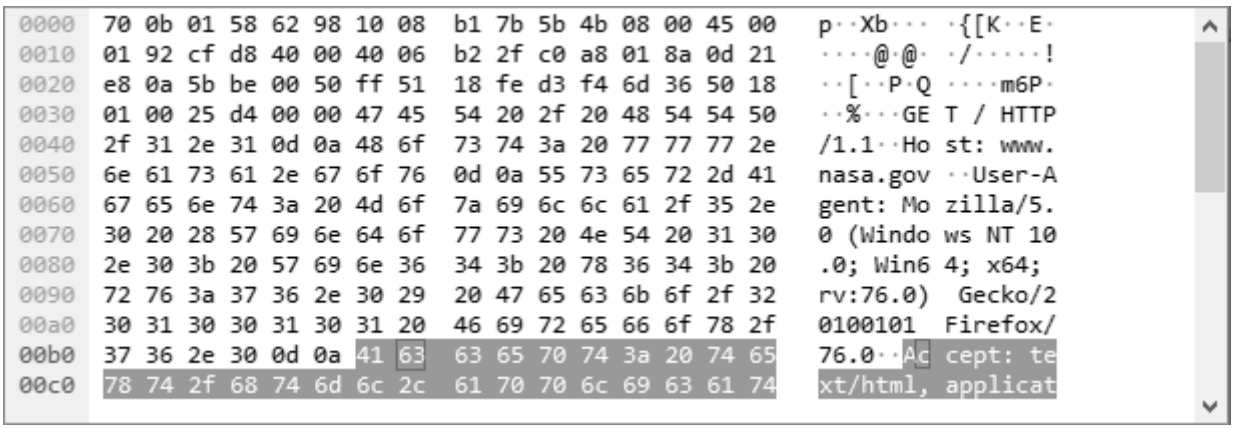


Figura F.6. Panel de los bytes del paquete.

F.3. COMENZAR A TRABAJAR CON WIRESHARK.

Cuando tengamos que realizar una práctica con Wireshark, hay un conjunto de acciones que siempre tendremos que repetir.

Comenzar la captura. Para empezar a capturar paquetes, desplegamos el menú "Capture" y pinchamos en "Options" para abrir la ventana de opciones de captura. Aquí hay un par de opciones que debemos

configurar antes de empezar a capturar paquetes: La pestaña "Input" muestra la lista de interfaces de red, en la que debemos seleccionar el interfaz (WiFi, Ethernet, etc.) del que queremos capturar tramas. Además, deberíamos tener habilitada la opción "Enable promiscuous mode on all interfaces", ver figura F.7.

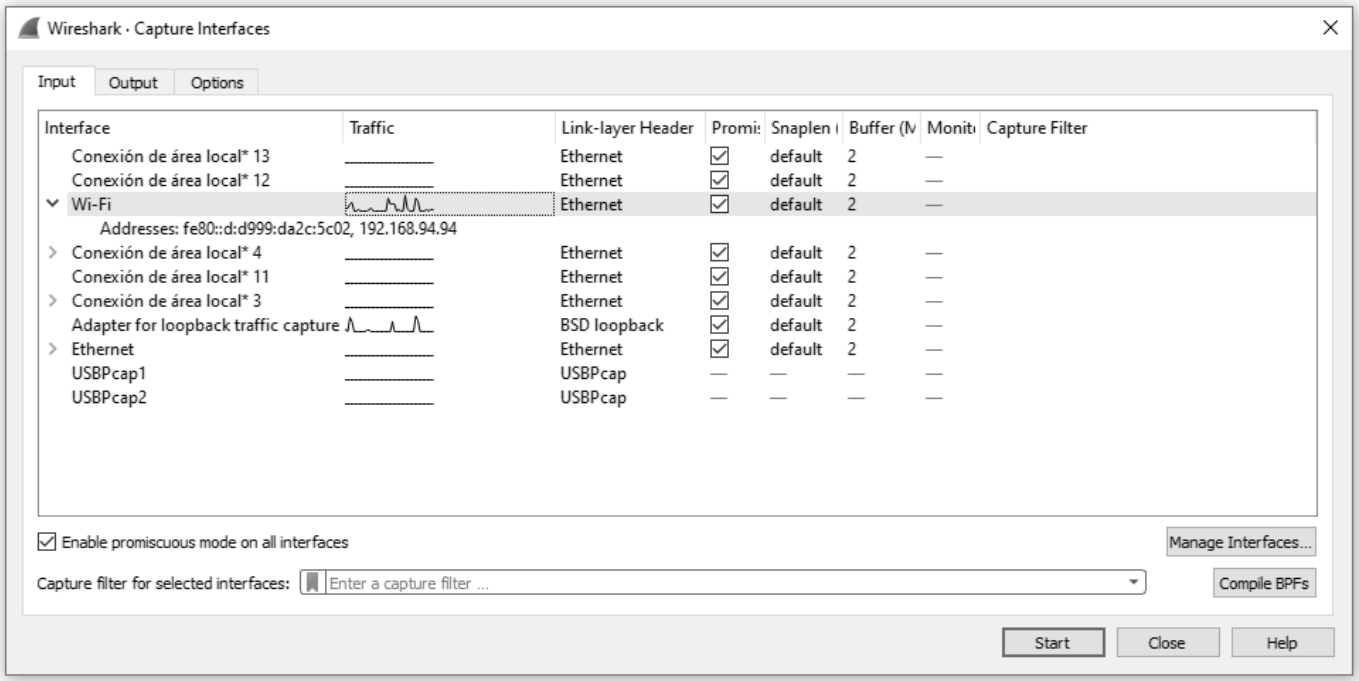


Figura F.7. Ventana de diálogo con las opciones de captura.

Después de haber configurado la captura, pulsamos en "Start". Wireshark empezará a capturar los paquetes intercambiados entre nuestro ordenador y la red a la que sté conectado (a través del interfaz seleccionado). Si después de unos instantes Wireshark no ha capturado ningún paquete, es porque hay algún problema.

Detener la captura. Cuando hayamos capturado todos los paquetes (tramas) que necesitamos para completar la tarea, podemos dejar de capturar. Para ello, vamos al menú desplegable "Capture" y seleccionamos la opción "Stop". Wireshark dejará de capturar tramas.

Guardar la captura. Después de haber terminado la captura, puede que queramos guardar la captura en un archivo para un uso posterior.

F.4. TRAMAS DE ENTRADA Y DE SALIDA.

Al observar la lista de las tramas capturadas, puede que nos preguntemos qué tramas son de entrada (recibidas) y qué tramas son de salida (enviadas). Esta información puede consultarse en la lista de las tramas capturadas. La lista de tramas muestra las direcciones de origen y de destino de cada trama. Si la dirección de origen es la dirección del ordenador con el que estamos trabajando (esta dirección se muestra en la ventana de opciones de captura cuando empezamos a capturar, ver figura F.7), la trama es de salida. Si la dirección de destino en la dirección de nuestro ordenador, esa trama es de entrada.

F.5. NUESTRA PRIMERA CAPTURA CON WIRESHARK.

En esta primera práctica sólo vamos a probar el software Wireshark para hacer nuestra primera captura. En concreto, vamos a conectarnos a una página web y a capturar los paquetes intercambiados.

1) Para empezar, arranca el navegador web y borra la memoria caché. (Si no sabes cómo hacerlo, conéctate a la web https://www.wikihow.com/Clear-Your-Browser%27s-Cache).

2) A continuación, borra la tabla ARP de tu ordenador. Esto ya deberías saber cómo hacerlo.
3) Ahora abre Wireshark y empieza a capturar paquetes. Después vuelve al navegador, y conéctate a una de tus páginas web favoritas.
4) Detén la captura, y guarda el archivo capturado con el nombre PrácticaF_Wireshark.
5) Usa un filtro para mostrar únicamente tramas capturadas en las que el protocolo origen o destino sea HTTP. Recordar que en la barra de filtros debemos escribir http en minúsculas.
6) Accede a la primera trama cuyo protocolo origen sea HTTP, y responde a las siguientes preguntas:
   - ¿Es una trama de entrada o de salida? Justifica tu respuesta.

   - Para el segmento de red donde se capturó la trama, ¿Cuál es la dirección física de origen? ¿Cuál es la dirección física de destino?

   - ¿Cuál es la dirección IP de origen de la trama? ¿Cuál es la dirección IP de destino de la trama?

   - ¿Cuál es el tiempo de vida del paquete IP (TTL = Time to Live)?[15]

   - ¿Cuál es el puerto de origen y el puerto de destino del paquete TCP?

   - ¿Cuál es el número total de bytes de la trama completa?
     PISTA: En el panel de detalles, expande la rama "Frame" y busca el número total de bytes transmitidos por el cable.

   - ¿Cuántos bytes ocupa el encabezado del protocolo de capa de enlace (Ethernet)?
     PISTA: Expande la rama del protocolo de capa de red IP y busca el número de bytes que ocupa el paquete IP. Si a la longitud total de la trama Ethernet le restas la longitud del paquete IP que encapsula, la diferencia te dará el tamaño del encabezado Ethernet.

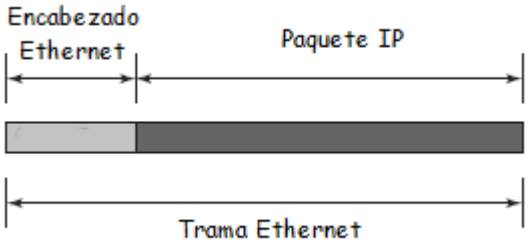


Figura F.8. Encapsulado de un paquete IP en una trama Ethernet.

   - ¿Cuántos bytes ocupa el encabezado del protocolo de red (IP)?
     PISTA: De forma análoga a como lo hiciste antes, expande la rama del protocolo de capa de transporte TCP. A continuación, resta al tamaño del paquete IP el tamaño del paquete TCP que encapsula.

Para justificar tus respuestas, haz una serie de capturas de pantalla que muestren la trama capturada en la lista de paquetes, y los distintos detalles de la trama elegida.

[15] El tiempo de vida (TTL =Time To Live) es un campo del encabezado IP que indica por cuántos saltos puede pasar un paquete antes de ser descartado por la red o devuelto a su origen. El valor se inicializa en el emisor, y se reduce en una unidad por cada salto que realiza.

PRÁCTICA F (2). ANÁLISIS DE UN PING CON WIRESHARK.

F.6. PLANTEAMIENTO.

En esta segunda práctica de Wireshark vamos a capturar y analizar paquetes del protocolo de capa de red ICMP. Para ello, sigue los siguientes pasos:

1) Abre la ventana de comandos de Windows, y ejecuta una serie de `ping`'s continuos a otro ordenador de la red en la que te encuentres, a tu rúter (puerta de enlace predeterminada), o a la dirección IP de una de tus páginas web favoritas. En mi caso, he ejecutado el comando `ping -t 192.168.1.1` al rúter inalámbrico de mi casa (ver figura F.9).

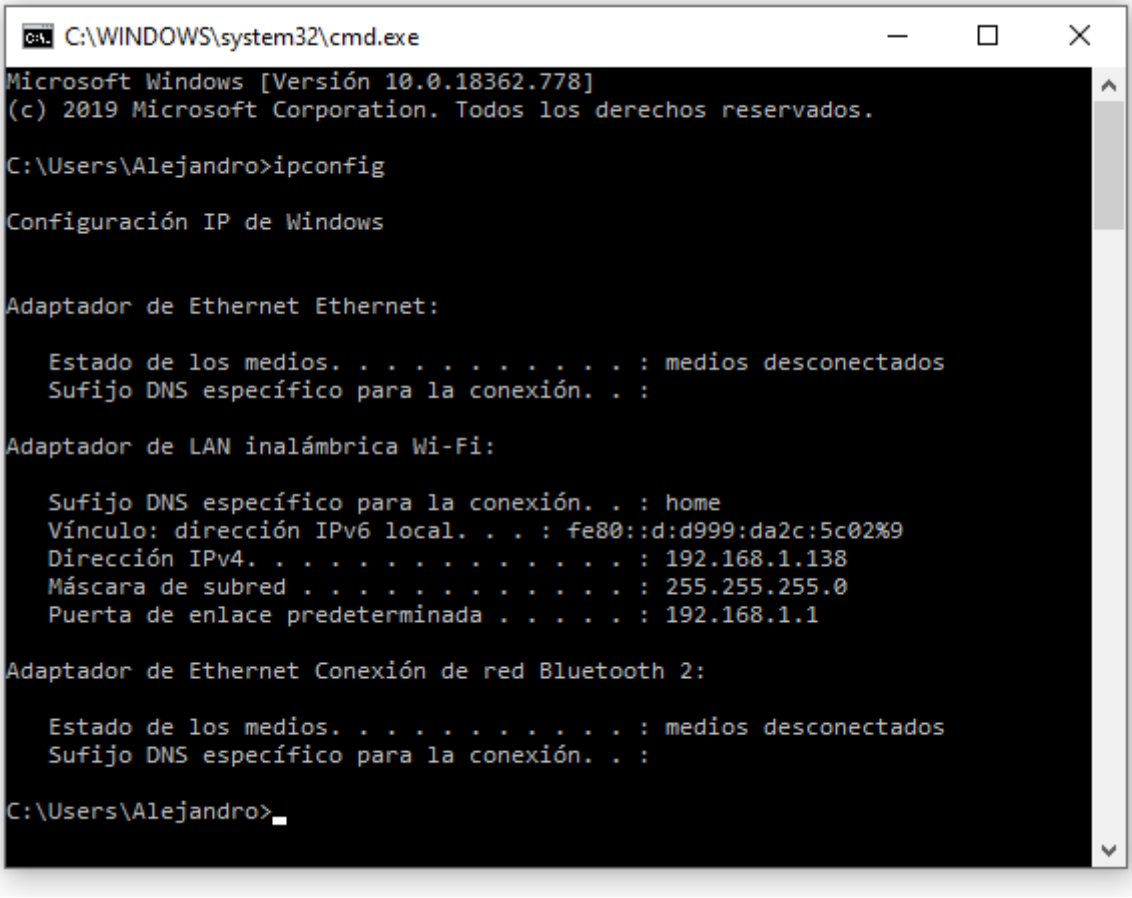


Figura F.9. Configuración IP de mi equipo.

Recordemos que el comando `ping` utiliza el **protocolo ICMP** (Internet Control Message Protocol). Este protocolo de capa de red permite que un host pueda enviar paquetes de prueba a otro host para comprobar si puede conectar con él.

Verifica que recibes respuestas de la IP destino elegida, y deja la ventana de comandos abierta mientras continúa haciendo `ping`'s al otro dispositivo.

2) Seguidamente abre Wireshark, filtra para capturar únicamente paquetes ICMP, e inicia la captura de paquetes.

3) Después de unos 20 o 30 segundos, detén la captura (ver figura F.10).

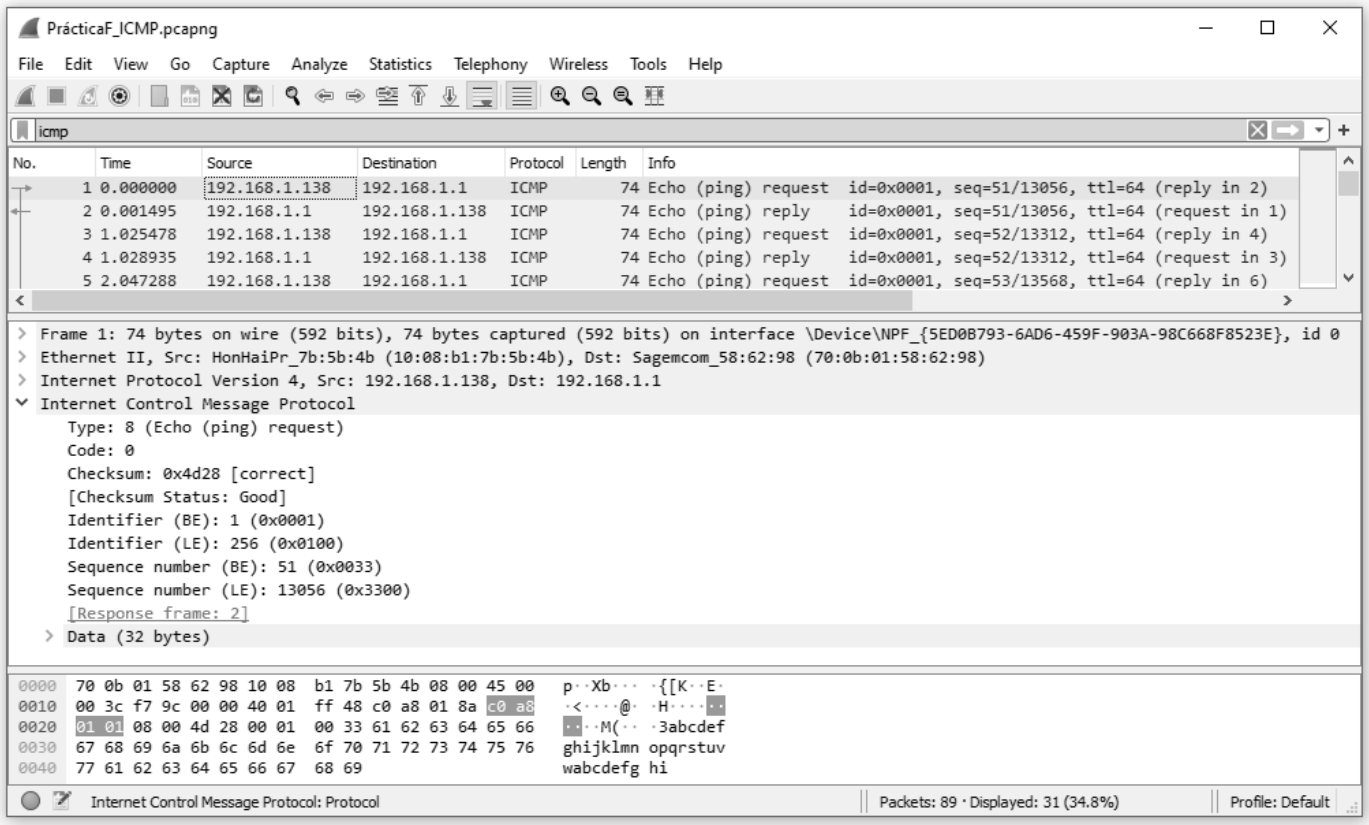


Figura F.10. Captura de paquetes ICMP en un `ping` continuo.

4) Observa la lista de paquetes capturados, y selecciona un paquete ICMP.

5) Ahora echa un vistazo a los detalles del paquete seleccionado, protocolo a protocolo. Recuerda que el origen y el destino de los paquetes no es necesariamente la capa de aplicación. De hecho, los paquetes ICMP se originan y se reciben en la capa de red, por lo que las tramas capturadas solo encapsularán información de los protocolos en las capas de enlace de datos (Ethernet) y de red (ICMP e IP).

   - Comienza expandiendo los detalles correspondientes al protocolo ICMP, y observa toda la información contenida, por ejemplo, si es un paquete de solicitud (request) o de respuesta (reply), el tamaño del paquete, los bits de detección de errores (checksum) y su estado, etc.

   - Expande los detalles correspondientes al protocolo IP, como la versión de IP usada, las direcciones IP de origen y destino, la longitud del paquete, el checksum, etc.

   - Expande los detalles correspondientes al protocolo Ethernet. Observa la información contenida, como las direcciones físicas de origen y de destino.

   - Por último, expande los detalles correspondientes a la trama, como el número de la trama, el tamaño de la trama, protocolos encapsulados en la trama, etc.

F.7. TAREAS A REALIZAR.

ANÁLISIS DEL PROCESO DE ENCAPSULADO.

Vamos a analizar el proceso de encapsulado de un paquete ICMP. Todo el proceso comienza en el programa del símbolo del sistema, enviando un `ping` de 32 bytes (un mensaje ICMP). Aunque ICMP es un protocolo

de capa de red, sus mensajes no se pasan directamente a la capa de enlace. En vez de eso, los mensajes ICMP se encapsulan dentro de paquetes IP antes de pasárselos a la capa inferior (ver figura F.11).

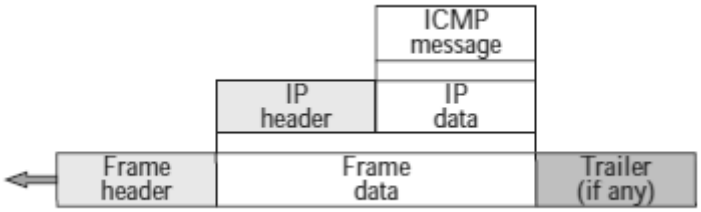


Figura F.11. En capsulado de un mensaje ICMP.

Selecciona un paquete ICMP de salida (esto es, un paquete enviado desde tu ordenador al otro dispositivo), abre el desplegable del protocolo IP, y responde a las siguientes preguntas:

a) ¿Cuál es el tamaño total en bytes del paquete IP? ¿Cuál hemos dicho que es el tamaño del mensaje ICMP?

b) A la vista de las respuestas a las preguntas previas, ¿cuál es el tamaño total del encabezado IP que añade la capa de red?

A continuación, el datagrama IP que contiene el mensaje ICMP se envía a la capa de enlace (al adaptador de red), que lo encapsula dentro de una trama Ethernet. Finalmente la trama se envía a la capa física, que la trasmite como un flujo continuo de bits. Abre el desplegable de los detalles de la trama (Frame) y responde a las siguientes preguntas:

c) ¿Cuál es el tamaño total en bytes y en bits de la trama (que a su vez, son los bytes y bits que envía la capa física a través del enlace)?

d) A la vista de la respuesta a la pregunta previa, ¿cuál es el tamaño total del encabezado y del terminador Ethernet que añade la capa de enlace de datos?

e) Compara el tamaño en bytes del paquete ICMP con el que comienza el proceso de encapsulado, y el tamaño en bytes de la trama que finalmente se envía a nivel de capa física. ¿Cuántos bytes han añadido en total los sucesivos protocolos al mensaje ICMP original?

ANÁLISIS DE LOS PAQUETES DE SOLICITUD Y DE RESPUESTA.

El procedimiento para enviar un `ping` a otro ordenador es muy sencillo, y básicamente consiste en el envío de un mensaje ICMP de solicitud (request), seguido de la recepción de un mensaje ICMP de respuesta (reply) o de un mensaje de error (debido a diferentes causas).

Paquete de solicitud.

En la lista de paquetes, selecciona un paquete ICMP de solicitud (request), y responde a las siguientes preguntas:

a) ¿Cuál es la dirección MAC de origen? Comprueba que esta dirección física coincide con la de tu ordenador.

b) ¿Cuál es la dirección MAC de destino? ¿Esta dirección es necesariamente la dirección MAC del dispositivo al que estás haciendo un `ping`? Responde a esta segunda pregunta considerando dos

situaciones: (1) el `ping` lo has hecho a un ordenador de la red LAN en la que te encuentras, y (2) el `ping` lo has hecho a un servidor web externo.

c) ¿Cuál es la dirección IP de origen? Comprueba que esta dirección lógica coincide con la de tu ordenador.

d) ¿Cuál es la dirección IP de destino? ¿Esta dirección es necesariamente la dirección IP del dispositivo al que estás haciendo un `ping`?

Paquete de respuesta.

En la lista de paquetes, selecciona el paquete ICMP de respuesta (reply) al paquete de solicitud previo, y responde a las siguientes preguntas:

e) ¿Cuál es la dirección MAC de origen? ¿Con qué dirección MAC del paquete de solicitud coincide?

f) ¿Cuál es la dirección MAC de destino? ¿Con qué dirección MAC del paquete de solicitud coincide?

g) ¿Cuál es la dirección IP de origen? ¿Con qué dirección IP del paquete de solicitud coincide?

h) ¿Cuál es la dirección IP de destino? ¿Con qué dirección IP del paquete de solicitud coincide?

i) ¿Cuánto tiempo ha tardado en llegar el paquete de respuesta?

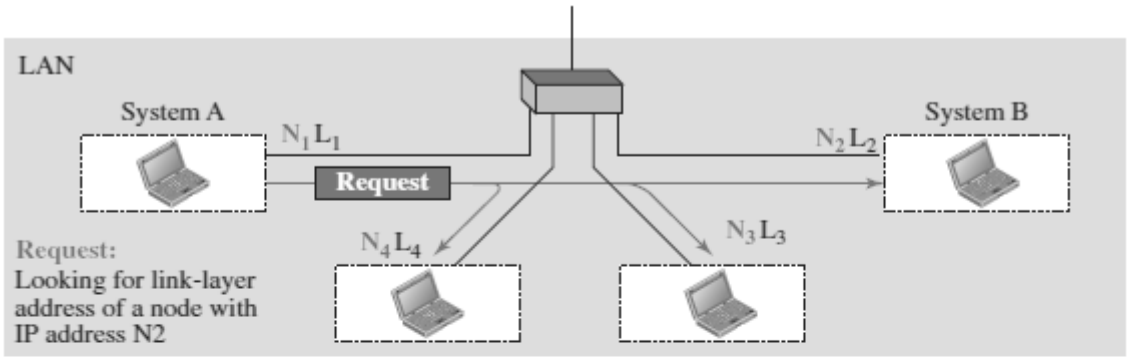
PRÁCTICA F (3). ANÁLISIS DEL PROTOCOLO ARP CON WIRESHARK.

F.8. EL PROTOCOLO ARP.

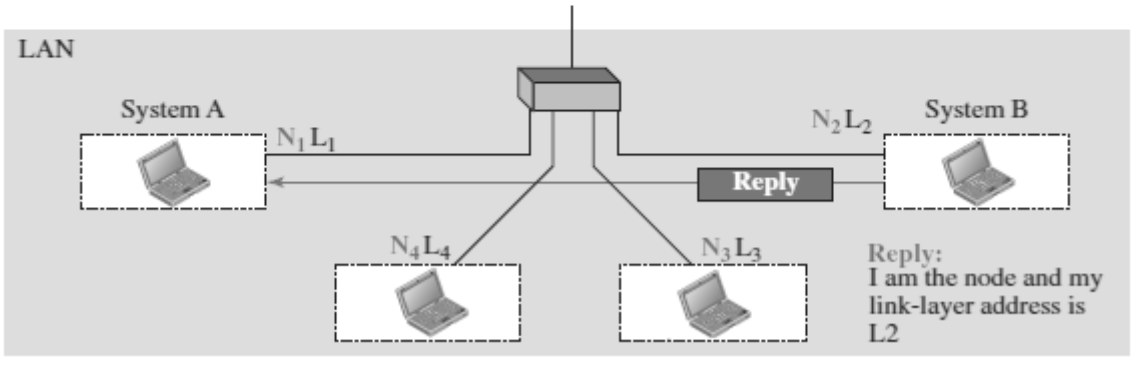
FUNCIONAMIENTO DE ARP.

Siempre que un nodo de una red tiene que enviar un datagrama IP a otro nodo de esa red, el nodo emisor dispone de la dirección IP del nodo receptor. Por ejemplo, el host de origen conoce la dirección IP del rúter por defecto. Cada rúter (excepto el último) obtiene de su tabla de encaminaamiento la dirección IP del siguiente rúter de la ruta hasta el host de destino final. Y el último rúter conoce la dirección IP del host de destino (porque es la dirección de destino del datagrama IP). Sin embargo, la dirección IP del siguiente rúter no nos sirve para mover una trama a través de la red a la que está conectado; lo que necesitamos es su dirección física. Aquí es donde entra en juego el **Protocolo de Resolución de Direcciones** (**ARP** = Address Resolution Protocol). El protocolo ARP recibe una dirección IP del protocolo IP, obtiene la dirección física correspondiente, y se la pasa a la capa de enlace.

Cuando un host o un rúter necesitan determinar la dirección física de otro host o rúter dentro de su red, consultan su **tabla ARP**. Cada entrada de la tabla ARP almacena una dirección IP y su correspondiente dirección física. Así pues, el nodo emisor intenta localizar una entrada en la tabla ARP para la dirección IP del nodo destino. Si esa entrada existe en la tabla, el nodo usa la dirección física asociada como dirección física de la trama a enviar. Pero si el nodo emisor no tiene una entrada en su tabla para la dirección IP de destino, envía un **paquete de solicitud ARP**. Este paquete incluye la dirección física y la dirección IP del nodo emisor, y la dirección IP del nodo receptor. Como el emisor no conoce la dirección física del receptor, la solicitud se difunde a todotoda la red usando la dirección física de broadcast (que en el caso de una LAN Ethernet, siempre es FF:FF:FF:FF:FF:FF).



a. ARP request is broadcast



b. ARP reply is unicast

Figura F.12. Operación de ARP.

Todos los hosts y los rúters de la red reciben y procesan el paquete de solicitud ARP, pero solo el destinatario correcto reconoce su dirección IP y envía un **paquete de respuesta ARP**. El paquete de respuesta contiene las direcciones IP y física del nodo receptor. Este paquete de respuesta se envía únicamente al nodo que envió el paquete de solicitud ARP.

En la figura F.12(a), el host A tiene un paquete que debe enviar al host B, cuya dirección IP es N_2. El sistema A pasa el paquete a su capa de enlace para efectuar la entrega, pero esta capa no conoce la dirección física de B. Por consiguiente, el host A le pide al protocolo ARP que envíe un paquete broadcast de solicitud ARP, para preguntar por la dirección física del host cuya dirección IP es N_2.

Este paquete llega a todos los nodos de la red, pero este paquete solo lo procesa el host B, porque su dirección IP coincide con la dirección IP de destino del paquete ARP. El host B contesta enviando un paquete unicast de respuesta ARP que incluye su dirección física (L_2), ver figura F.12(b). Tras recibir la respuesta, el host A puede enviar todos los paquetes que tiene para el host B, usando la dirección física que recibió desde B.

FORMATO DEL PAQUETE ARP.

La figura F.13 muestra el formato de un paquete ARP. El significado de los distintos campos es fácil de interpretar a partir de su nombre:

- El campo **tipo de hardware** (Hardware type) define el tipo de protocolo de capa de enlace empleado (por ejemplo, el protocolo Ethernet se identifica con un 1 en hexadecimal: 0x0001).

- El campo **tipo de protocolo** (Protocol type) define el tipo de protocolo de capa de red empleado (por ejemplo, IP se codifica con un 8 en hexadecimal: 0x0800).

- Los campos de **dirección del hardware origen** y **dirección del protocolo origen** (Source hardware address y Source protocol address) son campos de longitud variable que definen las direcciones física e IP del nodo emisor, respectivamente.

- Análogamente, los campos de **dirección del hardware destino** y de **dirección del protocolo destino** (Destination hardware address y Destination protocol address) definen las direcciones física e IP del nodo receptor, respectivamente. El campo de dirección del hardware destino estará vacío (todo ceros) en los paquetes de solicitud ARP, y relleno en los paquetes de respuesta ARP.

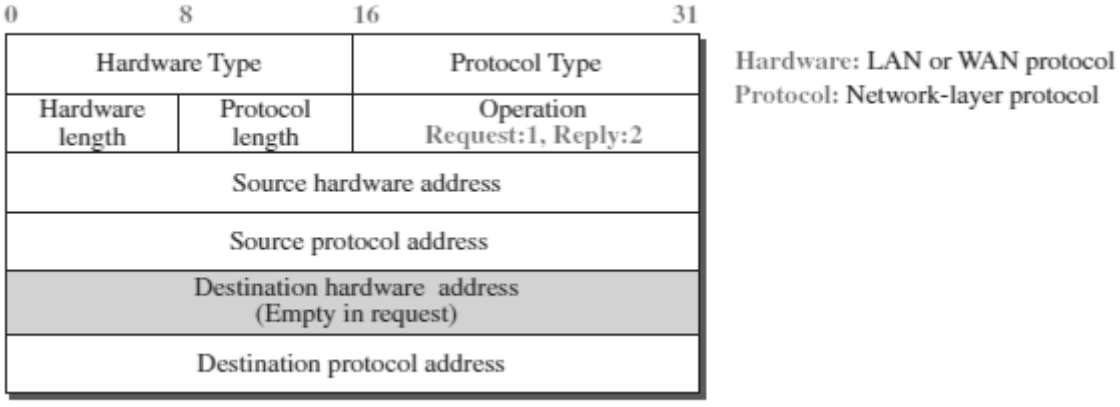


Figura F.13. Paquete ARP.

A modo de ejemplo, suponer que un host con dirección IP N_1 y dirección física L_1 tiene un paquete que enviar a otro host con dirección IP N_2 y dirección física L_2 (dirección física que el host emisor desconoce). Los dos host pertenecen a la misma red. La figura F.14 muestra los mensajes ARP de solicitud y de respuesta.

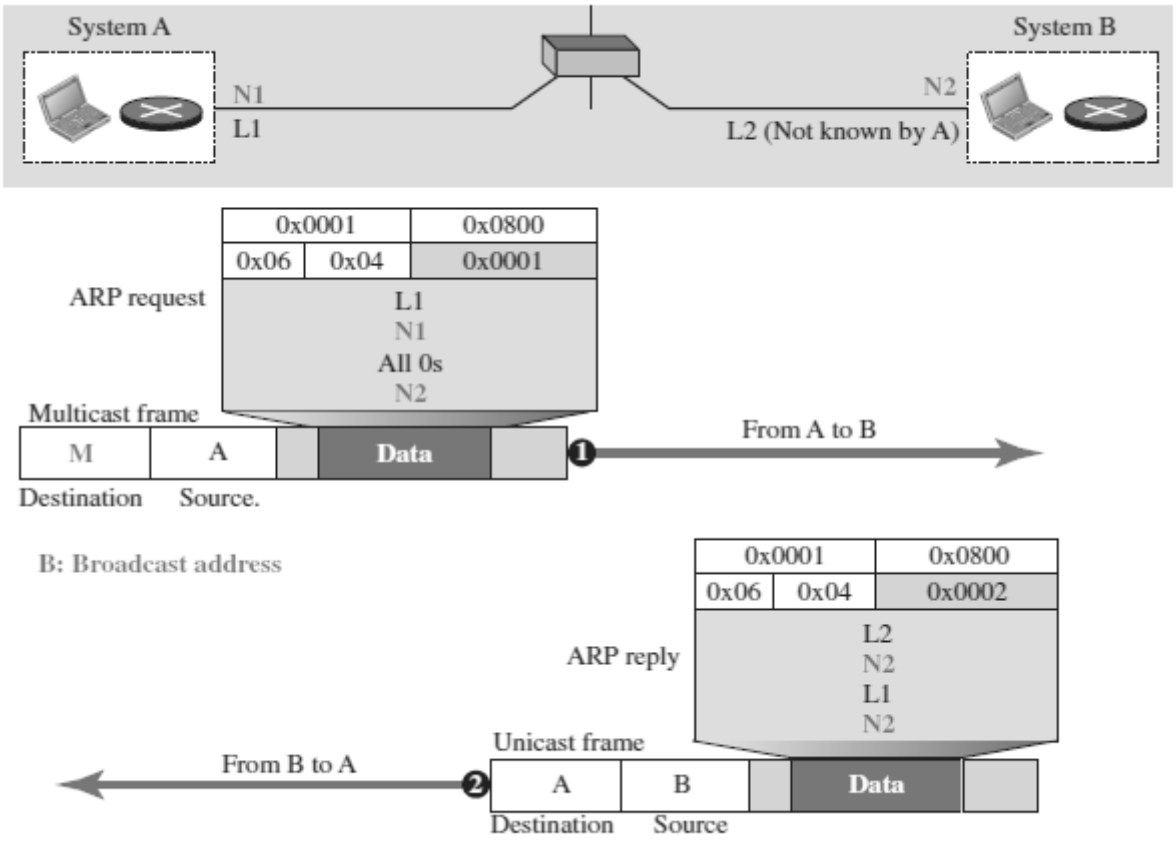


Figura F.14. Paquetes ARP de solicitud y de respuesta.

F.9. PLANTEAMIENTO DEL PROBLEMA.

En esta práctica vamos a capturar y a examinar con Wireshark paquetes ARP. Para ello, seguimos estos pasos:

- Cerramos el navegador o cualquier otro programa que pueda establecer conexiones en segundo plano.
- Arrancamos Wireshark y empezamos a capturar paquetes.
- Abrimos la ventana de comandos de Windows en **modo administrador**.
- Consultamos nuestra dirección IP, nuestra dirección física, y la dirección IP de la puerta de enlace predeterminada (el rúter).
- Consultamos la tabla ARP de nuestro ordenador mediante el comando `arp -a`.
- Limpiamos la tabla ARP de nuestro ordenador con el comando `arp -d *`. Nos aseguramos que en la tabla ARP ya no hay una entrada asociada a la puerta de enlace predeterminada.
- Rápidamente (antes de que la tabla ARP añada automáticamente una entrada para la puerta de enlace predeterminada), hacemos un `ping` a la dirección IP de la puerta de enlace predeterminada.

- Tras unos segundos, dejamos de capturar paquetes y guardamos el archivo de la captura. En la barra de filtros de Wireshark escribimos arp (en minúsculas) y pulsamos en "Apply".

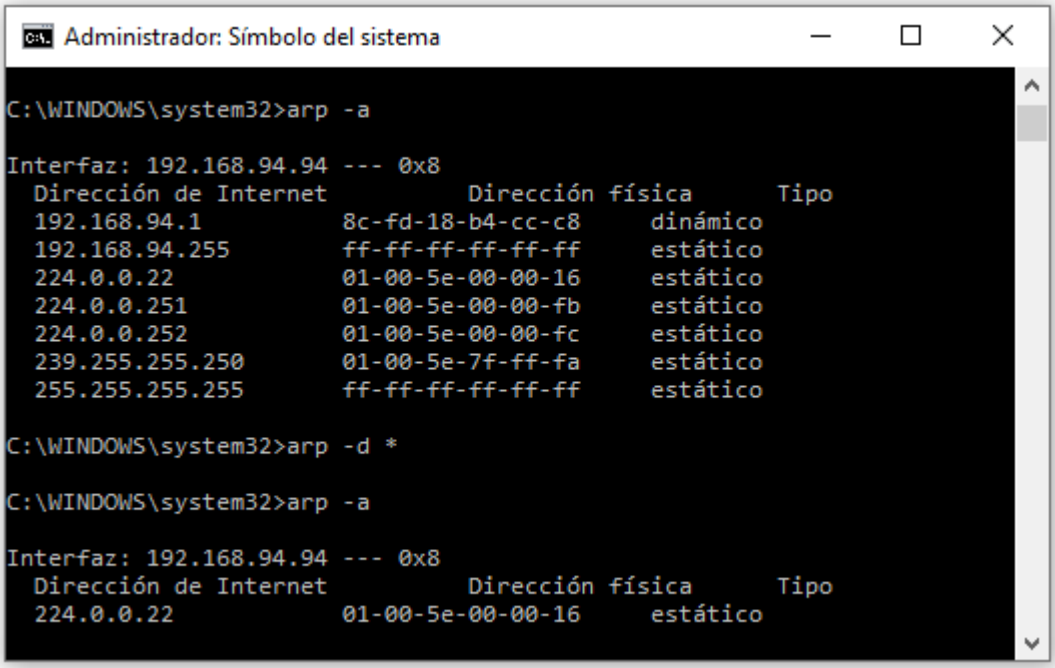


Figura F.15. Consulta y borrado de la tabla ARP.

F.10. ANÁLISIS DE UN PAQUETE DE SOLICITUD ARP.

En el panel de la lista de paquetes, seleccionamos el primer paquete de difusión que se corresponda con una **solicitud ARP**.

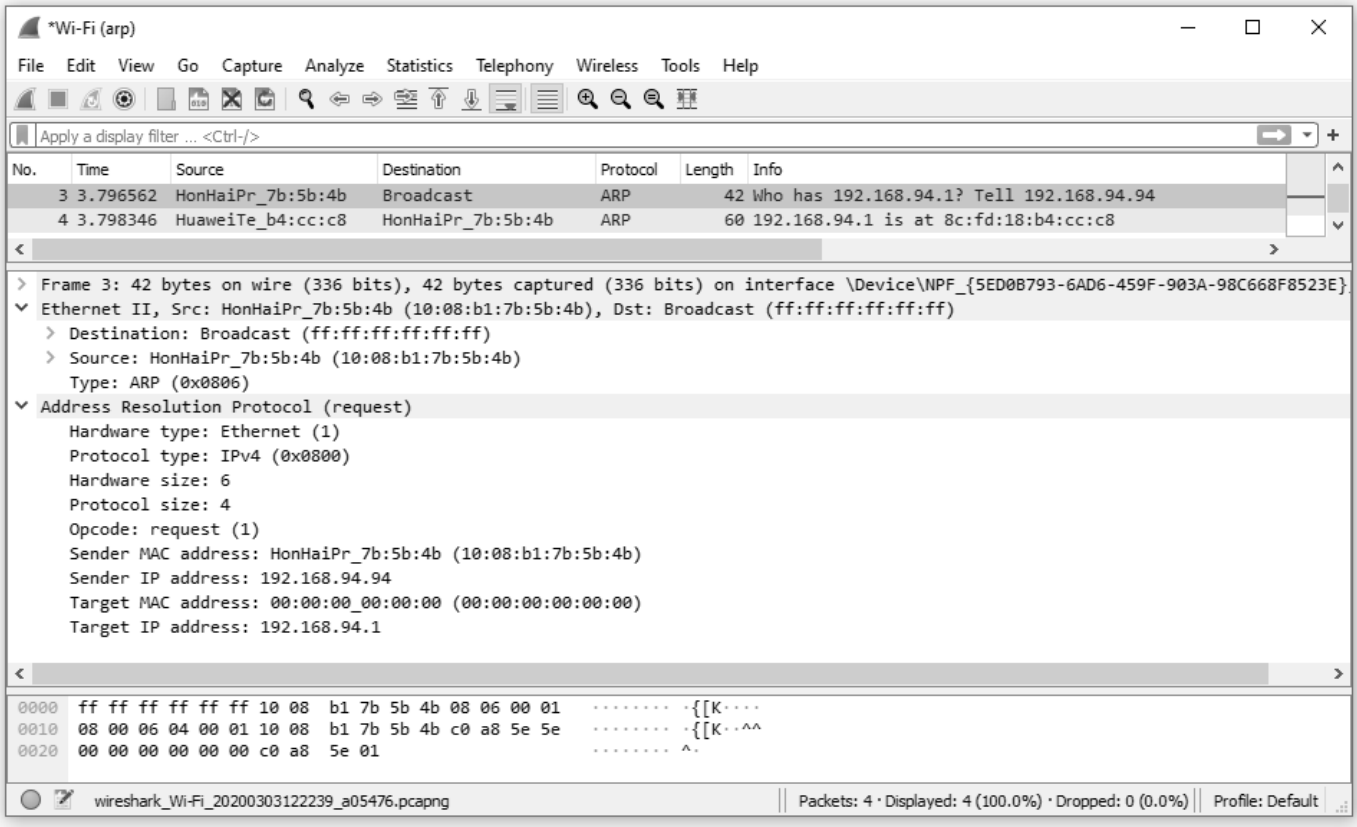


Figura F.16. Captura de un paquete de solicitud ARP.

Ayudándonos de la información obtenida de la ventana de comandos, de la captura Wireshark, y de las figuras F.13 y F.14, respondemos a las siguientes preguntas:

Ventana de comandos:

a) ¿Cuál es la dirección IP de tu ordenador?
b) ¿Cuál es la dirección física de tu ordenador?
c) ¿Cuál es la dirección IP de la puerta de enlace predeterminada?

Captura de Wireshark (detalles del desplegable "Ethernet II"):

d) ¿Cuál es la dirección física del nodo emisor?
e) ¿Cuál es la dirección física del nodo receptor? ¿Qué tipo de dirección física es (unicast o broadcast)?

Captura de Wireshark (detalles del desplegable "Address Resolution Protocol"):

f) ¿Cuál es el valor del campo *Hardware type* del paquete ARP? Indica su valor hexadecimal tal y como se muestra en el panel de los bytes del paquete, y su significado tal y como se muestra en el panel de detalles del paquete.
g) ¿Cuál es el valor del campo *Protocol type* del paquete ARP? Indica su valor hexadecimal y su significado.
h) ¿Cuál es el valor de los campos *Hardware length* y *Protocol length*? ¿Qué significan?
i) ¿Cuál es el valor del campo Operation (OpCode)? ¿Qué significa?
j) ¿Cuál es la dirección física (Hardware address) del nodo emisor? ¿Con qué dispositivo se corresponde?
k) ¿Cuál es la dirección IP (Protocol address) del nodo emisor? ¿Con qué dispositivo se corresponde?
l) ¿Cuál es la dirección física (Hardware address) del nodo receptor? ¿Qué significa esta dirección?
m) ¿Cuál es la dirección IP (Protocol address) del nodo receptor? ¿Con qué dispositivo se corresponde?

F.11. ANÁLISIS DE UN PAQUETE DE RESPUESTA ARP.

A continuación seleccionamos el paquete de **respuesta ARP** al paquete de solicitud ARP previo.

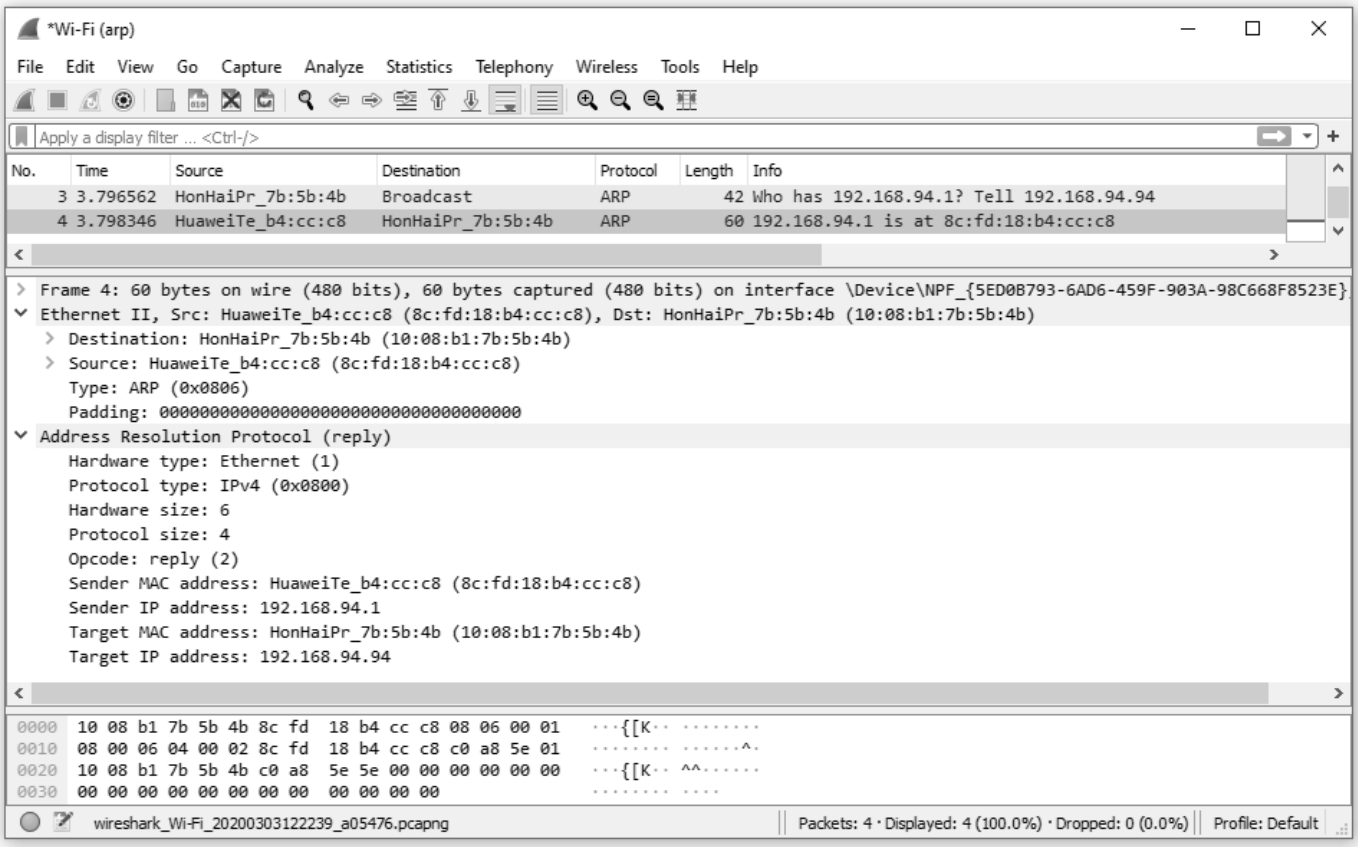


Figura F.17. Captura de un paquete de respuesta ARP.

De nuevo, y a partir de la información de la trama capturada, respondemos a estas preguntas:

Captura de Wireshark (detalles del desplegable "Ethernet II"):

a) ¿Cuál es la dirección física del nodo emisor?
b) ¿Cuál es la dirección física del nodo receptor? ¿Qué tipo de dirección física es (unicast o broadcast)?

Captura de Wireshark (detalles del desplegable "Address Resolution Protocol"):

c) ¿Cuál es el valor del campo *Hardware type* del paquete ARP? Indica su valor hexadecimal y su significado.
d) ¿Cuál es el valor del campo *Protocol type* del paquete ARP? Indica su valor hexadecimal y su significado.
e) ¿Cuál es el valor de los campos *Hardware length* y *Protocol length*? ¿Qué significan?
f) ¿Cuál es el valor del campo Operation (OpCode)? ¿Qué significa?
g) ¿Cuál es la dirección física (Hardware address) del nodo emisor? ¿Con qué dispositivo se corresponde?
h) ¿Cuál es la dirección IP (Protocol address) del nodo emisor? ¿Con qué dispositivo se corresponde?
i) ¿Cuál es la dirección física (Hardware address) del nodo receptor? ¿Qué significa esta dirección?
j) ¿Cuál es la dirección IP (Protocol address) del nodo receptor? ¿Con qué dispositivo se corresponde?

PRÁCTICA F (4). ANÁLISIS DE UNA CONEXIÓN HTTP CON WIRESHARK.

3. DISPOSITIVOS DE CONEXIÓN.

3.1. INTRODUCCIÓN.

Los hosts y las redes no suelen trabajar aislados. Lo habitual es que estén conectados unos a otros, o a Internet. Para interconectar hosts o redes se usan diferentes **dispositivos de conexión**. En este capítulo discutiremos los dispositivos de conexión más comunes, a saber, repetidores, concentradores (hubs), puentes (bridges), conmutadores (switches), y rúters. Los distintos dispositivos de conexión operan en diferentes capas de la pila TCP/IP: Los repetidores y los concentradores operan en la primera capa de la pila TCP/IP; los puentes y conmutadores operan en las dos primeras capas; y los rúters operan en las tres primeras capas (ver figura 3.1). El objetivo de este capítulo es presentar estos dispositivos, conocer su funcionamiento, y entender qué misión cumplen en una red o en una interred.

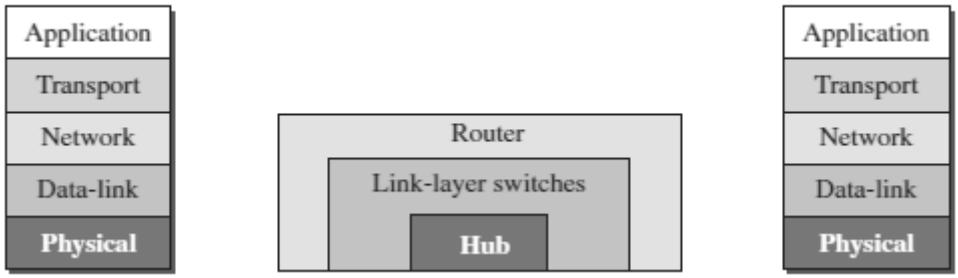


Figura 3.1. Capas en las que operan los concentradores, conmutadores, y rúters.

3.2. ADAPTADORES DE RED.

Antes de presentar los distintos dispositivos de conexión, vamos a discutir el papel que juegan los adaptadores de red. Un **adaptador de red** o **tarjeta de interfaz de red** (**NIC** = Network Interface Card) es un componente hardware que le permite a un host o a un rúter conectarse físicamente a una red a través de su medio de transmisión.

El adaptador de red es un dispositivo que opera tanto en la capa física como en la capa de enlace de datos. Como dispositivo de capa física, el adaptador de red de la estación emisora convierte los bits de datos en señales eléctricas u ópticas para su transmisión a través del medio físico (y recíprocamente, el adaptador de red de la estación receptora recibe las señales del medio y recupera los bits de datos originales). Como dispositivo de capa de enlace, el adaptador de red proporciona la dirección física (también llamada dirección de capa de enace o dirección MAC) a los equipos conectados a una red. Toda máquina conectada a una red debe tener al menos un adaptador de red con su propia dirección física. Esta dirección se usa para identificar a las distintas máquinas conectadas a una misma red.

Cuando los fabricantes de equipos producen un adaptador, graban la dirección física en su circuitería interna. El formato de la dirección física depende de la tecnología de la red empleada. Por ejemplo, las redes Ethernet utilizan direcciones físicas de 48 bits, representadas mediante 12 dígitos hexadecimales, como por ejemplo, A3:34:45:11:92:F1. Conjuntamente, los doce dígitos hexadecimales de la dirección física deben formar un identificador único para cada adaptador. Notar que, como la dirección física se graba en el adaptador durante el proceso de fabricación, se trata de un identificador permanente. Por consiguiente, la única forma de cambiar la dirección física de una estación es reemplazar su adaptador de red.

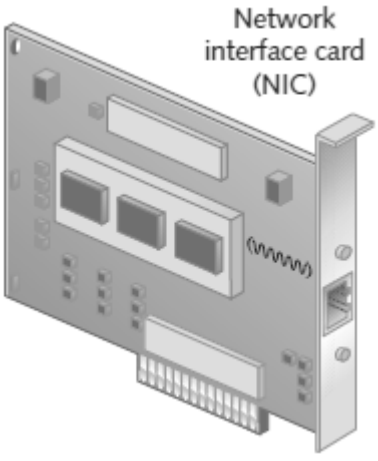


Figura 3.2. Adaptador de red Ethernet.

Como la dirección física sirve para identificar a un nodo (host o rúter) dentro de la red a la que pertenece, los protocolos de capa de enlace (por ejemplo, Ethernet) utilizan las direcciones físicas para mover las tramas de datos a través de esa red.

3.3. REPETIDORES Y CONCENTRADORES.

REPETIDORES.

Las señales que viajan a través de una red experimentan una pérdida de energía en forma de calor debida a la resistencia del medio de transmisión. A este proceso de degradación de la calidad de la señal se le denomina **atenuación**. La atenuación limita la distancia que pueden recorrer las señales antes de que la integridad de los datos se vea comprometida. Los repetidores nos permiten solventar los problemas derivados de la atenueción. Un **repetidor** es un dispositivo que solo opera en la capa física. El repetidor recibe una señal y regenera el patrón de bits original para evitar que la atenuación lo debilite o lo corrompa.

CONCENTRADORES (HUBS).

En el pasado, cuando las redes LAN Ethernet usaban una topología en bus, se usaban repetidores para conectar dos segmentos de red y superar las restricciones de longitud que imponía la atenuación del cable coaxial. Hoy en día, las redes LAN Ethernet usan una topología en estrella. En las topologías en estrella los repetidores son dispositivos multipuerto a los que se denomina **concentradores** (**hubs**), que pueden usarse como elementos de interconexión y como repetidores. La figura 3.3 muestra la situación en la que un ordenador A desea enviar un paquete a un ordenador B. El ordenador A transmite el paquete al concentrador, el cual regenera la señal digital que representa a dicha trama, y a continuación, la reenvía a todos sus puertos de salida excepto a aquel desde el que la recibió. En otras palabras, el concentrador difunde la trama a todos los ordenadores excepto al ordenador emisor. Todos los ordenadores de la red reciben la trama, pero solo el ordenador B la conserva; el resto de ordenadores la descartan.

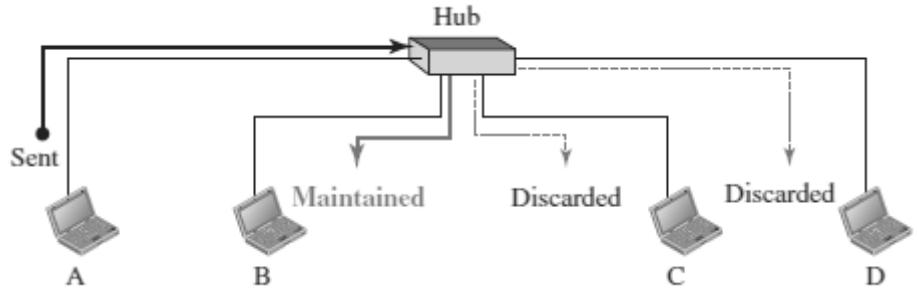


Figura 3.3. Concentrador (hub).

Como vemos, los concentradores no tienen ninguna capacidad de filtrado; no poseen la inteligencia necesaria como para determinar a cuál de sus puertos de salida deben reenviar la trama recibida.

Los concentradores y los repetidores son dispositivos de capa física. No tienen asignada una dirección de capa de enlace, y tampoco comprueban las direcciones de capa de enlace de las tramas recibidas; lo único que hacen es regenerar los bits recibidos y reenviarlos a todos sus puertos de salida.

3.4. SEGMENTACIÓN DE REDES.

La segmentación es un concepto fundamental en redes, y consiste en la división de una red muy poblada de equipos en varias porciones o segmentos poblados por menos equipos. Un **segmento** es una parte de una red que está separada lógica o físicamente del resto de la red. La técnica de segmentar una red permite mejorar su funcionamiento. En las redes no segmentadas (como por ejemplo, la red de la figura 3.3), toda

trama enviada por un nodo se difunde al resto de los nodos de la red, lo cual les obliga a procesarla, y en su caso, a admitirla o a descartarla. Esta forma de operar es muy poco eficiente. Sin embargo, al segmentar una red, las tramas enviadas a un cierto segmento no se reenvían al resto de segmentos. Como la segmentación reduce la cantidad de tramas que los nodos de la red deben comprobar y evaluar, esta técnica mejora el rendimiento de las redes en dos aspectos: En primer lugar, aumenta el ancho de banda disponible en una red, y en segundo lugar, permite separar dominios de colisión.

Aumento del ancho de banda. Consideremos la red LAN con doce ordenadores mostrada en la parte superior de la figura 3.4. Supongamos que la capacidad total de la red es de $10\,Mbps$. Si solo hay un ordenador que desee transmitir utilizará todo el ancho de banda disponible ($10\,Mbps$), pero si hay más estaciones que necesiten transmitir el ancho de banda total debe compartirse. Por consiguiente, en la red sin segmentar de la zona superior de la figura las doce estaciones se ven obligadas a compartir los $10\,Mbps$ de ancho de banda. Pero si usamos un *puente* (*bridge*) de dos puertos para dividir la red en dos segmentos con seis ordenadores, cada segmento es independiente del otro en términos de ancho de banda. En tal caso, los $10\,Mbps$ solo se comparten entre las seis estaciones de cada segmento (realmente siete, porque el puente actúa como una estación más). Obviamente, si dividimos la red en más segmentos podemos obtener más ancho de banda para cada segmento. Por ejemplo, si usáramos un puente de cuatro puertos en lugar de un puente de dos puertos, los cuatro segmentos resultantes solo compartirían los $10\,Mbps$ de ancho de banda entre cuatro estaciones.

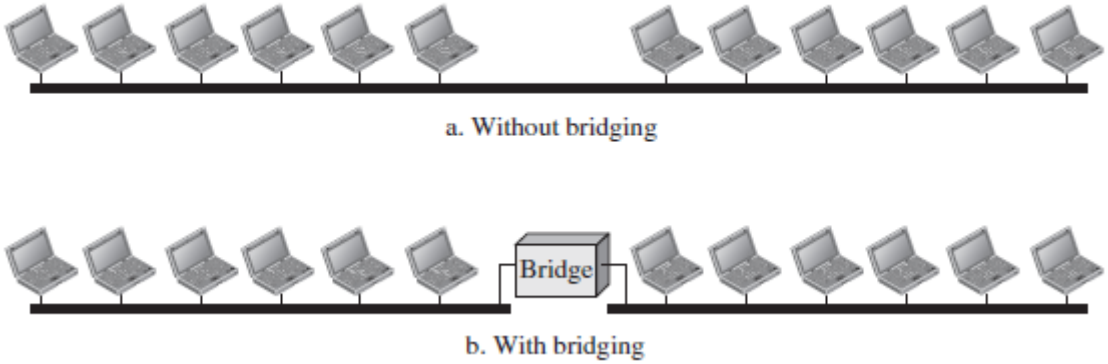


Figura 3.4. Aumento del ancho de banda con un bridge de dos puertos.

Separación de dominios de colisión. Otra de las ventajas de la segmentación es la separación de los distintos *dominios de colisión*. Una **colisión** es la situación en la que dos ordenadores en una red de difusión intentan enviar información simultáneamente a través del medio compartido. El resultado es una superposición de señales en el canal que hace que los datos sean irrecuperables. En las redes Ethernet no conmutadas las colisiones son frecuentes, debido a la difusión de los datos. Un **dominio de colisión** es el área de una red en la que puede producirse una colisión.

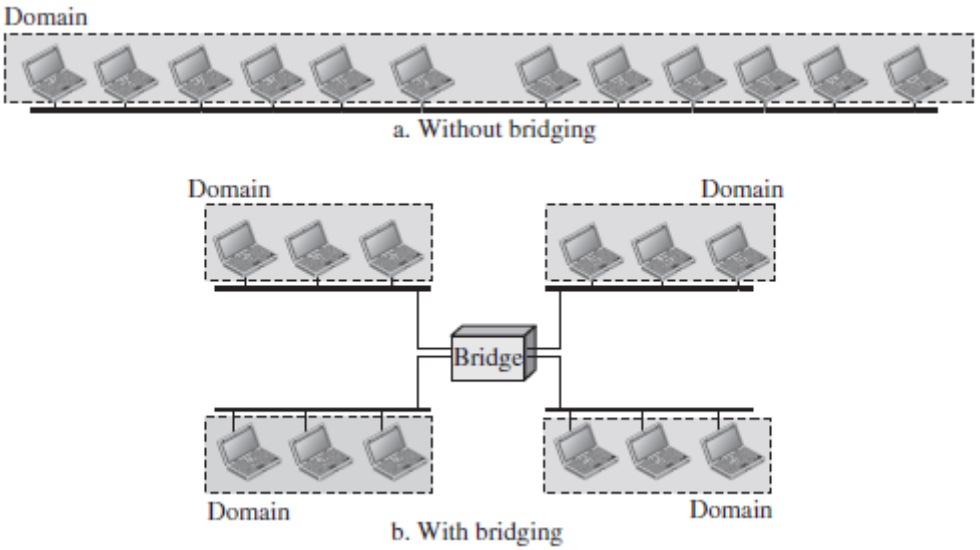


Figura 3.5. Dominios de colisión sin bridge y con bridge.

La figura 3.5 muestra los dominios de colisión en una red sin segmentar y en una red dividida en cuatro segmentos. Como vemos, los dominios de colisión se hacen mucho más pequeños en la red segmentada, y la probabilidad de que se produzca una colisión disminuye drásticamente. Sin segmentación, las doce estaciones de la red competirían por el acceso al medio; con segmentación solo hay tres estaciones por dominio de colisión contendiendo por acceder al medio.

3.5. PUENTES Y CONMUTADORES.

PUENTES (BRIDGES).

Un **puente** (**bridge**) es un dispositivo que opera tanto en la capa física como en la capa de enlace. Como dispositivo de capa física, regenera las señales que recibe. Y como dispositivo de capa de enlace, el puente es capaz de comprobar las direcciones físicas (origen y destino) contenidas en las tramas que recibe.

La principal diferencia entre un concentrador y un puente es su capacidad de **filtrado**. Un puente puede comprobar la dirección física de destino de una trama y decidir a qué puerto de salida debe reenviarla. Para ello, el puente dispone de una **tabla de reenvío** que le permite tomar sus decisiones de filtrado. Veamos un ejemplo: La figura 3.6 muestra una LAN conmutada en la que tenemos cuatro estaciones conectadas a los cuatro puertos de un puente. Si al puerto 1 llega una trama destinada a la estación cuya dirección física es 71:2B:13:45:61:42, el puente consulta su tabla para determinar el puerto de salida al que debe reenviar esa trama. Según su tabla, las tramas dirigidas a la estación 71:2B:13:45:61:42 solo deberían reenviarse a través del puerto 2; por consiguiente, no hay necesidad de reenviar la trama al resto de puertos. Sin embargo, si uno de los ordenadores envía una trama de broadcast, el puente la retransmitirá a todos los puertos de salida, excepto al puerto desde el que llegó la trama. (Una trama de difusión es una trama cuyos destinatarios son todos los nodos de una red, y que en la tecnología Ethernet viene identificada por la dirección física especial FF:FF:FF:FF:FF:FF). Además, si un puente no encuentra la dirección física de destino de una trama en su tabla, reenvía la trama a todos sus puertos de salida excepto a aquel desde el que la recibió.

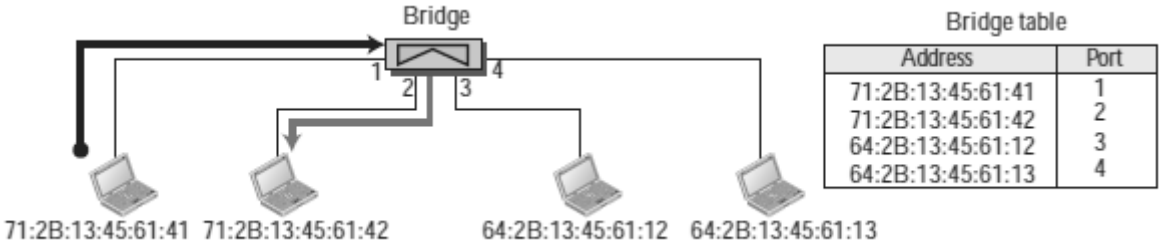


Figura 3.6. Puente (bridge).

Un hecho importante a destacar es que los puentes no cambian las direcciones físicas de las tramas; únicamente las analizan para determinar a cuál de sus puertos de salida deben reenviarlas.

PUENTES TRANSPARENTES. (*)

Un **puente transparente** es un puente que opera de forma que los ordenadores que conecta son completamente ajenos a su existencia. Los puentes transparentes también suelen denominarse **puentes de aprendizaje**, por la forma en la que construyen sus tablas de reenvío analizando las direcciones físicas de las tramas que reciben.

Los primeros puentes disponían de tablas de reenvío que eran estáticas. El administrador del sistema introducía manualmente cada entrada de la tabla durante la configuración del puente. Aunque el proceso de configuración era sencillo, no era nada práctico: Si se añadía o se quitaba una estación, o si la dirección física de una estación cambiaba, el administrador debía reconfigurar la tabla manualmente.

Una alternativa preferible a las tablas estáticas son las tablas dinámicas que asocian direcciones físicas a puertos de forma automática. Para poder tener una tabla dinámica, necesitamos un puente que aprenda gradualmente a partir de los movimientos de las tramas en la red. Para ello, el puente debe inspeccionar tanto la dirección origen como la dirección destino de cada trama que pase a través de él. El puente usa la dirección de destino para realizar las búsquedas en la tabla que le permiten tomar la decisión de reenvío, y usa la dirección de origen para añadir y para actualizar las entradas de su tabla dinámica. A modo de ejemplo, consideremos la situación ilustrada en la figura 3.7:

a) Cuando la estación A envía una trama a la estación D, el puente todavía no tiene una entrada en su tabla ni para A ni para D. La trama sale a través de los tres puertos y se difunde a *todas* las estaciones (la trama *inunda* la red). Sin embargo, analizando la dirección física de origen de la trama, el puente aprende que la estación A está conectada al puerto 1. Esto significa que, en el futuro, las tramas destinadas a la estación A deben reenviarse a través del puerto 1. El puente añade esta entrada a su tabla. La tabla tiene ahora su primera entrada.
b) Cuando la estación D envía una trama a la estación B, el puente no tiene una entrada para la estación B, así que vuelve a inundar la red. Sin embargo, el puente añade una entrada más a la tabla asociada a la estación D.
c) El proceso de aprendizaje continúa hasta que el puente añade en su tabla el puerto de salida que le permite llegar a cada estación de la red. Sin embargo, notar que el proceso de aprendizaje puede llevar bastante tiempo. Por ejemplo, si una estación no envía una trama (una situación bastante rara), la estación nunca tendrá una entrada en la tabla.

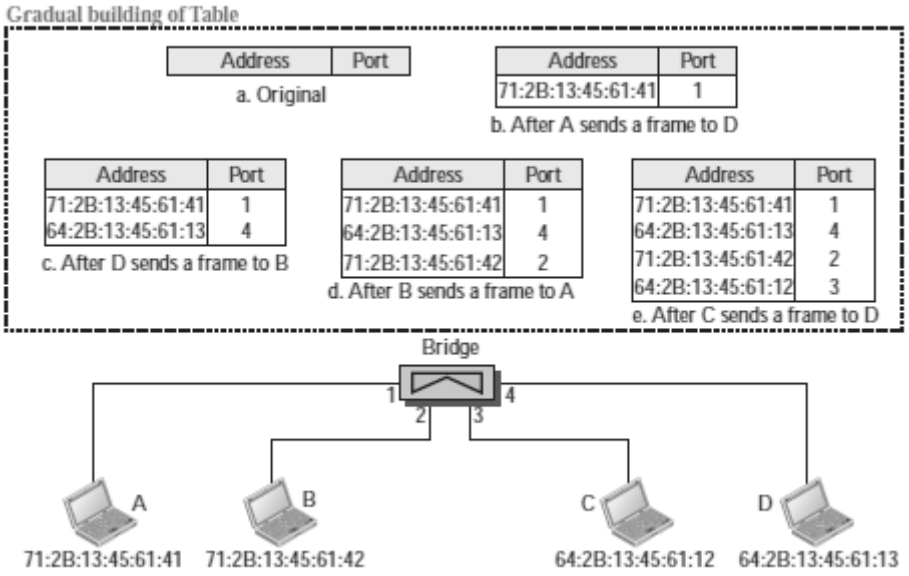


Figura 3.7. Proceso de aprendizaje de un puente transparente.

CONMUTADORES (SWITCHES).

Un **conmutador** (switch) es un dispositivo que opera tanto en la capa física como en la capa de enlace. Un conmutador es básicamente un puente más sofisticado que dispone de más puertos de salida (los puentes pueden tener dos o cuatro puertos), y que es capaz de procesar más rápidamente las tramas que recibe.

La figura 3.8 muestra una red LAN no conmutada sin segmentar con seis ordenadores, la misma red dividida en dos segmentos mediante un puente, y una red LAN conmutada con seis ordenadores conectados a un conmutador. A la vista de la figura, es evidente que los conmutadores son dispositivos más ventajosos que los puentes. Como el conmutador ofrece un mayor número de puertos, los ordenadores no tienen que agruparse en segmentos, sino que pueden conectarse directamente al conmutador. Ello implica que cada ordenador solo comparte el ancho de banda con el conmutador, y que los dominios de colisión quedan

reducidos a dos equipos (la estación y el conmutador). De hecho, si el enlace entre la estación y el conmutador es full – dúplex, el problema de las colisiones desaparece por completo.

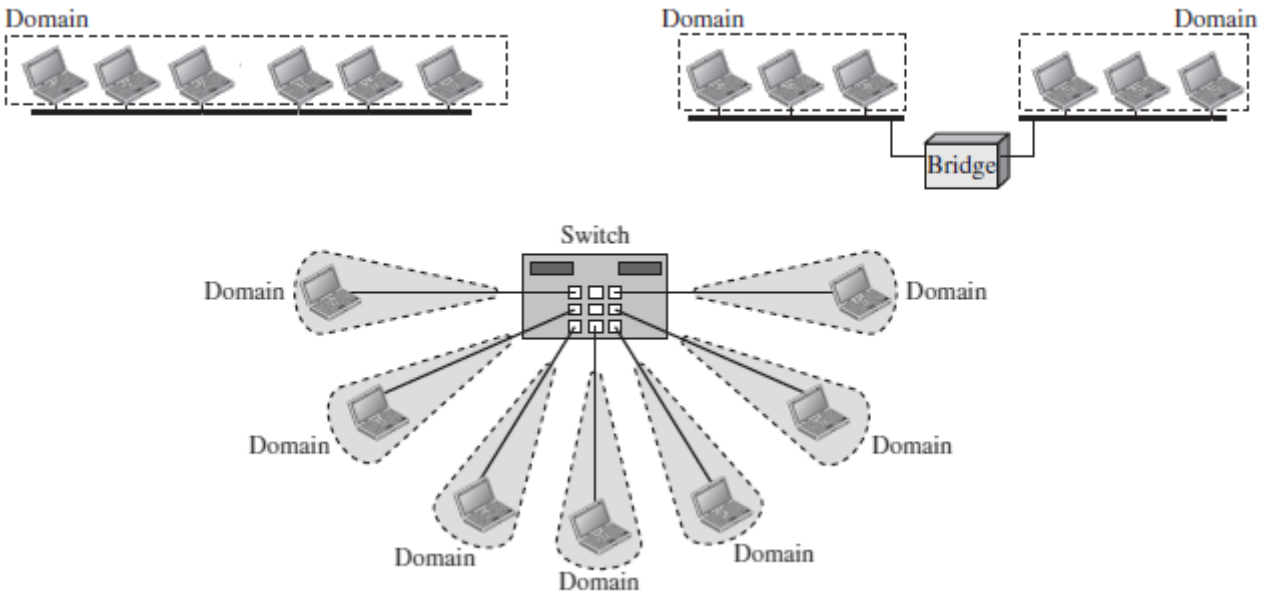


Figura 3.8. La misma red LAN en bus, puenteada, y conmutada.

La evolución de las antiguas redes LAN no segmentadas a las redes LAN segmentadas con puentes, y de ahí a las redes conmutadas permitió abrir el camino a nuevos estándares Ethernet mucho más rápidos y sin problemas de colisiones.

3.6. RÚTERS.

Un **rúter** es un dispositivo de tres capas, ya que opera en la capa física, en la capa de enlace de datos, y en la capa de red. Como dispositivo de capa física, el rúter regenera la señal que recibe. Como dispositivo de capa de enlace, el rúter comprueba las direcciones físicas (origen y destino) contenidas en el paquete. Y como dispositivo de capa de red, el rúter examina las direcciones lógicas de capa de red (direcciones IP).

Aunque un rúter puede usarse como dispositivo de capa de enlace para interconectar los distintos dispositivos de una red, ésta no es su aplicación natural (para eso ya están los puentes y los conmutadores). Los rúters sirven para interconectar redes. En otras palabras, un rúter es un dispositivo que permite conectar varias redes independientes para formar una interred. Según esta definición, las redes interconectadas mediante un rúter se convierten en una interred.

Los rúters se parecen a los puentes y conmutadores en que son capaces de filtrar el tráfico. Pero en vez de filtrar el tráfico analizando las direcciones físicas de las tramas que reciben (tal y como hacen los puentes y los conmutadores), los rúters filtran el tráfico comprobando las direcciones IP de los datagramas que reciben. Análogamente a los puentes y conmutadores, los rúters usan una **tabla de encaminamiento** para determinar a cuál de sus puertos (o interfaces) deben reenviar los paquetes recibidos. Además, cuando añadimos un rúter a una red, el rúter genera más redes, porque cada interfaz del rúter representa una red distinta. Otro aspecto diferenciador es que, mientras que un conmutador divide una red para generar varios segmentos, un rúter divide una red para generar varias redes. Los rúters crean dominios de colisión, pero también crean dominios de difusión, porque no dejan pasar los mensajes de broadcast de una red a otra. Un **dominio de difusión** es el conjunto de dispositivos de una red que reciben un mensaje de broadcast enviado por uno de los ordenadores de la red.

Otras características relevantes de los rúters son las siguientes:

1) Un rúter tiene una dirección física y una dirección IP por cada uno de sus interfaces. (Por ejemplo, si un rúter interconecta tres redes, dispondrá de tres parejas de direcciones físicas e IP).
2) Un rúter solo actúa sobre los paquetes cuya dirección física de destino coincide con la dirección física del interfaz al que llegan esos paquetes.
3) Un rúter cambia la dirección física del paquete (tanto la dirección de origen como la de destino) cuando reenvía el paquete a otra red.

Veamos un ejemplo: La figura 3.9 ilustra una organización con dos edificios independientes, cada uno de los cuales dispone de una red LAN conmutada de tipo Gigabit Ethernet ($1\ Gbps$). Las dos LANs conmutadas están interconectadas mediante un tercer conmutador para formar una LAN de tipo 10 - Gigabit Ethernet ($10\ Gbps$), que permite acelerar la conexión entre las dos redes LAN Gigabit Ethernet y la conexión al servidor de la organización. Además, el sistema dispone de un rúter para conectar toda la red a Internet. En este ejemplo, el rúter opera conectando dos redes: La red LAN 10 - Gigabit Ethernet de la organización, e Internet.

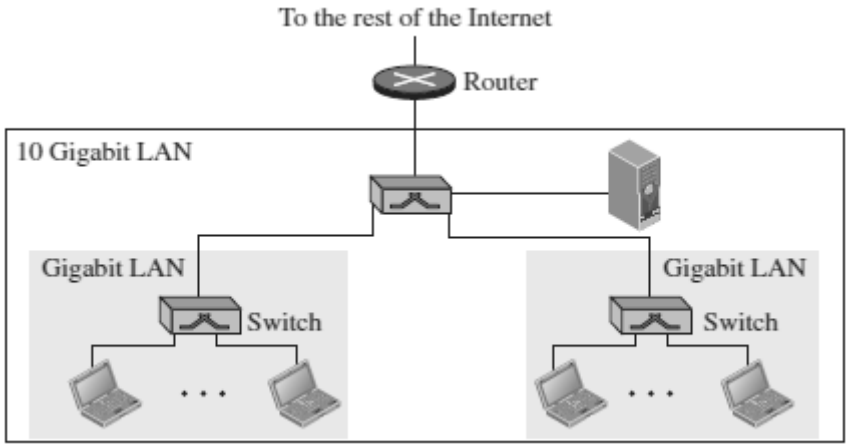


Figura 3.9. Ejemplo de utilización de un rúter.

Como vimos en el capítulo 2, un rúter opera cambiando las direcciones físicas de los paquetes que recibe, porque las direcciones físicas solo tienen jurisdicción local (esto es, solo son válidas dentro de la red a la que pertenecen).

PRÁCTICA G. PACKET TRACER (1): INTRODUCCIÓN.

G.1. ¿QUÉ ES PACKET TRACER?

Packet Tracer es un programa de *simulación* de redes de ordenadores desarrollado por **Cisco Systems**, uno de los principales fabricantes de dispositivos de comunicaciones del mundo (https://www.cisco.com/). Al contrario que las herramientas que hemos usado hasta ahora, que trabajan sobre redes reales, Packet Tracer nos permite simular redes virtuales sin las limitaciones de las redes reales (disponibilidad de equipos, conexionado físico de equipos, coste económico, tiempo de despliegue, etc.). Este software proporciona una gran variedad de dispositivos Cisco, como conmutadores, rúters, dispositivos inalámbricos, dispositivos terminales (como PCs o portátiles), y servidores, que podemos interconectar para construir y simular redes en nuestro ordenador

G.2. ARRANCAR PACKET TRACER.

Para arrancar Packet Tracer pinchamos dos veces en el icono del programa, después de lo cual se abre una ventana de bienvenida. En la parte inferior de esta ventana se nos pide que accedamos con una cuenta de usuario, o que ingresemos como invitados. Pulsamos el botón de acceso como invitados ("Guest Login"). Tras pulsar, en el mismo botón aparecerá una cuenta atrás de 15 segundos, tras la cual el texto del botón cambia a "Confirm Guest". Volvemos a pulsar en ese botón para entrar en el programa propiamente dicho.

G.3. LA VENTANA PRINCIPAL DE PACKET TRACER.

La figura G.1 muestra la ventana principal de Packet Tracer, donde vemos que está dividida en varias zonas.

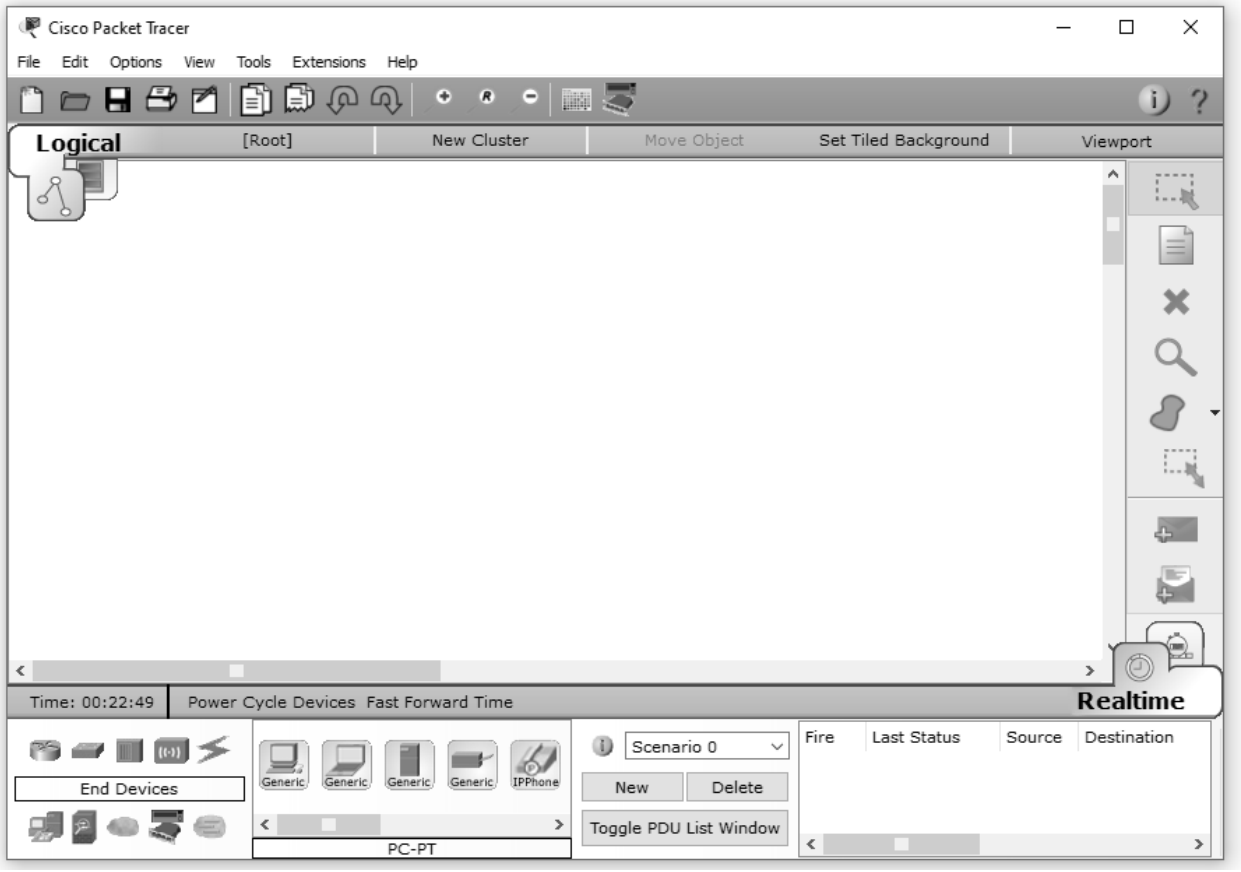


Figura G.1. La ventana principal de Packet Tracer.

La **barra de menús** en la parte superior de la pantalla incluye los menús típicos de cualquier aplicación software, y se usa para abrir, guardar, imprimir, cambiar preferencias, etc.

La **barra principal de herramientas** se ubica bajo la barra de menús, y proporciona algunos botones de acceso directo a las opciones más usadas de los menús superiores, como abrir, guardar, ampliar y reducir, deshacer, rehacer, etc.

Las **pestañas lógica y física** en la esquina superior izquierda del área de trabajo nos permiten pasar de la zona de trabajo lógica a la zona de trabajo física.

La **zona de trabajo** es el área en blanco donde se crean las topologías de red a simular, y donde se muestran las simulaciones.

La **barra de herramientas de uso común** es la columna de botones a la derecha del área de trabajo que contiene los controles para manipular la topología de la red creada, como por ejemplo, hacer selecciones, mover la topología, añadir una nota, borrar, redimensionar, inspeccionar, y añadir PDUs (paquetes) simples y complejas para lanzar simulaciones.

Las **pestañas tiempo real y simulación** ubicadas en la esquina inferior derecha del área de trabajo se usan para cambiar entre los modos de tiempo real y de simulación. También incorporan botones para controlar el tiempo y para capturar paquetes.

La **caja de componentes de red** bajo el área de trabajo y a la izquierda contiene todos los dispositivos de red disponibles en Packet Tracer, y se divide en dos partes: (1) La **caja de selección del tipo de dispositivo** contiene las distintas categorías en las que se agrupan los dispositivos (dispositivos finales, rúters, conmutadores, concentradores, dispositivos inalámbricos, conexiones, etc.). (2) La **caja de selección del dispositivo específico** incluye todos los dispositivos disponibles en la categoría elegida.

Por último, la **caja de paquetes creados por el usuario** bajo el área de trabajo y a la derecha es donde los usuarios pueden crear paquetes altamente personalizados para probar la topología diseñada.

G.4. CONEXIÓN DIRECTA ENTRE DOS ORDENADORES.

Una vez familiarizados con la interfaz de usuario de Packet Tracer, vamos a crear nuestra primera topología de red. Para ello, seguimos estos pasos:

1) En la caja de componentes de red pinchamos en la categoría "End devices", y arrastramos al área de trabajo los dispositivos genéricos PC y Laptop (portátil).

2) Ahora pinchamos en la categoría "Connections", seleccionamos un **cable cruzado** ("Copper Cross-Over"), y pinchamos en el componente PC0 para seleccionar su interfaz "FastEthernet0". A continuación, arrastramos el cable y pinchamos sobre el componente Laptop0 para conectar el PC al portátil a través de su interfaz FastEthernet0 (ver figura G.2).

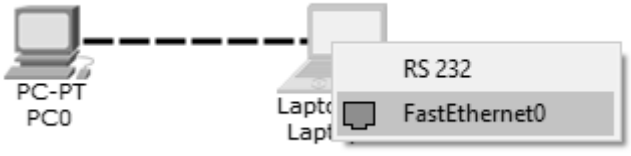


Figura G.2. Conexión a los interfaces Ethernet de los ordenadores.

El LED de estado del enlace debería aparecer en verde, indicando que el enlace está disponible y operativo. La topología final debería quedar como muestra la figura G.3:

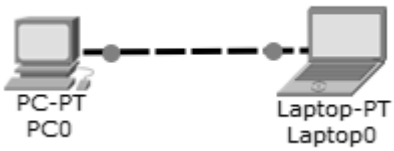


Figura G.3. Topología final de la conexión directa entre dos ordenadores.

3) Vamos a asignar direcciones IP estáticas a los ordenadores, comenzando con el PC. Para ello, pulsamos sobre el PC, acudimos a la pestaña "Config", y seleccionamos el interfaz FastEthernet0. Se abrirá la ventana de configuración IP, donde escribiremos la dirección IP del PC (ver figura G.4). Aquí usaremos la dirección IP privada 192.168.1.101. A continuación, proporcionamos la máscara de subred correspondiente a esa IP privada, que es 255.255.255.0. No vamos a establecer la configuración IPv6, porque solo trabajaremos con el protocolo IPv4. Notar que en esta ventana también podemos consultar la dirección física del PC, que en este caso resulta ser 0000.0C1A.4735.

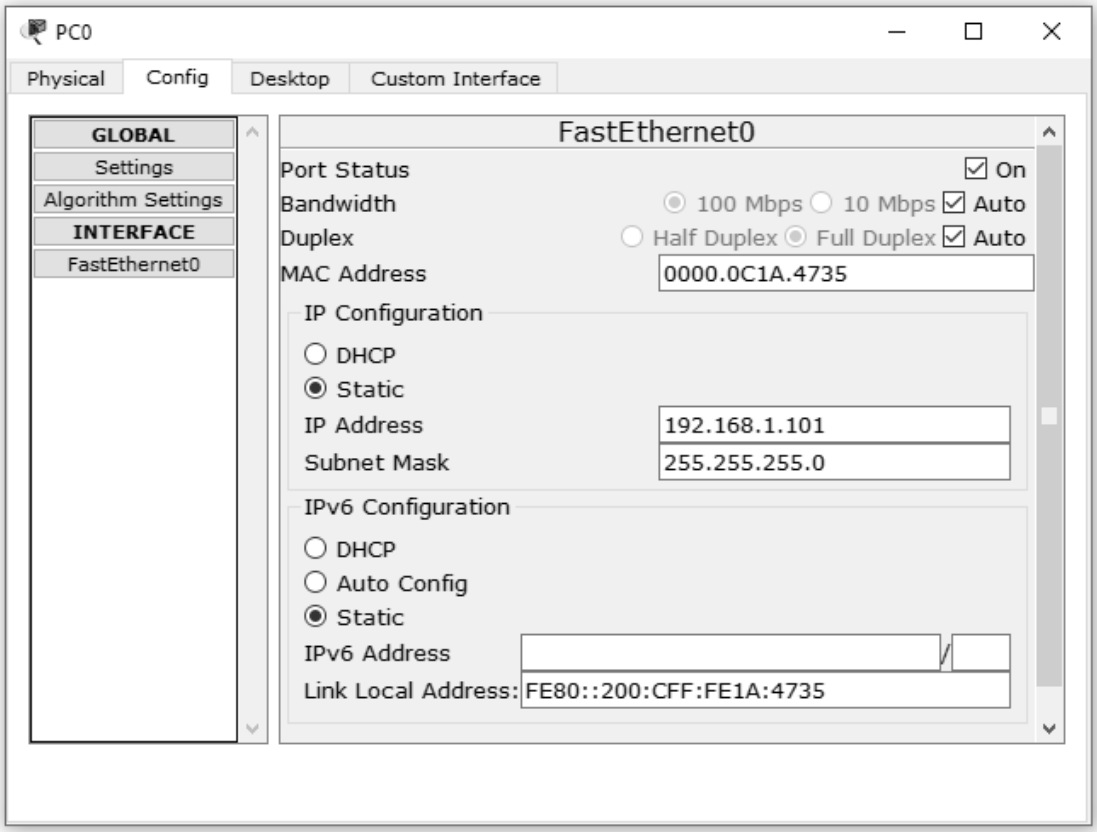


Figura G.4. Configuración IP del PC.

4) Cerramos la ventana, pulsamos sobre el portátil, y procedemos con su configuración IP de la misma forma que lo hicimos con el PC. En este caso, vamos a asignar al portátil la IP 192.168.1.102. Notar que ambas direcciones IP pertenecen a la misma red (192.168.1.0).

5) Cerramos la ventana de configuración IP del portátil. Vamos a probar la conectividad entre los dos equipos. Para ello, pulsamos en el PC, seleccionamos la etiqueta "Desktop" (escritorio), y de entre todas las opciones posibles, entramos en la ventana de comandos del PC ("Command prompt"). Una vez dentro de la ventana de comandos, hacemos un `ping` a la dirección IP del portátil (ver figura G.5).

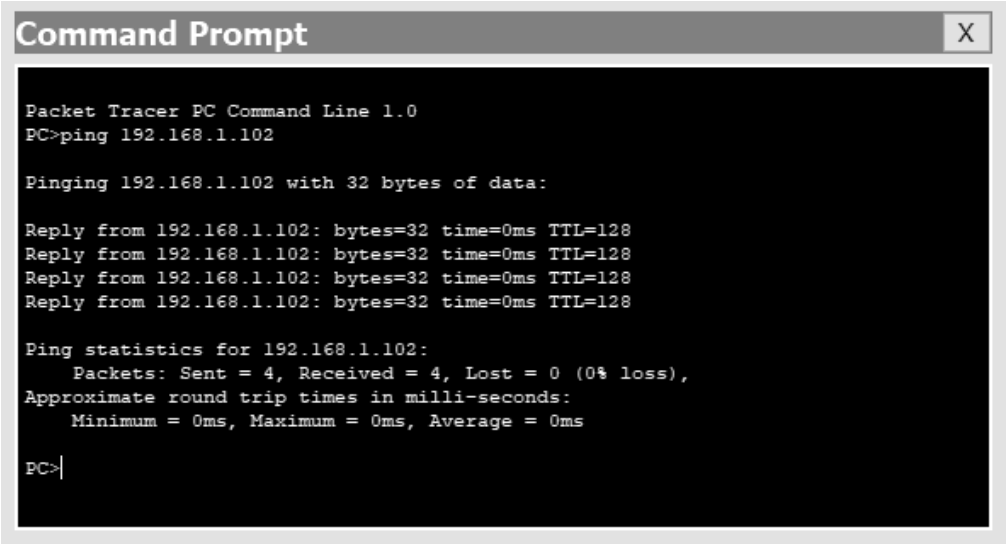


Figura G.5. Resultado de un `ping` desde el PC al portátil.

G.5. CONEXIÓN DE DOS ORDENADORES MEDIANTE UN CONMUTADOR.

A continuación vamos a construir una nueva topología en la que los dos ordenadores previos están interconectados mediante un conmutador Ethernet. Esta configuración nos permitirá conectar más de dos ordenadores a la red. Para construir esta topología, realizamos los siguientes pasos:

1) En la caja de selección del tipo de dispositivo, pinchamos en la categoría "Switches", y arrastramos al área de trabajo un conmutador cualquiera (excepto el conmutador llamado Switch-PT-Empty). Aquí hemos usado el conmutador genérico (switch-PT).
2) Eliminamos el enlace directo entre el PC y el portátil, usando la herramienta de borrado disponible en la barra de herramientas de uso común.
3) En la categoría de conexiones, seleccionamos un **cable directo** ("Copper Straight-Through") y conectamos el interfaz FastEthernet0 del PC al primer interfaz FastEthernet del conmutador (el interfaz FastEthernet0/1, ver figura G.6).[16]

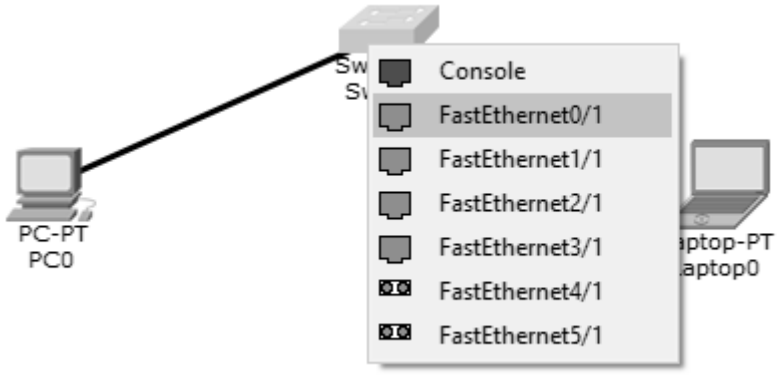


Figura G.6. Conexión de un PC a un conmutador.

A continuación, y mediante un cable directo, conectamos el interfaz FastEthernet0 del portátil con el interfaz FastEthetnet1/1 del conmutador. Acabamos de montar una red LAN conmutada de dos ordenadores. Llegados a este punto, los LEDs indicadores del conmutador deben brillar en naranja,

[16] Podemos distinguir dos tipos de cables de red: Los cables directos y los cables cruzados. Un **cable directo** es el cable que se usa para conectar dispositivos que *no* son iguales, como un ordenador a un concentrador (hub), un ordenador a un conmutador (switch), etc. Por el contrario, un **cable cruzado** es aquel que se utiliza para conectar dispositivos similares, como dos ordenadores entre sí, o dos conmutadores entre sí.

porque el conmutador está pasando por el proceso de aprendizaje que le permitirá rellenar su tabla de reenvío.

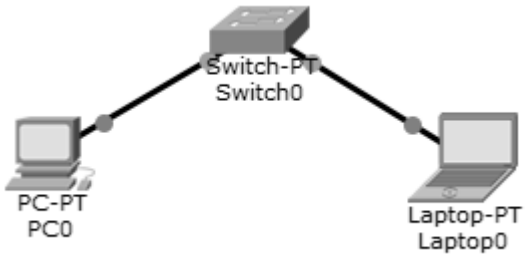


Figura G.7. Conexión de dos ordenadores a través de un conmutador.

4) Cuando los indicadores pasen a verde, hacemos un `ping` para comprobar la conectividad entre los dos ordenadores.
5) Para guardar esta topología, acudimos al menú "File" → "Save", y elegimos la localización donde queremos guardar el archivo, y el nombre del archivo. El archivo se guardará con una extensión .pkt.

PRÁCTICA H. PACKET TRACER (2): LAN ESTÁNDAR.

H.1. MONTAR Y CONFIGURAR LA TOPOLOGÍA DE LA RED.

En las próximas prácticas vamos a usar el simulador Packet Tracer para ilustrar cómo funcionan los dispositivos de conexión presentados en el capítulo 3. Aquí comenzaremos montando la topología de una red LAN estándar (no conmutada) de 4 PCs conectados a un concentrador (hub). Para ello arrastramos cuatro PCs al área de trabajo, y a continuación, añadimos un hub genérico. Seguidamente, conectamos mediante cables directos los puertos FastEthernet0 de los PCs a cuatro puertos libres del hub. La topología final debería quedar como en la figura H.1.

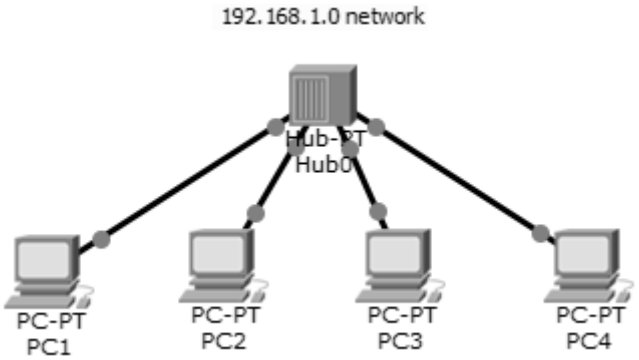


Figura H.1. Topología física de la red LAN estándar.

Ahora vamos a asignar una dirección IP estática a cada PC de la red: Al PC1 le asignamos la dirección IP privada 192.168.1.101, al PC2 le asignamos la dirección IP 192.168.1.102, y así sucesivamente. En cada PC, añadimos también la máscara de subred correspondiente, que para este tipo de direcciones IP es 255.255.255.0. Si así lo deseamos, podemos añadir una nota en el área de trabajo que indique que estamos trabajando con la red 192.168.1.0. Para ello, usamos la herramienta "Place Note (N)" disponible en la barra de herramientas de uso común.

H.2. SIMULAR LA RED.

Para lanzar simulaciones en Packet Tracer, primero debemos seleccionar la **pestaña de simulación** en la parte inferior derecha del área de trabajo. (Es muy importante no olvidarnos de esto; en caso contrario, los envíos de paquetes se harán en tiempo real, y no veremos nada). Al pulsar en esta pestaña se abrirá el **panel de simulación**, que incluye la lista de los eventos ocurridos en la simulación ("Event list"), los controles para reproducir la simulación ("Play controls"), y las opciones de filtrado de los paquetes que mostrará la simulación, por tipo de protocolo. Todo ello podemos verlo en la figura H.2.

Para lanzar nuestra primera simulación, vamos a comprobar si los paquetes de información llegan de un ordenador a otro. Para ello acudimos a la barra de herramientas de uso común, y pulsamos en el botón "Add simple PDU (P)". Seguidamente, pinchamos con el sobre en el ordenador origen (por ejemplo, PC1) y a continuación, en el ordenador destino (por ejemplo, PC4). (Esto es otra manera alternativa de lanzar un `ping` del PC1 al PC4 sin acudir a la ventana de comandos). Al hacerlo, arranca la simulación. Los paquetes intercambiados se muestran como sobres que viajan a lo largo de la topología de la red. Los distintos eventos que ocurren en la simulación aparecen como líneas independientes en la lista de eventos, y entre otras cosas, indican el instante en el que ocurrió el evento (Time), la localización previa del paquete (Last device), la localización actual del paquete (At device), y el tipo (protocolo) del paquete intercambiado. Con los controles de reproducción podemos elegir si reproducir la simulación de manera continua (botón "Autocapture / Play"), o si reproducir la simulación evento a evento (Botones "Capture / Forward" y "Back").

Vamos a visualizar la simulación evento a evento: Para ello vamos pulsando en el botón "Capture / Forward" cada vez que queramos que la simulación avance al siguiente evento. Además, y para simplificar las cosas, vamos a aplicar un filtro para mostrar únicamente los paquetes ICMP. Al empezar la simulación, vemos que el PC1 crea un paquete ICMP para hacer el `ping` al PC4. El PC1 envía el paquete ICMP al concentrador, el cual lo reenvía a todos sus puertos de salida, excepto al puerto desde el que llegó el paquete. El paquete ICMP llega a los PCs 2, 3, y 4, pero solo lo conserva el PC4. A continuación, El PC4 responde al `ping` enviando un paquete ICMP de vuelta al concentrador, que lo difunde a los PCs 1, 2, y 3. Como vemos, solo el PC1 procesa el mensaje de respuesta ICMP; los PCs 2 y 3 lo descartan.

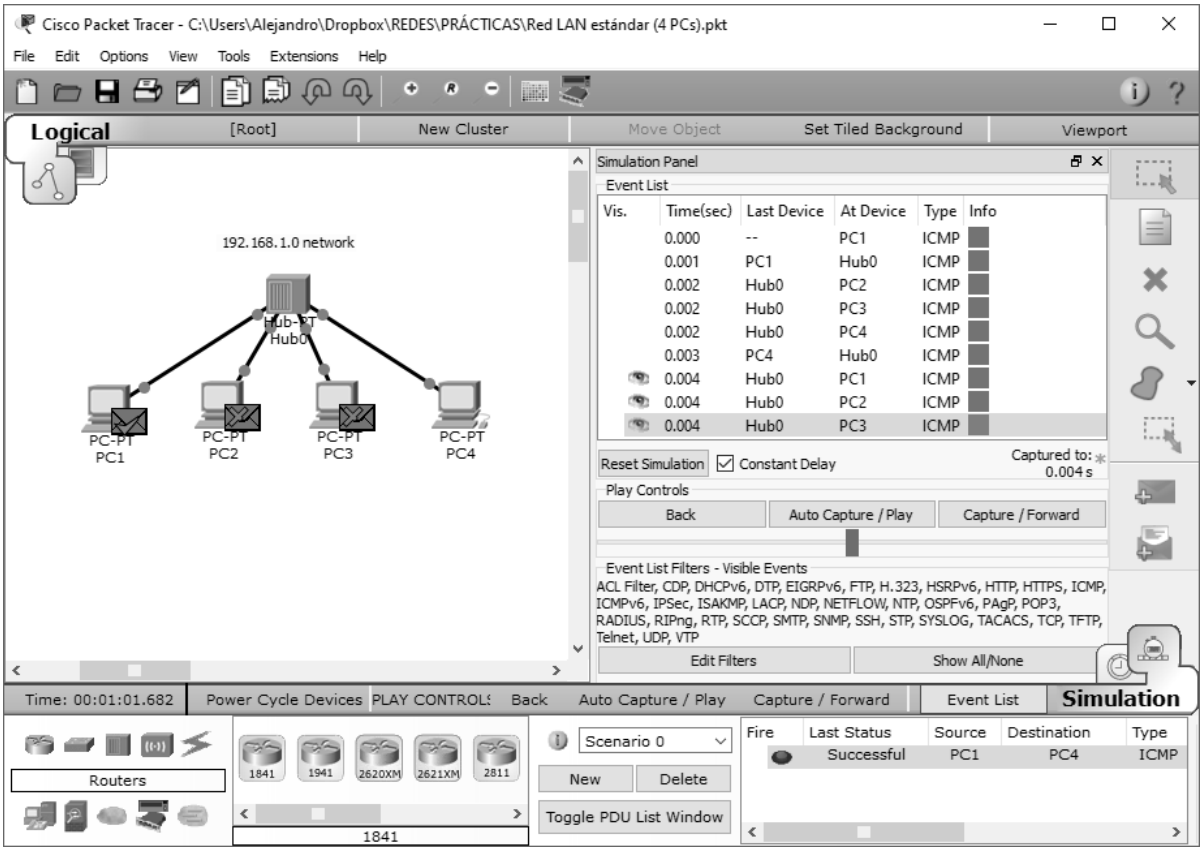


Figura H.2. Simulación de un ping en una red LAN con concentrador.

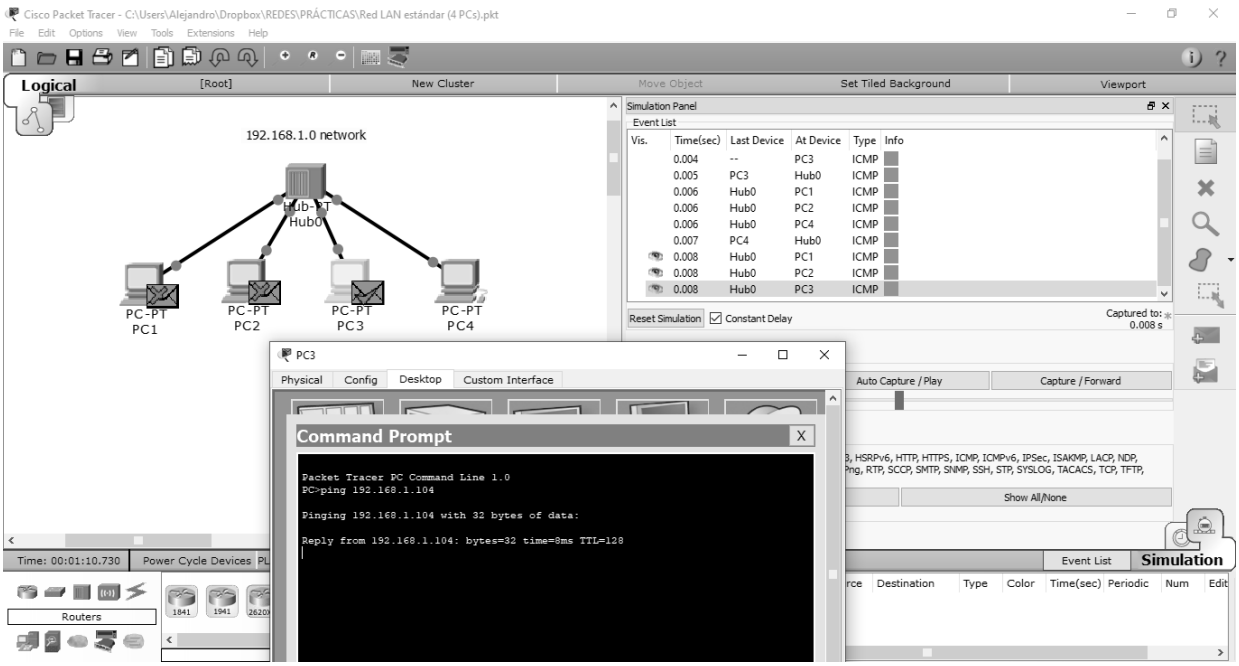


Figura H.3. Simulación de un `ping` del PC3 al PC4 por ventana de comandos.

Si queremos volver a iniciar la misma simulación, pulsamos en el botón "Reset simulation" bajo la lista de eventos. Si queremos borrar la simulación para lanzar una nueva, pulsamos en el botón "Delete" en la zona inferior de la pantalla. Notar que podemos lanzar varias simulaciones distintas en la misma topología sin borrar la simulación previa, simplemente pulsando en el botón "New". Las distintas simulaciones quedan registradas con nombres diferentes (Scenario 0, Scenario 1, etc.).

Otra forma de lanzar la misma simulación es ejecutar el `ping` a través de la ventana de comandos del ordenador emisor. A modo de ejemplo, vamos a hacer un `ping` desde el PC3 al PC4. Para ello, acudimos a la ventana de comandos del PC3, y ejecutamos el comando `ping 192.168.1.104`. Al igual que antes, veremos los paquetes ICMP circular por la red, y la lista de eventos rellenándose con los distintos eventos ocurridos durante la simulación (Ver figura H.3).

PRÁCTICA I. PACKET TRACER (3): RED LAN CONMUTADA.

I.1. MONTAR Y CONFIGURAR LA TOPOLOGÍA DE LA RED.

En esta práctica comenzamos montando la topología de una red LAN con 6 PCs conectados a un conmutador Cisco modelo 2950-24. (Este conmutador ofrece más puertos de conexión que el conmutador genérico que usamos en la práctica de introducción). Mediante cables directos conectamos los interfaces FastEthernet0 de los seis PCs a seis puertos FastEthernet libres del conmutador. Para seguir un orden lógico, el PC1 lo conectaremos al interfaz FastEthernet0/1 del conmutador, el PC2 al interfaz FastEthernet0/2, y así sucesivamente. La topología final debería quedar como muestra la figura I.1:

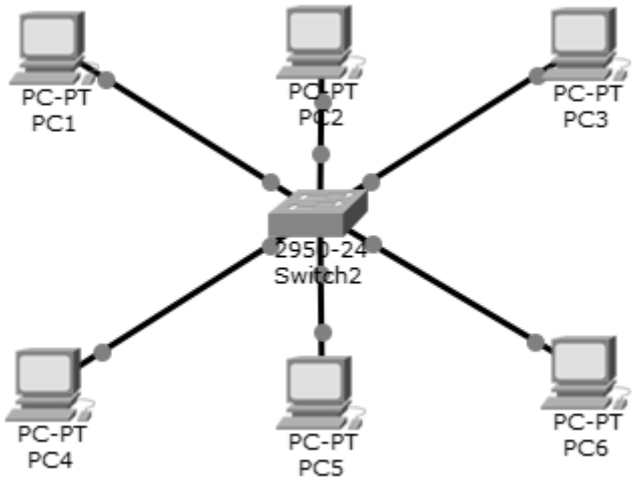


Figura I.1. Topología de la red LAN conmutada.

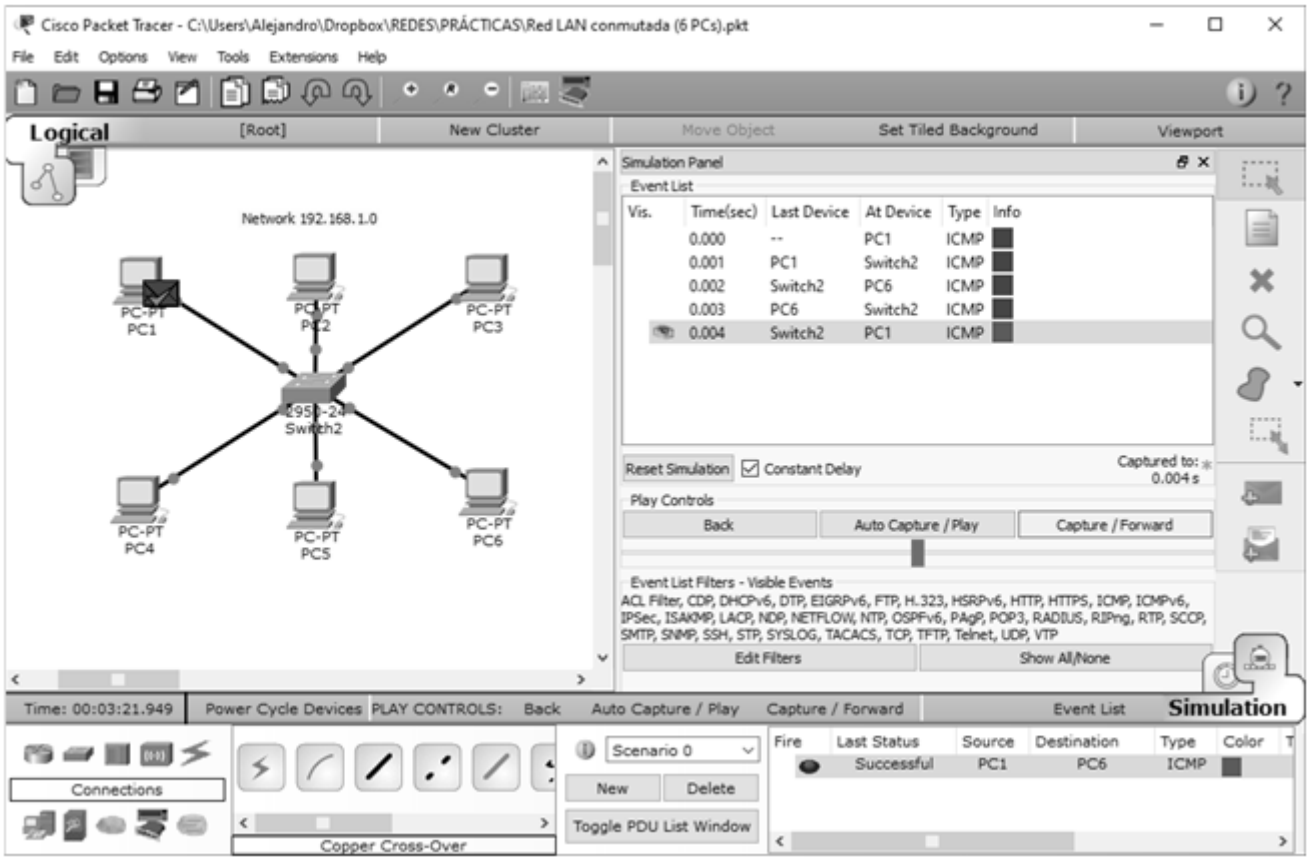


Figura I.2. Simulación de un `ping` desde el PC1 al PC6 con una PDU simple.

Como siempre, asignamos una dirección IP estática a cada PC de la red. Al PC1 le asignamos la dirección IP privada 192.168.1.101, al PC2 le asignamos la dirección IP 192.168.1.102, etc. No debemos olvidar añadir la máscara de subred 255.255.255.0.

I.2. SIMULAR LA RED.

Como en la práctica previa, vamos a comprobar la conectividad entre los distintos ordenadores de la red. Para ello, enviamos una PDU simple del PC1 al PC6. Como vemos, el PC1 crea un paquete ICMP para hacer un `ping` al PC6. El PC1 manda el paquete ICMP al conmutador, el cual lo reenvía al PC6. El PC6 responde a `ping` enviando un paquete ICMP de vuelta al conmutador, que lo redirige al PC1. Notar que, en este caso, el conmutador no difunde los paquetes ICMP, sino que los reenvía únicamente al ordenador de destino deseado.

PRÁCTICA J. PACKET TRACER (4): DOS REDES INTERCONECTADAS MEDIANTE UN RÚTER (ENCAMINAMIENTO ESTÁTICO).

J.1. MONTAR Y CONFIGURAR LA TOPOLOGÍA DE LA RED.

A continuación vamos a montar una interred formada por dos redes LAN conmutadas. Para empezar, construimos una LAN A (con dirección IP 192.168.1.0) formada por tres PCs interconectados mediante un conmutador (switch0). Después levantamos una LAN B (con dirección IP 192.168.2.0) con cuatro PCs y un conmutador (switch1). La figura J.1 muestra cómo debería quedar la topología.

Como siempre, configuramos las direcciones IP de los ordenadores de ambas redes de forma apropiada.

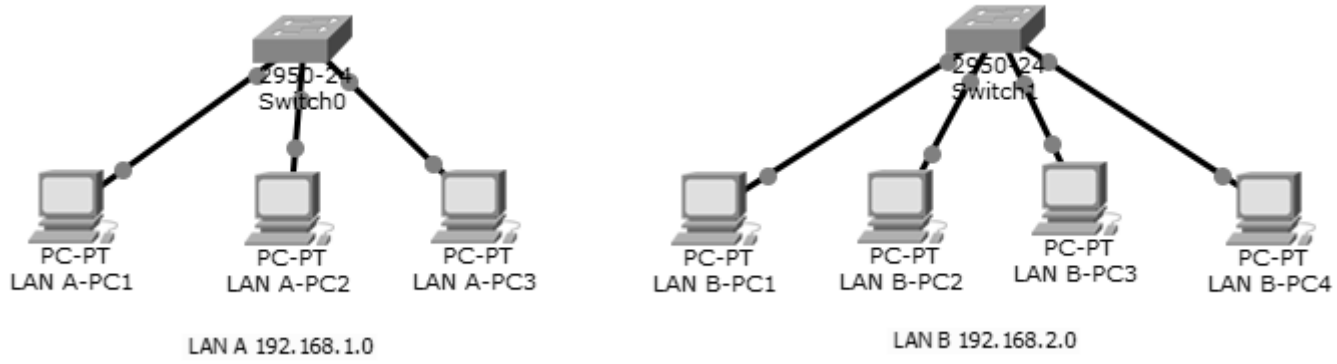


Figura J.1. Dos redes LAN conmutadas aisladas.

Para interconectar ambas redes, necesitamos un rúter. Tomamos un rúter de dos interfaces (rúter 1841) y lo ubicamos entre los dos conmutadores. A continuación, usamos un cable directo para conectar el puerto FastEthernet0/24 del switch0 (LAN A) al interfaz FastEthernet0/0 del rúter. Análogamente, conectamos el puerto FastEthernet0/24 del switch1 (LAN B) al interfaz FastEthernet0/1 del rúter (ver figura J.2). Como vemos, los indicadores luminosos correspondientes a las conexiones entre el rúter y las redes LAN brillan en rojo: Esto significa que las conexiones no están operativas, porque todavía no hemos habilitado los interfaces del rúter ni establecido su configuración IP.

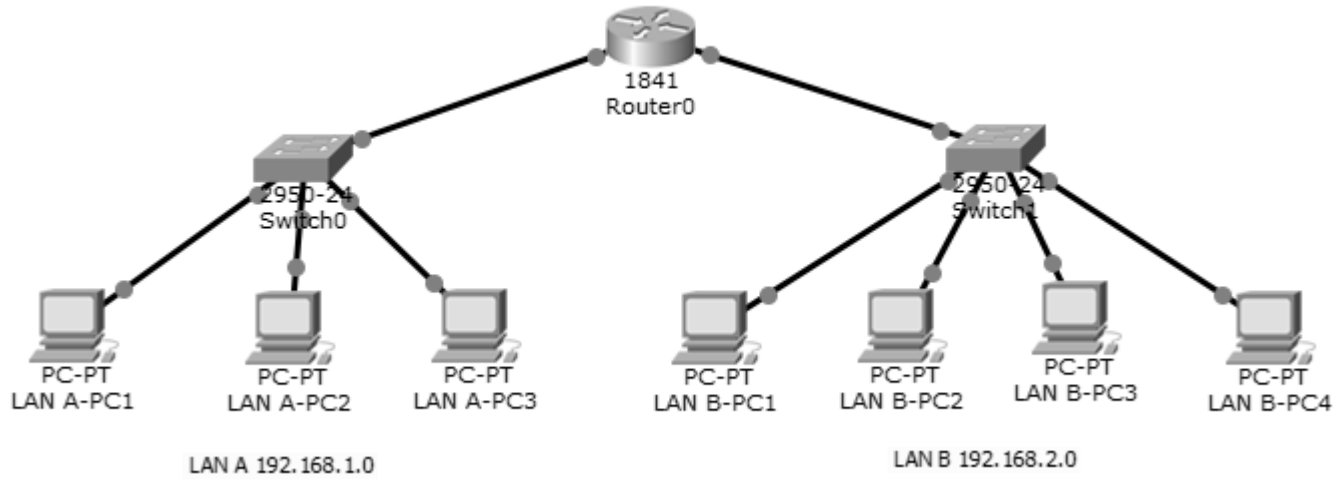


Figura J.2. Dos redes LAN conmutadas conectadas mediante un rúter.

Vamos a solventar estos problemas: Con el ratón, pinchamos en el rúter y accedemos a la pestaña de configuración. Para empezar, habilitamos los puertos de los interfaces FastEthernet0/0 y

FastEthernet0/1, activando la casilla de verificación "Port Status (On)" en ambos interfaces. En la topología de la red, veremos que los indicadores luminosos cambian de rojo a verde.

El siguiente paso es establecer las direcciones IP de ambos interfaces. Al interfaz FastEthernet0/0 (conectado a la red 192.168.1.0) le asignamos la dirección 192.168.1.1 (ver figura J.3), y al interfaz FastEthernet0/1 (conectado a la red 192.168.2.0) le asignamos la dirección 192.168.2.1. Notar que la dirección IP de cada interfaz pertenece a la red a la que se conecta.

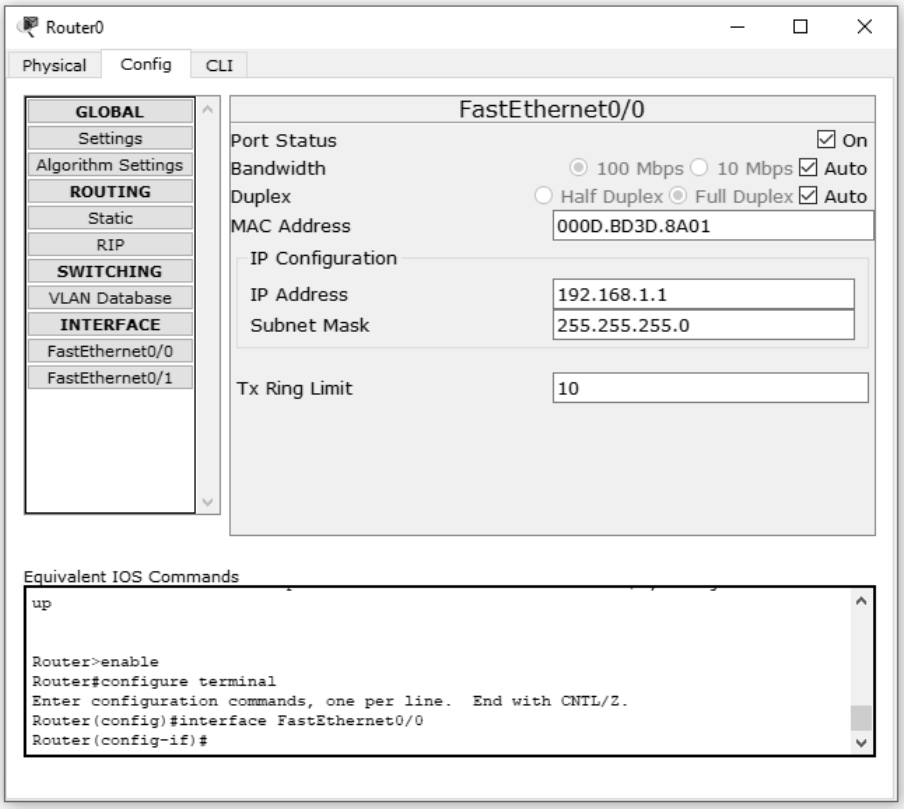


Figura J.3. Configuración del rúter.

Si ahora probamos la conectividad, veremos que dos ordenadores dentro de una misma red LAN pueden comunicarse. Sin embargo, si intentamos mandar un paquete de un ordenador en la LAN A a otro ordenador en la LAN B, comprobaremos que la comunicación no es posible. Esto es normal, porque los ordenadores todavía no saben cómo acceder a la otra red (en otras palabras, los ordenadores no saben cuál es la IP del salto que les permite llegar a la otra red). Para ello, hemos de indicarles la dirección IP del interfaz del rúter que les conecta a la otra red. Esto lo hacemos pinchando en el PC1 de la LAN A, y en la pestaña de configuración, accediendo a los ajustes globales. A continuación, en la sección "Gateway" (**puerta de enlace**) escribimos la dirección IP del interfaz del rúter conectado a la LAN A (a saber, 192.168.1.1), ver figura J.4. Hacemos lo mismo para el resto de PCs de la LAN A. A continuación configuramos de forma análoga los ordenadores de la LAN B, solo que en este caso la dirección IP a usar como puerta de enlace es la IP del interfaz del rúter conectado a la LAN B (esto es, 192.168.2.1).

J.2. SIMULAR LA RED.

Vamos a comprobar si podemos enviar un paquete desde un equipo en la LAN B (por ejemplo, el ordenador LAN B-PC1) a un equipo en la LAN A (por ejemplo, el ordenador LAN A-PC3).

Para empezar, vamos a simplificar la simulación mostrando únicamente los paquetes ICMP. Para ello, pulsamos en el botón "Show All/None" para borrar de la lista de filtros todos los protocolos. A continuación, pulsamos en "Edit Filters", y elegimos mostrar únicamente los paquetes de tipo ICMP.

Ahora mandamos una PDU simple del ordenador LAN B-PC1 al ordenador LAN A-PC3. Como vemos en la lista de eventos de la figura J.5, el ordenador emisor comienza creando el paquete ICMP, y lo envía al conmutador de su red (switch1). El conmutador reenvía el paquete hacia el rúter, que lo encamina hacia el conmutador de la otra red (switch0). Este conmutador reenvía el paquete a la máquina destinataria, que es el ordenador LAN A-PC3. Este ordenador responde enviando un paquete ICMP de respuesta al ordenador LAN B-PC1 a través del switch0, del rúter, y del switch1.

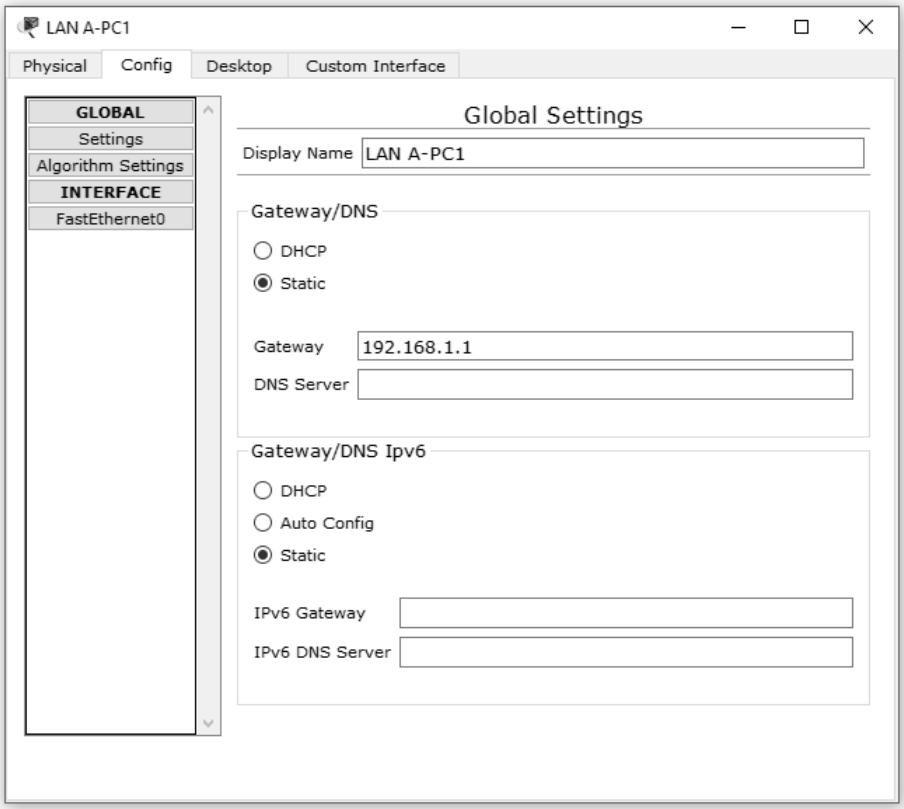


Figura J.4. Configuración del Gateway de los PCs.

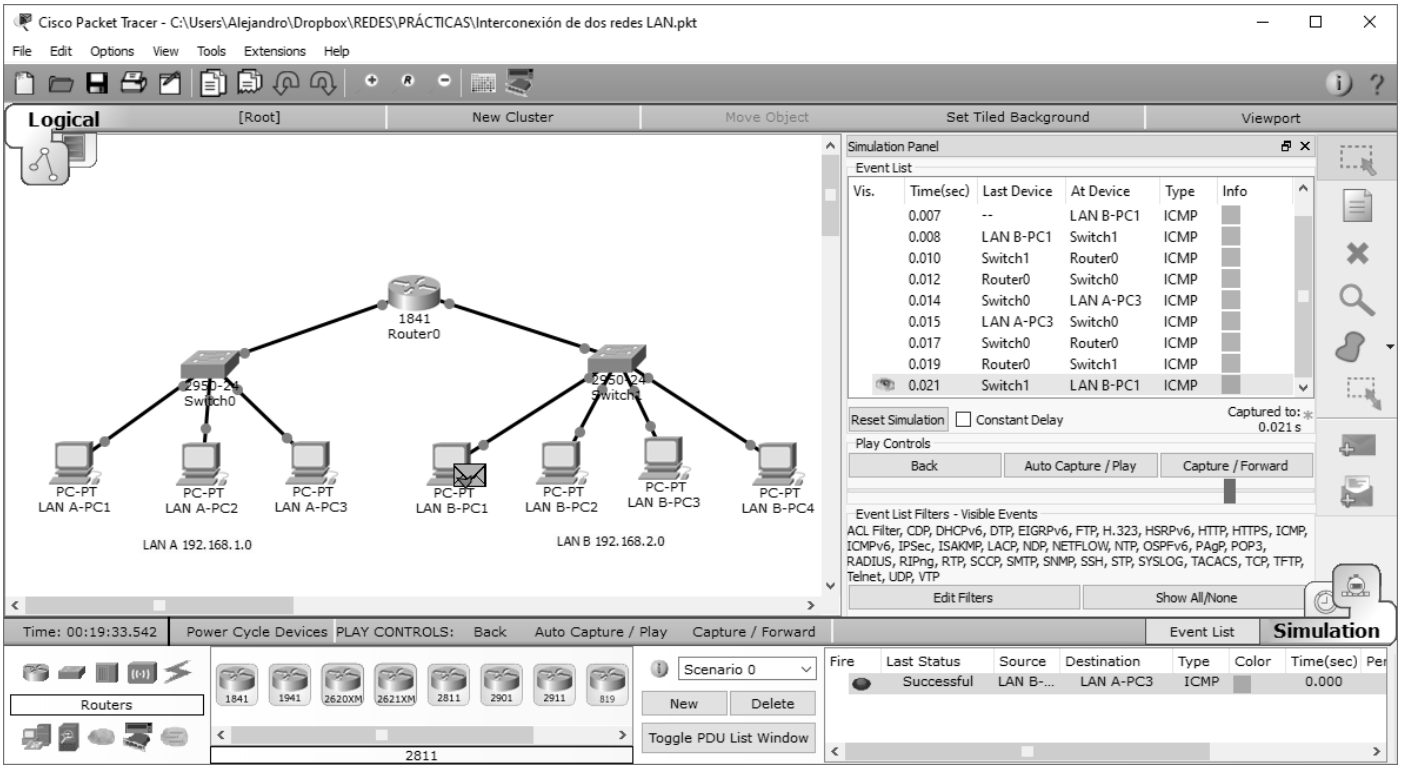


Figura J.5. Simulación de un `ping` del LAN B-PC1 al LAN A-PC3.

4. TECNOLOGÍAS DE REDES LAN.

4.1. INTRODUCCIÓN.

Aunque la pila de protocolos TCP/IP es una arquitectura en cinco capas, el modelo no define ningún protocolo para las capas de enlace y física. En otras palabras, TCP/IP acepta cualquier protocolo que sea capaz de proporcionar servicios a la capa de red. Las capas de enlace y física se implementan en las redes LAN y WAN subyacentes, y en los dispositivos de conexión que las interconectan. Esto significa que cuando hablamos de estas dos capas, en la práctica estamos hablando de las redes que las implementan.

En este capítulo solo daremos una pequeña introducción a dos las tecnologías de red LAN más importantes (Ethernet y WiFi), y a la forma en la que implementan las dos capas más bajas de la arquitectura TCP/IP.

4.2. ETHERNET.

Recordar que una LAN es una red privada que se ha diseñado para dar servicio a un área geográfica limitada, como un aula, un edificio, o un campus. Aunque las LANs pueden usarse como redes aisladas (por ejemplo, para interconectar los ordenadores de una organización con objeto de compartir recursos), en la práctica la mayoría de las LANs están conectadas a una red de área amplia (WAN) o a Internet.

El mercado ofrece distintas tecnologías para redes LAN de cable, como Ethernet, Token Ring, Token Bus, FDDI, etc. Algunas de estas tecnologías tuvieron éxito durante algún tiempo, pero **Ethernet** es hoy en día la tecnología dominante. Por ello, en esta sección nos centraremos en el estudio de la tecnología Ethernet para redes locales cableadas. Aunque Ethernet ya ha pasado por cuatro generaciones, su operación básica sigue siendo la misma. Ethernet ha evolucionado para satisfacer la demanda de mayores velocidades de transmisión, y para competir con las nuevas tecnologías emergentes.

FORMATO DE LA TRAMA.

Los paquetes intercambiados en las redes LAN Ethernet se denominan **tramas**. En esta sección discutimos el formato y la longitud de la trama usada en las diferentes versiones de Ethernet.

La trama Ethernet contiene siete campos: Preámbulo, delimitador de comienzo de trama (SFD), dirección de destino (DA), dirección de origen (SA), longitud o tipo de la unidad de datos, datos de la capa superior, y CRC. Ethernet no proporciona ningún mecanismo para confirmar la recepción de las tramas enviadas, dando lugar a lo que se conoce como un medio no fiable. Las confirmaciones de recepción deben implementarse en las capas superiores. La figura 6.1 muestra el formato de la trama.

- **Preámbulo**. El primer campo de la trama contiene 7 bytes (56 bits) de 0s y 1s alternados que sirven para alertar al sistema receptor de la llegada de una nueva trama. Este patrón únicamente proporciona una alerta y un pulso de sincronización, y permite que las estaciones puedan perderse algunos de los bits del comienzo de la trama.
- **Delimitador de comienzo de trama (SFD)**. El segundo campo (1 byte: 10101011) señaliza el inicio de la trama. Los dos últimos bits son 11, y alertan al receptor de que el siguiente campo es la dirección de destino.
- **Dirección de destino (DA)**. El campo DA es de 6 bytes, y contiene la dirección física de la estación o estaciones destinatarias de la trama. En breve discutiremos más detalladamente el esquema de direccionamiento empleado en Ethernet.
- **Dirección de origen (SA)**. El campo SA también es de 6 bytes, y contiene la dirección física de la estación emisora de la trama.

- **Longitud o tipo**. La versión original de Ethernet usaba este campo para definir el protocolo de capa superior encapsulado dentro de la trama. Los actuales estándares IEEE usan este campo para indicar el número de bytes contenidos dentro del campo de datos. Ambos usos son muy comunes en la actualidad.
- **Datos**. Este campo transporta los datos encapsulados de los protocolos de la capa superior. Tiene un tamaño mínimo de 46 bytes y un tamaño máximo de 1500 bytes.
- **CRC**. El último campo contiene información para posibilitar la detección de errores. En este caso, se trata de un mecanismo de comprobación de redundancia cíclica (CRC = Cyclic Redundancy Check).

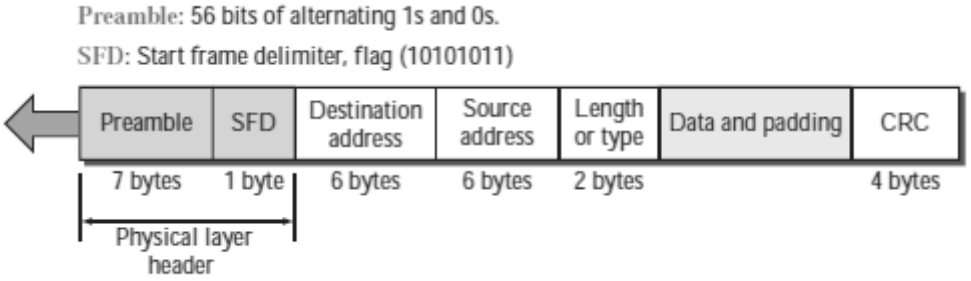


Figura 6.1. Formato de la trama Ethernet.

DIRECCIONAMIENTO.

Todas las estaciones de una red Ethernet (como por ejemplo, un PC, una estación de trabajo, o una impresora) tienen su propio **adaptador de red** o **tarjeta de interfaz de red** (**NIC**). La NIC se inserta en la estación y le otorga una **dirección física**. Las direcciones físicas Ethernet son de 6 bytes, y suelen representarse con 12 dígitos hexadecimales, donde cada byte (cada pareja de dígitos hexadecimales) está separado del siguiente por dos puntos. Un ejemplo de dirección física Ethernet es 4A:30:10:21:10:1A.

Direcciones unicast, multicast, y broadcast.

La dirección de origen siempre es una dirección unicast: La trama la envía una sola estación. Sin embargo, la dirección de destino puede ser unicast, multicast, o broadcast. Una dirección de destino unicast define un único destinatario; la relación entre el emisor y el receptor es de uno a uno. Una dirección de destino multicast define un grupo de direcciones; la relación entre el emisor y los receptores es de uno a muchos. Las direcciones broadcast es un caso especial de dirección multicast; los destinatarios son todos los equipos de la red.

La figura 6.2 muestra cómo distinguir una dirección unicast de una dirección multicast. Si el bit menos significativo del primer byte es 0, la dirección es unicast; si es 1, la dirección es multicast. Una forma más sencilla de discriminarlas es mirar el segundo dígito hexadecimal comenzando desde la izquierda: Si es par, la dirección es unicast; si es impar, la dirección es multicast.

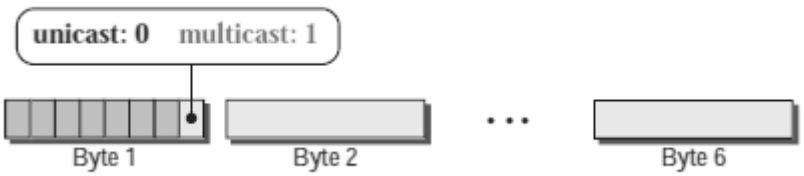


Figura 6.2. Direcciones unicast y multicast.

Un ejemplo de dirección unicast es 4A:30:10:21:10:1A, porque A es el dígito hexadecimal que se corresponde con el valor decimal 12 (número par), que en binario es 1100. Un ejemplo de dirección multicast es 47:20:1B:2E:08:EE, porque 7 es el dígito hexadecimal que se corresponde con el valor decimal 7 (número impar), que en binario es 0111. La dirección broadcast siempre es FF:FF:FF:FF:FF:FF (donde el valor hexadecimal F se corresponde con el valor decimal 15, que en binario es 1111).

ETHERNET ESTÁNDAR.

La tecnología Ethernet fue desarrollada por Xerox en 1976, y desde entonces ha pasado por cuatro generaciones: **Ethernet estándar** ($10\ Mbps$), **Fast Ethernet** ($100\ Mbps$), **Gigabit Ethernet** ($1\ Gbps$), y **10 – Gigabit Ethernet** ($10\ Gbps$). Aquí comenzamos estudiando la tecnología Ethernet estándar (o tradicional). Aunque el estándar Ethernet original ($10\ Mbps$) es hoy día una tecnología obsoleta, vamos a discutir sus características para poder entender las versiones más recientes.

Método de acceso: CSMA/CD.

El estándar IEEE 802.3 define el **acceso múltiple por escucha de portadora con detección de colisión** (**CSMA/CD** = Carrier Sense Multiple Access with Collision Detection) como método de acceso de la tecnología Ethernet tradicional. En una red Ethernet tradicional, las estaciones pueden interconectarse mediante una topología física en bus o en estrella (con hub), pero la topología lógica siempre es en bus. Eso significa que el medio (el canal) se comparte entre todas las estaciones, y que solo una estación puede usarlo en un instante dado. Esto también implica que todas las estaciones reciben la trama transmitida por una estación (a esto se le denomina **difusión**, o *broadcasting*). La estación destinataria conserva la trama; el resto la descartan. En esta situación, ¿cómo podemos asegurar que dos estaciones no usarán el medio al mismo tiempo? Si lo hacen sus tramas *colisionarán*, y la información será irrecuperable.

El método CSMA/CD se creó para minimizar la probabilidad de colisión, y por consiguiente, para mejorar el funcionamiento de la red. La probabilidad de colisión puede reducirse si las estaciones "escuchan" el medio antes de intentar transmitir. El **acceso múltiple por escucha de portadora** (**CSMA**) requiere que las estaciones comprueben el estado del medio antes de transmitir. CSMA permite reducir la probabilidad de colisión, pero no consigue eliminarla. La razón para ello se muestra en la figura 6.3.

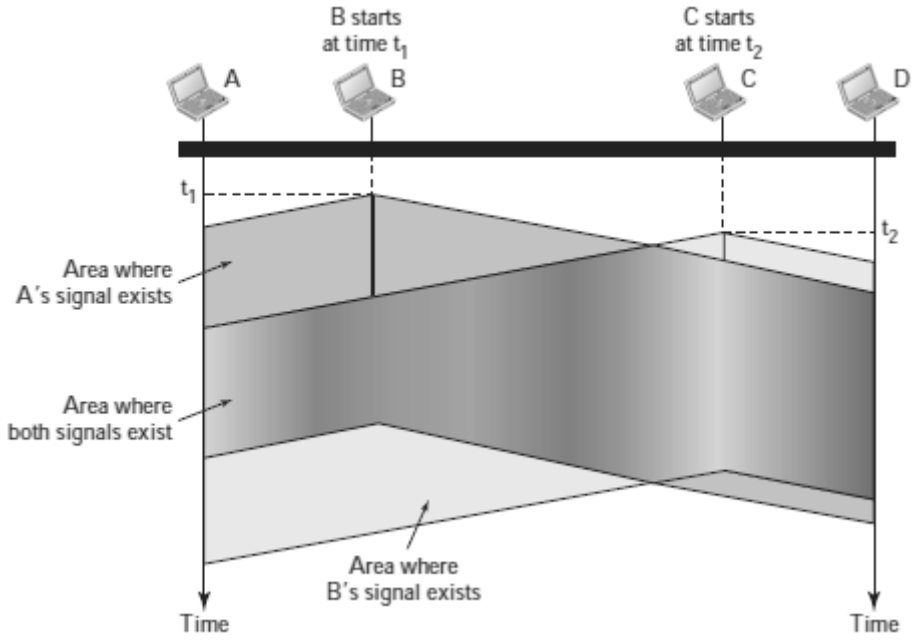


Figura 6.3. Diagrama espacio – temporal de una colisión en CSMA.

En la figura tenemos cuatro estaciones (A, B, C, y D) conectadas a un canal compartido. La posibilidad de colisión sigue existiendo debido al retardo en la propagación de las señales. Cuando una estación envía una trama, el primer bit de la trama tarda un tiempo (muy pequeño) en llegar al resto de estaciones. Esto significa que una estación podría detectar que el canal está desocupado, pero solo porque aún no ha recibido el primer bit de la trama enviada por otra estación.

En el instante t_1 la estación B escucha el medio y lo encuentra desocupado, así que envía una trama. Un instante de tiempo posterior t_2 la estación C escucha el medio y también lo encuentra desocupado, porque en este instante los primeros bits de la estación B todavía no han llegado a la estación C. La estación C también envía una trama. Las dos señales colisionan y las dos tramas se destruyen.

El **acceso múltiple por escucha de portadora con detección de colisión** (CSMA/CD) mejora el algoritmo CSMA para gestionar las colisiones. En este método, las estaciones monitorizan el medio después de haber enviado una trama, para comprobar si la transmisión tuvo éxito. De ser así, la transmisión termina. Pero si hay una colisión, la trama vuelve a enviarse. Para entender mejor CSMA/CD, vamos a observar los primeros bits transmitidos por dos estaciones implicadas en una colisión. Aunque las dos estaciones continúan enviando bits hasta que detectan la colisión, veamos qué ocurre cuando colisionan los primeros bits. La figura 6.4 muestra la situación cuando las estaciones implicadas son los ordenadores A y C.

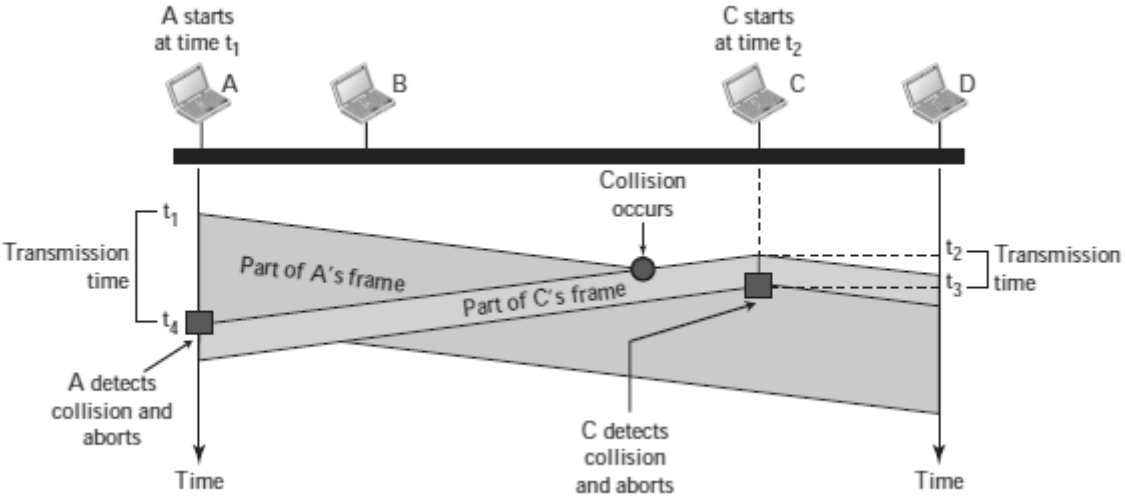


Figura 6.4. Colisión del primer bit en CSMA/CD.

En el instante t_1 la estación A empieza a enviar los bits de su trama. En t_2 la estación C todavía no ha detectado el primer bit enviado por A; por consiguiente, la estación C empieza a enviar los bits de su trama, que se propagan tanto hacia la derecha como hacia la izquierda. La colisión ocurre un poco después del instante t_2. La estación C detecta la colisión en el instante t_3, cuando recibe el primer bit de la trama de A. Inmediatamente, la estación C aborta la transmisión. Por su parte, la estación A detecta la colisión en el instante t_4, cuando recibe el primer bit de C, y aborta la transmisión inmediatamente.

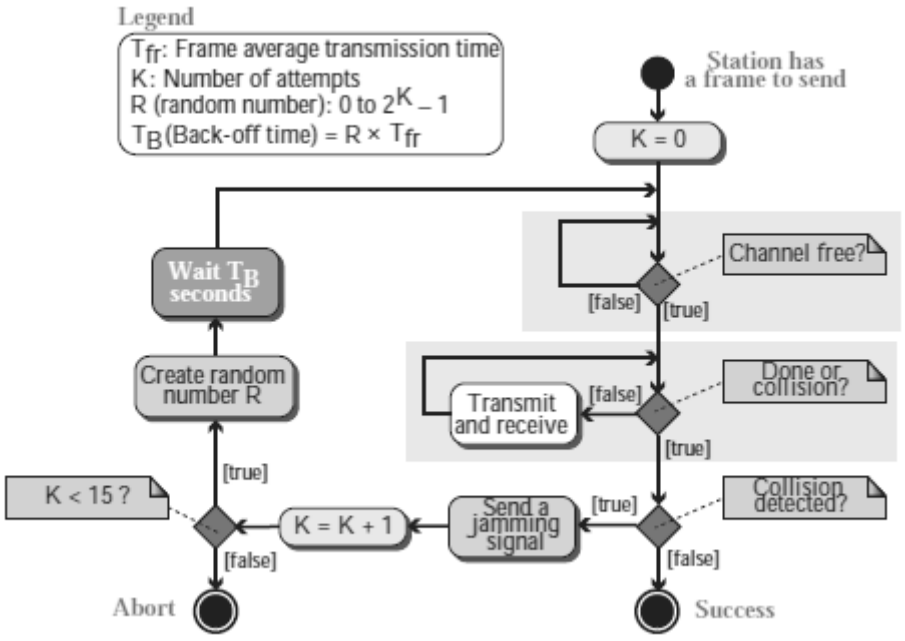


Figura 6.5. Diagrama de flujo de CSMA/CD.

Procedimiento.

La figura 6.5 muestra el diagrama de flujo del algoritmo CSMA/CD. Cuando una estación tiene datos que enviar, se pone a escuchar el canal antes de comenzar a transmitir una trama. Cuando la estación encuentra el canal desocupado no transmite la trama entera, sino que transmite y recibe de forma simultánea para poder detectar una posible colisión. La figura usa un bucle para mostrar que el proceso de transmisión y recepción es continuo: La estación monitoriza constantemente para poder detectar una de estas dos condiciones: (a) La transmisión termina, o (b) se detecta una colisión. Cualquiera de estos dos eventos detiene el proceso continuo de transmisión y recepción. Si no se detecta colisión, la transmisión finaliza: Se ha transmitido toda la trama. Si ha ocurrido una colisión, la estación envía de una pequeña señal de interferencia (*jamming signal*) que refuerza la colisión en el caso de que otras estaciones todavía no la hayan detectado. Además, la estación incrementa le valor de un contador K, y evalúa su valor. Si $K = 15$ (esto es, si ya han ocurrido 15 colisiones), la estación aborta la transmisión; si $K < 15$ la estación espera un tiempo aleatorio antes de volver a ponerse a escuchar el canal. La práctica K propone escribir un pequeño programa en Python que implemente una versión simplificada del método CSMA/CD.

Implementación.

La tabla 6.1 muestra un resumen de las distintas implementaciones de Ethernet Estándar que acabaron implantándose. En la nomenclatura 10Base–X, el número indica la velocidad de transmisión ($10\ Mbps$), el término Base significa transmisión digital en banda base (sin modular), y X define aproximadamente la longitud máxima del cable de transmisión en 100 metros (por ejemplo, 5 para 500 metros, o 2 para 185 metros), o el tipo de cable (donde T es cable de pares sin apantallar (UTP), y F es fibra óptica).

Características	10Base5	10Base2	10Base-T	10Base-F
Medio	Coaxial grueso	Coaxial delgado	2 UTP	2 fibras
Longitud máxima (metros)	500	185	100	2000

CAMBIOS EN EL ESTÁNDAR.

Antes de estudiar las versiones Ethernet a más alta velocidad, debemos revisar los cambios por los que pasó la tecnología Ethernet Estándar a $10\ Mbps$. Esto cambios impulsaron la evolución de la tecnología Ethernet, lo que le permitió competir con otras tecnologías LAN más rápidas (como Token Ring), e incluso llegar a superarlas.

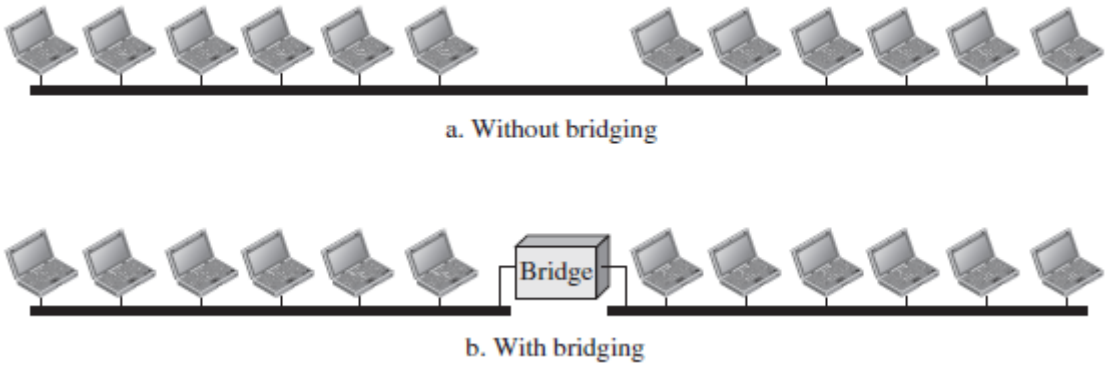


Figura 6.7. Ethernet Estándar y Ethernet puenteada.

Ethernet puenteada.

El primer paso en la evolución de Ethernet fue la división de una LAN mediante **puentes** (bridges), ver figura 6.7. Como ya estudiamos en el capítulo 3, los puentes permiten segmentar una red en múltiples porciones, con objeto de incrementar el ancho de banda compartido y separar dominios de colisión. Por ejemplo, en la red LAN sin puentear de la figura el ancho de banda total ($10\ Mbps$) debe compartirse entre

12 estaciones, pero en la red puenteada el ancho de banda solo se comparte entre 7 estaciones (los 6 ordenadores de cada segmento más el puente). En la red no puenteada hay un único dominio de colisión con 12 estaciones, mientras que en la red puenteada tenemos dos dominios de colisión con solo 6 estaciones, lo que reduce la probabilidad de colisión.

Ethernet conmutada.

Los primeros puentes solo contaban con dos o cuatro puertos de salida, que permitían dividir la red en dos o cuatro segmentos. Pero la idea de una LAN puenteada puede generalizarse para tener LANs conmutadas. Así, en lugar de dividir una LAN en dos o cuatro segmentos, ¿por qué no dividirla en N segmentos, donde N es el número total de estaciones de la LAN? En otras palabras, ¿por qué no usar un puente multipuerto con N puertos de salida? De esta forma, el ancho de banda solo se comparte entre una estación y el conmutador ($5\ Mbps$ para cada uno), y los dominios de colisión se reducen a un único equipo.

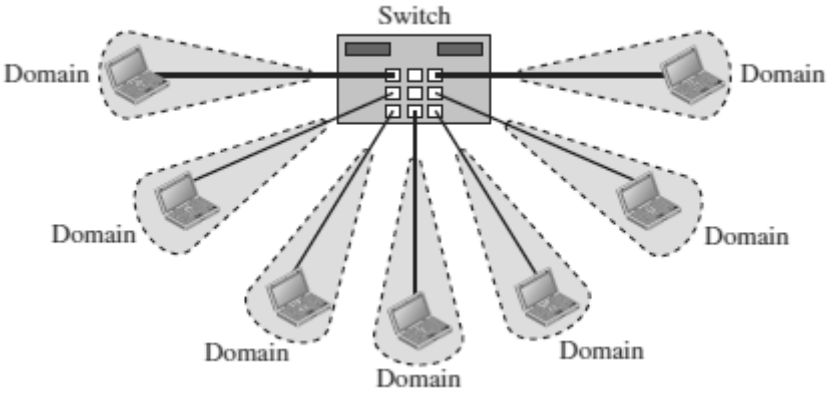


Figura 6.8. LAN Ethernet conmutada.

Un **conmutador de capa de enlace** es un puente de N puertos con la sofisticación adicional de poder procesar los paquetes de forma mucho más rápida. La evolución de la tecnología Ethernet puenteada a la tecnología **Ethernet conmutada** fue el paso que permitió abrir el camino a versiones Ethernet más rápidas, como veremos a continuación. La figura 6.8 muestra una LAN Ethernet conmutada.

Ethernet full – dúplex.

Una de las limitaciones de 10Base5 y 10Base2 es que la comunicación es half – dúplex (10Base-T siempre es full - dúplex). Esto significa que una estación puede enviar o recibir, pero nunca al mismo tiempo. El siguiente paso en la evolución de Ethernet fue pasar de una Ethernet conmutada half – dúplex a una Ethernet conmutada full – dúplex. El modo full – dúplex aumenta la capacidad de cada dominio de 10 a $20\ Mbps$. La figura 6.9 muestra una LAN Ethernet conmutada en modo full – dúplex. Notar que en vez de usar un enlace entre la estación y el conmutador, la configuración usa dos enlaces: Uno para transmitir y otro para recibir.

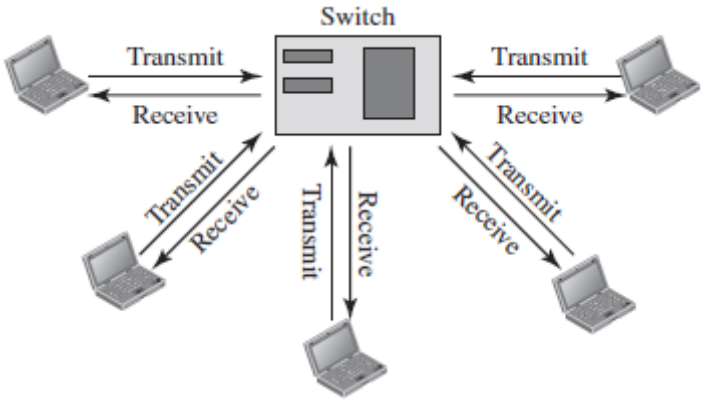


Figura 6.9. LAN Ethernet conmutada full – dúplex.

En la tecnología Ethernet conmutada full – dúplex no hay necesidad del método CSMA/CD. Como están conectadas al conmutador mediante dos enlaces separados, cada estación puede enviar y recibir independientemente sin preocuparse de que pueda ocurrir una colisión. Cada enlace es una vía punto a punto dedicada entre la estación y el conmutador. Por consiguiente, ya no es necesaria la escucha de portadora ni la detección de colisión.

FAST ETHERNET.

Fast Ethernet se diseñó para competir con tecnologías LAN como FDDI y Canal de Fibra (Fiber Channel). La tecnología Fast Ethernet es compatible con Ethernet Estándar, pero es capaz de transmitir a una velocidad 10 veces superior, esto es, a $100\ Mbps$. Los objetivos principales de Fast Ethernet son los siguientes:

- Mejorar la velocidad de transmisión de datos a $100\ Mbps$.
- Ser compatible con Ethernet Estándar.
- Usar las mismas direcciones de 48 bits.
- Mantener el mismo formato de trama.

Sin embargo, en el estándar que desarrolló esta tecnología se decidió eliminar la topología en bus y conservar únicamente la topología en estrella. Para la topología en estrella hay dos opciones: Half – dúplex y full – dúplex. En el método half - dúplex las estaciones se conectan a un concentrador (hub); en el método full – dúplex la conexión se hace con un conmutador (switch).

En el enfoque half – dúplex, el método de acceso al medio es el mismo (CSMA/CD). En cambio, en Fast Ethernet full – dúplex no hay necesidad de aplicar CSMA/CD, pero se sigue implementando para proporcionar la compatibilidad con Ethernet Estándar.

Autonegociación.

La **autonegociación** es una nueva característica de la tecnología Fast Ethernet, y les permite a dos dispositivos (estaciones o hubs) negociar el modo de operación y la velocidad de transmisión. La autonegociación se diseñó para cumplir con los siguientes propósitos:

- Permitir la interconexión de dispositivos incompatibles. Por ejemplo, un dispositivo con una capacidad máxima de $10\ Mbps$ puede comunicarse con un dispositivo con una capacidad de $100\ Mbps$, operando a la velocidad más baja.
- Permitir que un dispositivo pueda tener múltiples capacidades (por ejemplo, $10\ Mbps$ o $100\ Mbps$).
- Permitir que una estación pueda comprobar las capacidades de un hub.

Implementación.

La implementación de Fast Ethernet a nivel de capa física puede hacerse con dos cables o con cuatro cables. Para la implementación con dos cables, éstos pueden ser cables de pares apantallados (STP) o fibra óptica. La implementación con cuatro cables solo utiliza cable de pares sin apantallar (UTP). La tabla 6.2 muestra un resumen de las implementaciones Fast Ethernet.

Características	100Base-TX	100Base-FX	100Base-T4
Medio	STP	Fibra	UTP
Número de cables	2	2	4
Longitud máxima (metros)	100	100	100

GIGABIT ETHERNET.

La necesidad de velocidades de transmisión superiores a $100\,Mbps$ condujo al desarrollo de Gigabit Ethernet ($1000\,Mbps = 1\,Gbps$). Los objetivos principales de Gigabit Ethernet son los siguientes:

- Elevar la velocidad de transmisión a $1\,Gbps$.
- Ser compatible con las tecnologías Estándar Ethernet y Fast Ethernet.
- Utilizar las mismas direcciones de 48 bits.
- Usar el mismo formato de trama.
- Soportar la autonegociación definida por Fast Ethernet.

Para conseguir una tasa de transferencia de datos de $1\,Gbps$, Gigabit Ethernet utiliza dos sistemas distintos para el acceso al medio: Half – dúplex y full – dúplex. Casi todas las implementaciones de Gigabit Ethernet utilizan el sistema full – dúplex. Sin embargo, aquí también discutiremos el sistema half – dúplex para mostrar que Gigabit Ethernet puede ser compatible con las generaciones previas.

Modo full – dúplex.

En modo full – dúplex hay un conmutador central que se conecta a todos los ordenadores o a los demás conmutadores. En este modo no hay posibilidad de colisión, lo que significa que CSMA/CD no es necesario. La ausencia de colisiones implica que la longitud máxima del cable viene determinada por la atenuación de la señal en el cable, y no por el proceso de detección de colisión.

Modo half – dúplex.

Gigabit Ethernet también puede usarse en modo half – dúplex, aunque es bastante inusual. En este caso, el conmutador (switch) puede reemplazarse por un concentrador (hub), que actúa como el cable común en el que pueden ocurrir colisiones. Ello implica que el modo half – dúplex necesita usar CSMA/CD.

Implementación.

La tabla 6.3 es un resumen de las distintas implementaciones de Gigabit Ethernet.

Características	1000Base-SX	1000Base-LX	1000Base-CX	1000Base-T4
Medio	Fibra de onda corta	Fibra de onda larga	STP	UTP Cat. 5
Número de cables	2	2	2	4
Longitud máxima (metros)	550	5000	25	100

10-GIGABIT ETHERNET.

Los objetivos fundamentales de 10-Gigabit Ethernet pueden resumirse como sigue:

- Aumentar la velocidad de transmisión a $10\,Gbps$.
- Ser compatible con las tecnologías Estándar Ethernet, Fast Ethernet, y Gigabit Ethernet.
- Utilizar las mismas direcciones de 48 bits.
- Usar el mismo formato de trama.
- Permitir la interconexión de las redes LAN existentes para formar redes WAN.
- Hacer Ethernet compatible con tecnologías WAN como Frame Relay y ATM.

Implementación.

10-Gigabit Ethernet solo opera en modo full – dúplex, lo que significa que no hay necesidad de contención por el acceso al medio: No necesita usar el método CSMA/CD. Las tres implementaciones más comunes son 10GBase-S, 10GBase-L, y 10GBase-E. La tabla 6.4 resume las características de estas implementaciones:

Características	10GBase-S	10GBase-L	10GBase-E
Medio	Fibra multi-modo	Fibra mono-modo	Fibra mono-modo
Número de cables	2	2	2
Longitud máxima (metros)	300	10000	40000

4.3. ESTÁNDAR 802.11: WiFi.

Las comunicaciones inalámbricas son actualmente unas de las tecnologías de comunicación más pujantes. La posibilidad de conectar dispositivos a la red sin usar cables es una demanda muy extendida en la actualidad. Las redes inalámbricas están presentes en los hogares, en las escuelas, en los campus universitarios, en los edificios de oficinas, en sedes oficiales, y en muchas otras áreas públicas. Existen varias tecnologías de conexión inalámbricas (WiMAX, Bluetooth, etc.), cada una de ellas con un ámbito de aplicación, pero en esta sección nos centraremos en la tecnología inalámbrica para redes LAN definida en el **estándar IEEE 802.11**, más conocida como **WiFi**.

ESTÁNDAR IEEE 802.11.

El instituto de normalización IEEE ha definido en su **estándar IEEE 802.11** las especificaciones para el funcionamiento de las redes LAN inalámbricas. Este estándar cubre las capas física y de enlace de datos.

Arquitectura.

El estándar define dos tipos de servicio: El conjunto de servicios básico (BSS = Basic Service Set), y el conjunto de servicios ampliado (ESS = Extended Service Set).

Conjunto de servicios básico. IEEE 802.11 define el **conjunto de servicios básico** (**BSS**) como el bloque de funcionamiento básico de la tecnología LAN inalámbrica. Un BSS está formado por una o varias estaciones inalámbricas móviles o estacionarias, y por una estación base central opcional denominada **punto de acceso** (**AP**). La figura 6.10 muestra las dos posibilidades para un BSS.

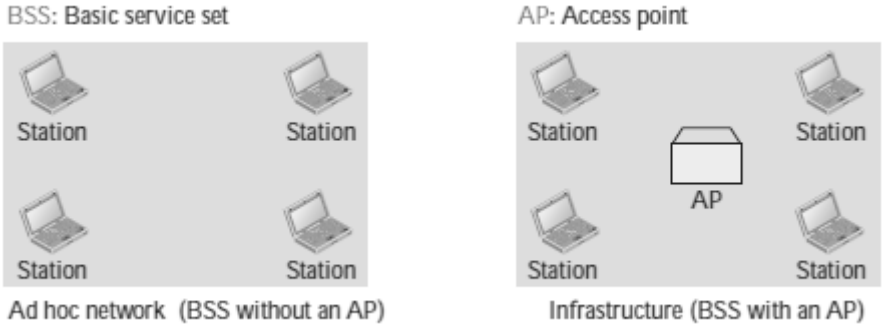


Figura 6.10. Los dos tipos de conjuntos de servicios básicos (BSSs).

El BSS sin AP constituye una red aislada y no puede enviar datos a otros BSSs. A esta forma de funcionamiento se la denomina **modo ad hoc**. En esta arquitectura, las estaciones pueden formar una red sin necesidad de un AP; las estaciones se conectan inalámbricamente unas a otras para formar el BSS. Por otra parte, a la arquitectura de un BSS con un AP se le denomina **modo infraestructura**.

Conjunto de servicios ampliado. Un **conjunto de servicios ampliado** (**ESS**) está formado por dos o más BSSs con sus respectivos APs. En este caso, las BSSs están conectadas mediante un **sistema de distribución**, que suele ser una LAN cableada. El sistema de distribución interconecta los APs de los distintos BSSs. El estándar 802.11 no especifica la tecnología del sistema de distribución, que puede ser cualquier tecnología de red LAN (por ejemplo, Ethernet). Notar que el conjunto de servicios ampliado usa dos tipos de estaciones: Móviles y estacionarias (fijas). Las estaciones móviles son las estaciones normales dentro de un BSS. Las estaciones estacionarias son los APs que forman parte de la LAN cableada. La figura 6.11 muestra un ESS.

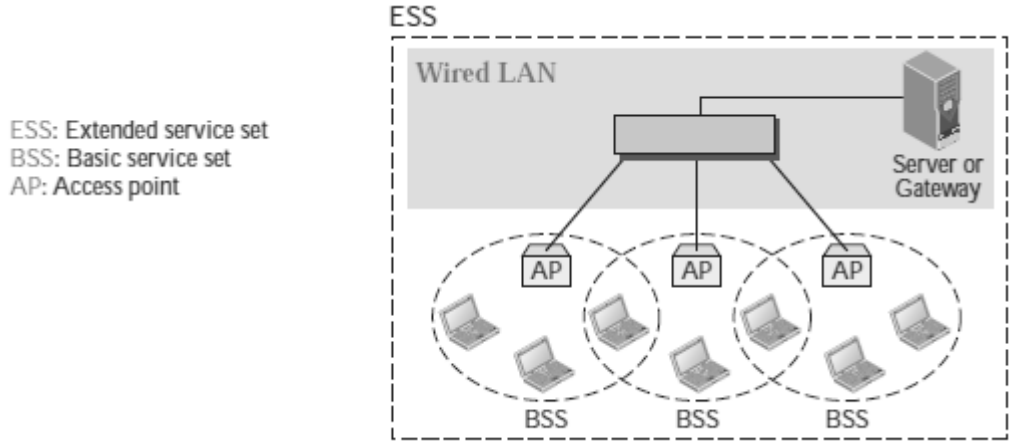


Figura 6.11. Un ejemplo de conjunto de servicios ampliado (ESS).

Cuando varios BSSs están conectados, las estaciones dentro de la zona de cobertura de un BSS pueden comunicarse sin necesidad de un AP. Sin embargo, las comunicaciones entre estaciones pertenecientes a dos BSSs distintos suele ocurrir a través de sus respectivos APs.

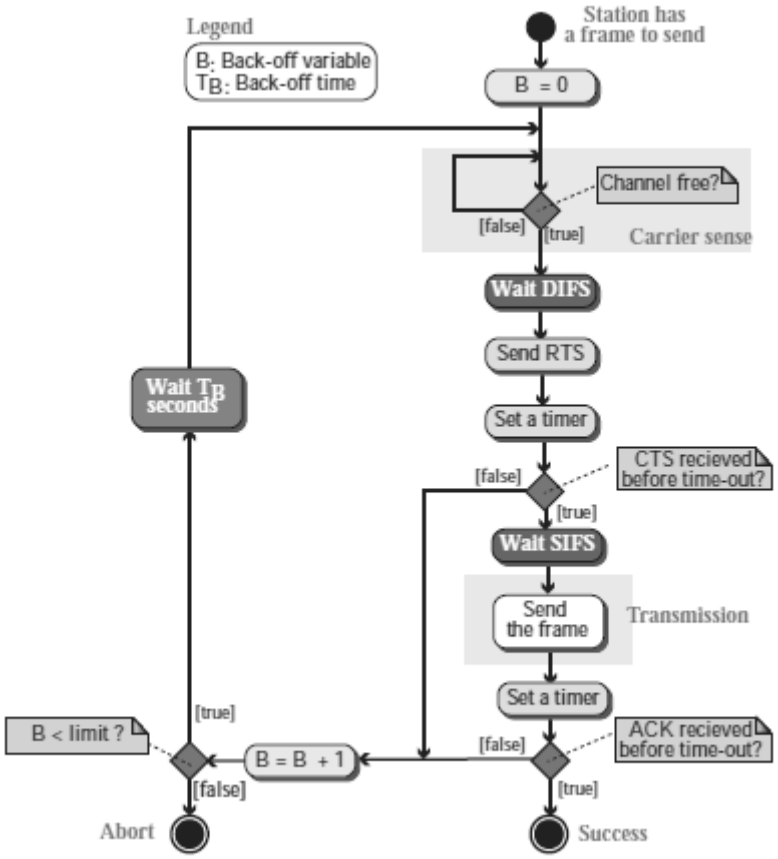


Figura 6.12. Diagrama de flujo de CSMA/CA.

ACCESO AL MEDIO.

El método de acceso al medio definido para el protocolo 802.11 es el **acceso múltiple por escucha de portadora con prevención de colisión** (**CSMA/CA** = Carrier Sense Multiple Access with Collision Avoidance). La figura 6.12 muestra el diagrama de flujo de CSMA/CA. Más adelante explicaremos este método de acceso. Las LANs inalámbricas no implementan el método CSMA/CD por tres razones:

1) Para que una estación pueda detectar una colisión debe ser capaz de enviar datos y de recibir señales de colisión al mismo tiempo. Esto implicaría estaciones muy costosas y un mayor ancho de banda.
2) La colisión podría no detectarse debido al **problema de nodo oculto**, del que hablaremos más adelante.
3) La distancia entre las estaciones puede ser muy grande. El desvanecimiento de la señal con la distancia podría hacer que una estación en un extremo no escuchase una colisión en el otro extremo.

Proceso de intercambio de tramas.

La figura 6.13 muestra el proceso de intercambio de tramas de datos y de control a lo largo del tiempo:

1) Antes de enviar una trama, la estación origen escucha el medio comprobando el nivel de energía en la frecuencia portadora.
   a. La estación usa una estrategia de persistencia hasta que encuentra el canal desocupado.
   b. Cuando la estación encuentra el canal desocupado, se pone a la espera durante un periodo de tiempo llamado **espacio de intertrama distribuido** (**DIFS** = Distributed InterFrame Space); a continuación, la estación envía una trama de control llamada **solicitud de envío** (**RTS** = Request To Send).
2) Después de recibir el RTS y de esperar un periodo de tiempo llamado **espacio de intertrama corto** (**SIFS** = Short InterFrame Space), la estación destino envía una trama de control llamada **listo para recibir** (**CTS** = Clear To Send) a la estación origen. Esta trama de control informa de que la estación destino está preparada para recibir datos.
3) La estación origen envía los datos después de volver a esperar una cantidad de tiempo igual al SIFS.
4) Después de esperar un periodo de tiempo igual al SIFS, la estación destino envía una trama de **confirmación de recepción** (ACK = Acknowledgement) para informar de que los datos se han recibido correctamente. Este protocolo necesita confirmación de recepción porque la estación origen no tiene ninguna forma de saber si los datos se recibieron correctamente en el destino.

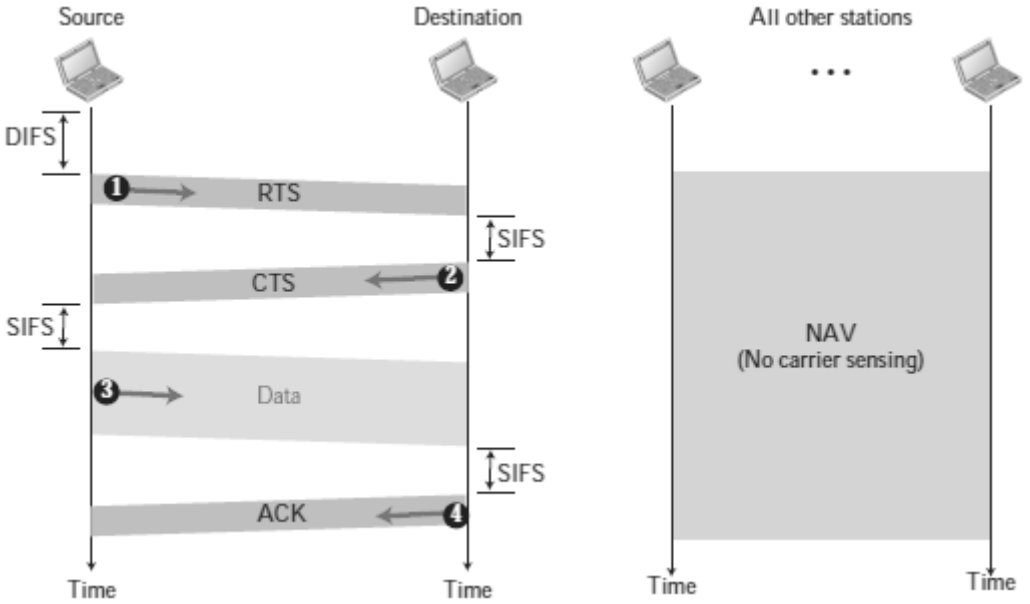


Figura 6.13. CSMA/CA y NAV.

Vector de asignación de red.

Cuando una estación consigue acceder al medio, ¿qué hacen el resto de estaciones para aplazar el envío de sus datos? En otras palabras, ¿Cómo se implementa la **prevención de colisión** (CA) en el método CSMA/CA? Esto se consigue con el **vector de asignación de red** (**NAV** = Network Allocation Vector). Cuando una estación envía una trama RTS, incluye el tiempo que necesitará ocupar el canal. Las estaciones que se ven afectadas por esta transmisión crean un temporizador llamado *vector de asignación de red* (NAV) que indica cuánto tiempo debe pasar antes de que puedan volver a escuchar el canal. Cada vez que una estación accede al medio y envía una trama RTS, las otras estaciones arrancan su temporizador NAV. Dicho de otra forma, antes de ponerse a escuchar el medio físico para detectar si está desocupado, las estaciones primero comprueban si su NAV ha expirado. ¿Qué ocurre si hay una colisión durante el periodo de tiempo en el que se están intercambiando las tramas RTS y CTS? Dos o más estaciones podrían intentar enviar tramas RTS al mismo tiempo, y esas tramas de control podrían colisionar. Como no hay mecanismo de control de colisión, la estación emisora asume que ha habido colisión si no recibe la trama CTS del receptor. En ese caso, se emplea la estrategia de persistencia, y el emisor lo intenta otra vez.

Problemas de estación oculta y de estación expuesta.

Problema de estación oculta. La figura 6.14 muestra un ejemplo del **problema de nodo oculto**. La estación B tiene una zona de cobertura representada por el óvalo izquierdo (realmente, una esfera en el espacio). Toda estación dentro de esta zona puede oír las señales emitidas por B. Por su parte, la estación C tiene una zona de cobertura representada por el óvalo derecho, y toda estación dentro de esta zona puede oír las señales transmitidas por C. La estación C esta fuera de la zona de cobertura de la estación B, y viceversa. Pero la estación A está en un área cubierta tanto por B como por C, y puede oír las señales emitidas por B y C.

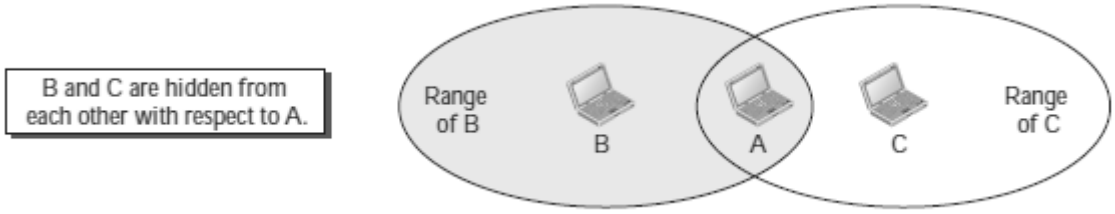


Figura 6.14. Problema de estación oculta.

Suponer que la estación B está enviando datos a la estación A. Además, imaginar que a mitad de esta transmisión, la estación C también necesita enviar datos a la estación A. El problema es que la estación C está fuera del alcance de B, y las transmisiones de B no llegan a C. Por lo tanto, C piensa que el medio está desocupado, y envía sus datos a A, provocando una colisión. En este caso, se dice que las estaciones B y C están ocultas respecto a sus transmisiones hacia A. Las estaciones ocultas pueden reducir la capacidad de la red debido a la posibilidad de colisión.

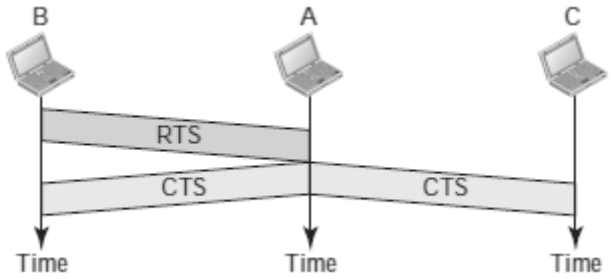


Figura 6.15. La trama CTS del método CSMA/CD evita las colisiones debidas a una estación oculta.

La solución a este problema es usar las tramas de control RTS y CTS de las que hablamos antes. La figura 6.15 muestra que el mensaje RTS enviado por B llega a A, pero no a C. Sin embargo, como A está dentro del rango de alcance de B y C, el mensaje CTS (que contiene la duración de la transmisión de datos de B a A) sí

que llega a C. Así es como la estación C sabe que una estación oculta está usando el canal, y se abstiene de transmitir durante la duración de la transmisión.

Problema de estación expuesta. Consideremos ahora la situación inversa a la anterior: El problema de la estación expuesta. En este caso, la estación se abstiene de usar el canal cuando, de hecho, el canal está disponible. En la figura 6.16 la estación A está transmitiendo a la estación B. La estación C también tiene unos datos para enviar a la estación D, que pueden transmitirse sin interferir sobre la transmisión de A a B. Sin embargo, la estación C está expuesta a la transmisión de A (esto es, detecta que A está transmitiendo) y se abstiene de enviar sus datos. En otras palabras, C es demasiado conservadora y desperdicia la capacidad del canal.

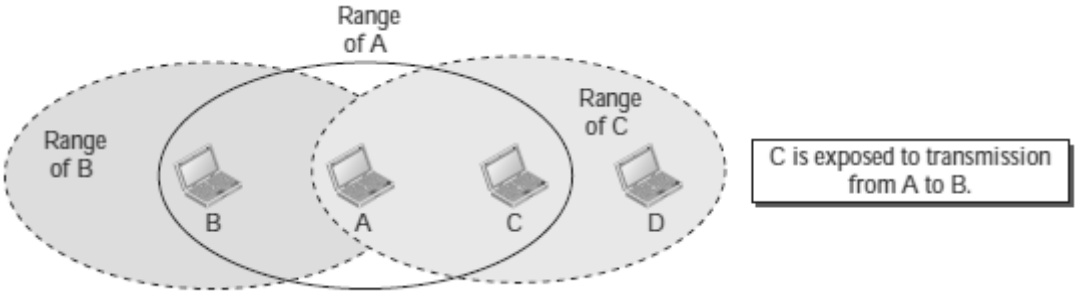


Figura 6.16. Problema de estación expuesta.

En este caso, y en contra de lo que cabría pensar, los mensajes RTS y CTS no son de ayuda. La figura 6.22 muestra la situación.

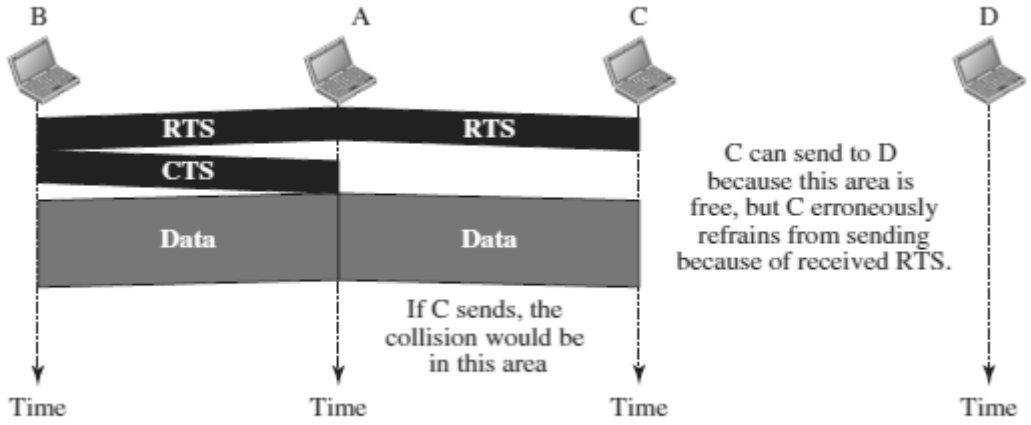


Figura 6.17. Uso de RTS y CTS en el problema de estación expuesta.

En la figura, la estación C escucha el RTS de la estación A, y se abstiene de transmitir, incluso aunque la comunicación entre C y D no puede causar una colisión en la zona de comunicación entre A y C. La estación C no puede saber que la transmisión de A no puede afectar a la zona de comunicación entre C y D.

PRÁCTICA K. PROGRAMACIÓN DE CSMA/CD CON PYTHON.

K.1. PLANTEAMIENTO DEL PROBLEMA.

En esta práctica vamos a escribir un programa en Python que implemente una versión simplificada del método de acceso CSMA/CD de Ethernet, ver sección 6.2 del capítulo previo. En este algoritmo simplificado, el ordenador que desee emitir procede de la siguiente forma:

1) Cuando la estación tiene un dato para transmitir, inicializa el contador K de número de intentos a cero, y a continuación, comprueba si el canal está libre. En nuestro caso, será el propio programa el que simule la ocupación del canal (con una probabilidad de un 25% de encontrarlo libre, y una probabilidad del 75% de hallarlo ocupado).
2) Si el canal está ocupado, la estación vuelve a comprobarlo otra vez, hasta encontrarlo libre. Si el canal está libre, la estación se pone a enviar el paquete de datos mientras se pone a comprobar si hay colisión. En nuestro caso, el propio programa simulará la situación de colisión, decidiendo aleatoriamente si se produce una colisión (con una probabilidad del 50%).
3) Si no hay colisión, la estación consigue enviar los datos con éxito.
4) En el caso de que ocurra una colisión, la estación envía una señal de interferencia (*jamming signal*) para alertar al resto de estaciones, e incrementa en 1 unidad el contador K. El número máximo de intentos será de tres. Si la estación ha intentado enviar los datos en tres ocasiones, abortará la transmisión. En caso contrario, la estación vuelve a comprobar el estado del canal.

A continuación mostramos dos ejemplos de ejecución del programa, uno con una transmisión exitosa, y otro con una transmisión fallida (abortada).

a) Transmisión exitosa:

```
La estación de origen tiene datos para enviar.
Intento número 0.
Escuchando el canal...
Canal ocupado.
Canal ocupado.
Canal ocupado.
Canal ocupado.
Canal ocupado.
Canal libre.
Transmisión en progreso...
Colisión detectada!
Enviando señal de interferencia...
Intento número 1.
Escuchando el canal...
Canal ocupado.
Canal ocupado.
Canal libre.
Transmisión en progreso...
Mensaje enviado correctamente.
```

b) Transmisión fallida:

```
La estación de origen tiene datos para enviar.
Intento número 0.
Escuchando el canal...
Canal ocupado.
```

```
Canal ocupado.
Canal ocupado.
Canal ocupado.
Canal ocupado.
Canal ocupado.
Canal libre.
Transmisión en progreso...
Colisión detectada!
Enviando señal de interferencia...
Intento número 1.
Escuchando el canal...
Canal libre.
Transmisión en progreso...
Colisión detectada!
Enviando señal de interferencia...
Intento número 2.
Escuchando el canal...
Canal ocupado.
Canal ocupado.
Canal ocupado.
Canal ocupado.
Transmisión en progreso...
Colisión detectada!
Enviando señal de interferencia...
Intento número 3. Número máximo de intentos alcanzado.
Transmisión abortada.
```

K.2. IMPORTAR MÓDULOS Y EMPEZAR EL PROGRAMA PRINCIPAL.

En este programa necesitaremos importar los módulos `random` (para obtener enteros aleatorios) y `sys` (para terminar el programa con la función `sys.exit()`).

En el programa principal, lo primero que debemos hacer es indicar que la estación emisora tiene datos para enviar (`La estación de origen tiene datos para enviar.`) e inicializar a cero el valor del contador K de número de intentos. Por el momento, dejaremos así el programa principal. Lo completaremos más adelante.

K.3. FUNCIÓN CSMA().

A continuación vamos a escribir la función `csma()` que implementa el procedimiento de escucha del canal (CSMA). Lo primero que hace la función es indicar que la estación va a comprobar el estado del canal (`Escuchando canal...`). A continuación, la función obtiene un número aleatorio entre 1 y 100. (Con este número aleatorio simularemos la probabilidad de encontrarlo libre u ocupado). Mientras el número aleatorio sea mayor o igual que 25 (esto es, el 75% de las veces), la función indicará que el canal está ocupado (`Canal ocupado.`), y volverá a obtener un nuevo número aleatorio. En el momento en el que el número aleatorio sea menor que 25 (el 25% de las veces), la función indicará que el canal se encuentra libre (`Canal libre.`), y que la estación inicia la transmisión (`Transmisión en progreso...`).

```
def csma ():
    # Rellena aquí el código de la función
```

K.4. FUNCIÓN CD().

Ahora vamos a escribir la función `cd()` que simula la técnica de detección de colisión. Esta función recibirá como argumento el número actual de intentos (K), y devolverá el número de intentos actualizado.

Para empezar, la función obtiene un número aleatorio entre 0 y 1. Si el número aleatorio obtenido es 0, asumiremos que no se ha producido colisión, la estación transmite el mensaje con éxito (`Mensaje enviado correctamente.`), y el programa termina.

Por el contrario, si el número aleatorio es 1 supondremos que ha ocurrido una colisión. En ese caso, la estación informa de este hecho (`Colisión detectada!`), indica que pasa a enviar la señal de interferencia (`Enviando señal de interferencia...`), e incrementa el número de intentos K en una unidad.

Por último, la función termina devolviendo el valor actualizado del número de intentos.

```
def cd (K):
    # Rellena aquí el código de la función
```

K.5. COMPLETAR EL PROGRAMA PRINCIPAL.

Para terminar, vamos a completar el programa principal.

Tras haber inicializado el valor del contador K de número de intentos, el programa principal entra en un bucle que se repite mientras el contador de intentos sea menor que tres. A cada pasada del bucle, el programa indica el número del intento actual (`Intento número ?.`), luego llama a la función `csma()` para acceder al medio , y después llama a la función `cd()`. Recordar que la función `cd()` devuelve el número de intentos actualizado. Cuando el número de intentos se hace igual a tres, la función sale del bucle, indica que se han alcanzado el número máximo de intentos (`Intento número 3. Número máximo de intentos alcanzado.`), y aborta la transmisión (`Transmisión abortada.`).

K.6. MODIFICACIONES DEL PROGRAMA.

1) Tal y como está, el programa funciona, pero de una forma muy predecible: La estación consigue enviar el mensaje en el primer intento, en el segundo, en el tercero, o aborta la transmisión. Para hacerlo un poco más impredecible, modifica el programa para que el usuario pueda elegir a priori el número máximo de intentos, de forma que no sea siempre tres.

2) Pero si permitimos que el usuario pueda elegir un número máximo de intentos mayor que tres, debemos hacer que las colisiones sean más probables para que el programa pueda llegar a ese número máximo de intentos. Hasta ahora, y para un número máximo de tres intentos, la probabilidad de colisión es del 50%. Modifica el programa para hacer que la probabilidad de colisión crezca de forma proporcional al número de intentos, pero sin superar nuca el 90%. Si x es el número máximo de intentos; un ejemplo de función para obtener la probabilidad de colisión sería:

$$P(x) = \begin{cases} 50 & ,x = 2,3 \\ 50 + 4x & ,3 < x \leq 10 \\ 90 & ,x > 10 \end{cases}$$

   Notar que el número máximo de intentos x solo puede tomar valores enteros positivos, excluyendo el 0 y el 1.

PRÁCTICA L. MONITORIZACIÓN DE REDES INALÁMBRICAS (Y ALGO DE EXCEL).

L.1. PROGRAMA VISTUMBLER.

Vistumbler (https://www.vistumbler.net/) es un escáner de redes inalámbricas diseñado para visualizar los puntos de acceso (APs) más cercanos. Vistumbler es capaz de recopilar gran cantidad de datos del interfaz radio, como son el nombre de las redes detectadas (SSID), canal en el que transmiten ($1 - 13$), tipo de encriptación, dirección MAC del punto de acceso, nivel de potencia de la señal recibida (RSSI), calidad de la señal recibida (Signal), etc.

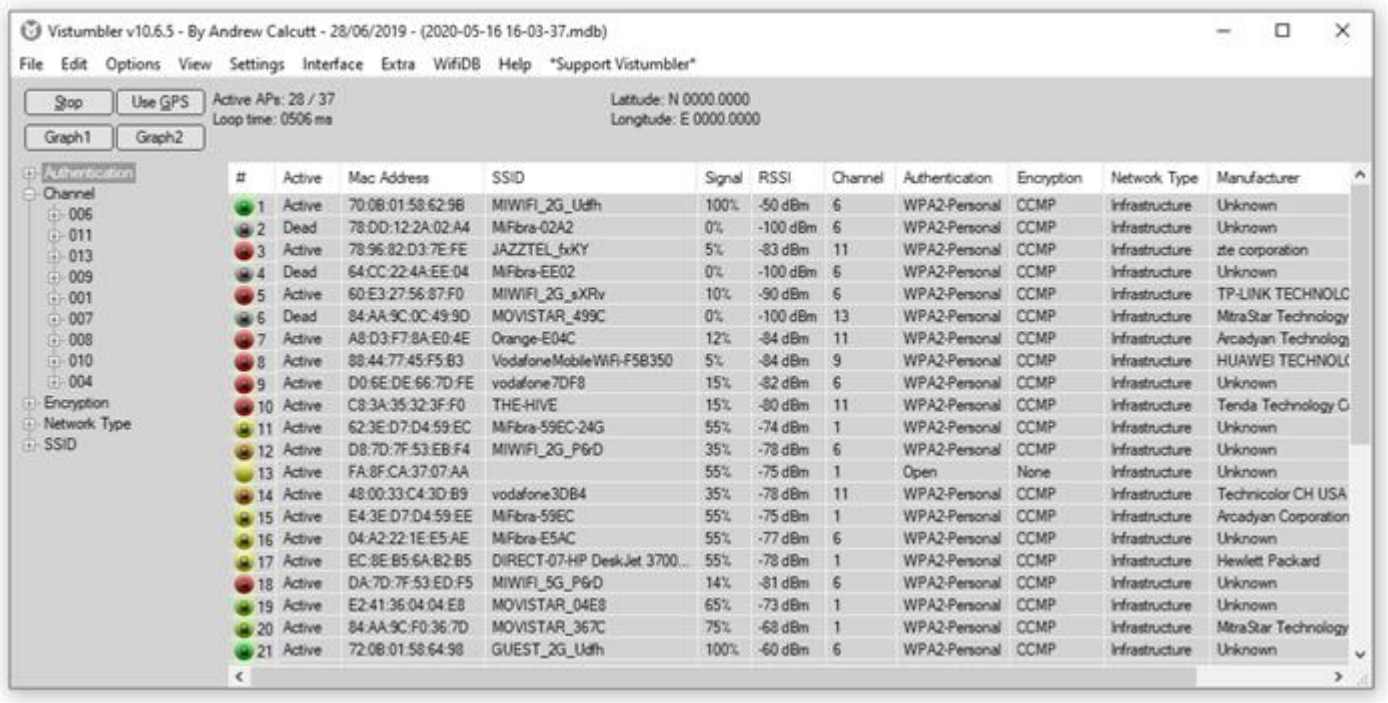


Figura L.1. Pantalla principal de Vistumbler.

L.2. NIVEL DE SEÑAL (RSSI) Y CALIDAD DE SEÑAL (SIGNAL).

La figura L.1 muestra la pantalla principal de Vistumbler en modo de búsqueda de APs. En la primera columna vemos una serie de círculos de color que muestran el estado de la conexión a la red en cuestión, en términos de nivel de señal recibida (columna "RSSI") y calidad de la señal recibida (columna "Signal"). El significado es el siguiente:

- Verde oscuro: Muy buena conexión.
- Verde claro: Buena conexión
- Amarillo: Conexión regular.
- Naranja: Mala conexión.
- Rojo: Muy mala conexión.
- Gris: Sin conexión.

El **RSSI** (Received Signal Strength Indicator) es un parámetro que mide el nivel de potencia de la señal recibida, en relación a una potencia de referencia de $1\,mW$. Aunque teóricamente podría tomar valores positivos, la escala para el RSSI suele variar entre los $0\,dBm$ y los $-100\,dBm$. Esto significa que cuanto más negativo es el valor del RSSI, más débil es la señal recibida.

Ahora bien, el RSSI solo indica la *intensidad* de la señal recibida, pero no su *calidad*. En toda comunicación hay señales indeseadas (denominadas conjuntamente como *ruido*) que degradan la calidad de la señal útil. El parámetro **Signal** es un indicador que reporta la calidad del enlace. Este parámetro toma valores entre 0% y 100%, donde un valor del 100% indica la máxima calidad del enlace.

Conjuntamente, los parámetros RSSI y Signal determinan el estado de la conexión (esto es, el color del icono circular asociado a la conexión). Por ejemplo, para dos redes con el mismo nivel de señal recibida (RSSI), la conexión será mejor en aquella con una mayor calidad del enlace. Incluso una conexión podría ser mejor que otra con un valor más bajo de RSSI, si la calidad de la conexión es superior.

La figura L.4 muestra un gráfico con el valor a corto plazo del RSSI recibido por mi portátil para la red WiFi a la que estaba conectado (MIWIFI_2G_Udfh). En este caso, el portátil estaba inmóvil a tan solo unos pocos metros del punto de acceso. Estos gráficos de nivel de señal nos permiten determinar las zonas de cobertura de la red WiFi que estemos monitorizando. Si nos desplazamos con un portátil por la zona a la que se desea dar servicio con una red inalámbrica, podemos identificar las zonas con mala cobertura o sin cobertura.

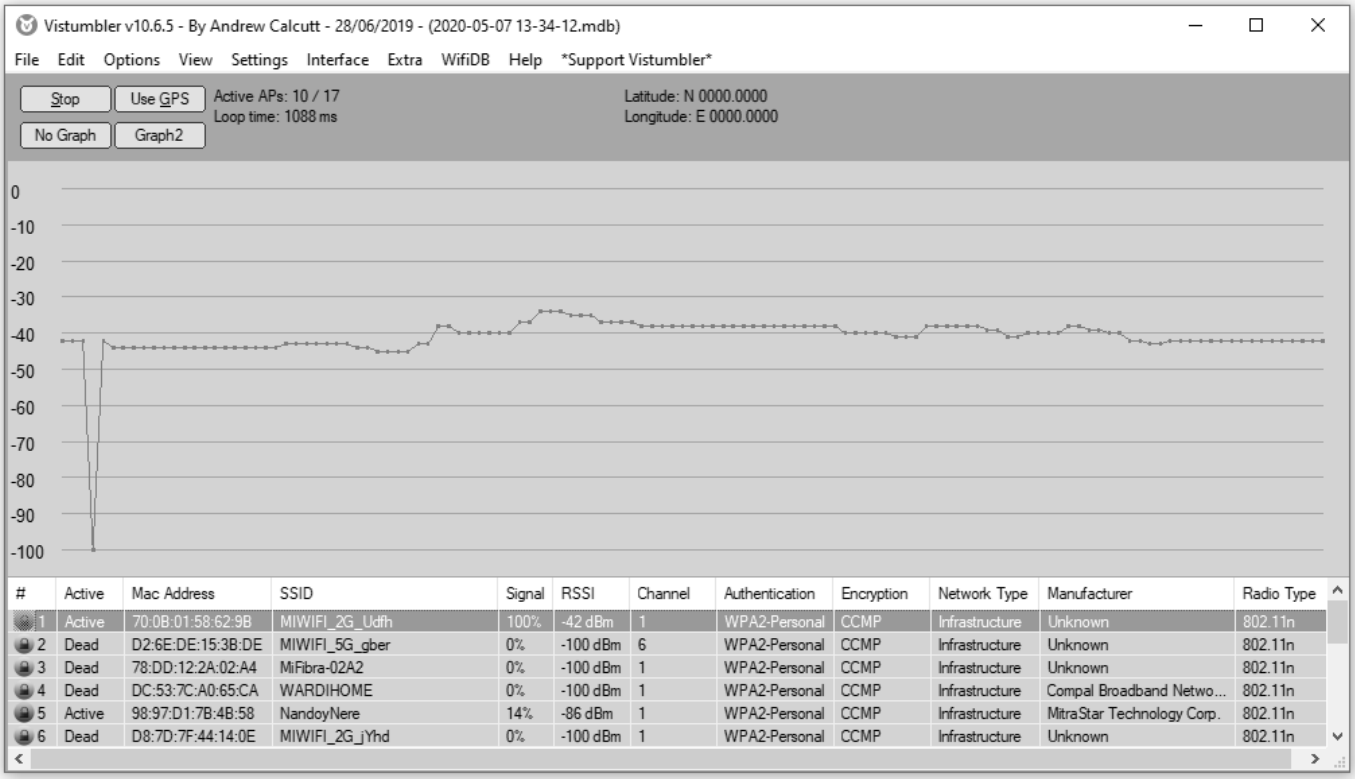


Figura L.4. Gráfica de RSSI en Vistumbler.

Por supuesto, cuanto más lejos nos movamos del AP, más débil será la señal recibida. Las señales inalámbricas también son muy sensibles a las interferencias, a la existencia de obstáculos como muros y paredes, y a la presencia de agua (tuberías, baños, etc.).

La figura L.5 es un gráfico con el valor a largo plazo del RSSI mientras me desplazaba por mi casa con el portátil, y nos sirve como ejemplo de un sencillo estudio de cobertura. La zona (1) muestra el nivel de señal en el salón, donde se localiza el AP. Como vemos, el RSSI es de aproximadamente $-40\ dBm$. La zona (2) es la terraza de verano junto al salón, donde hay una bajada de $20\ dB$ en el nivel de señal, probablemente debido al grosor de los tabiques exteriores. La zona (3) es la cocina, junto a la terraza y el salón. El nivel de señal es $15\ dB$ inferior al nivel del salón, a pesar de que cocina y salón solo están separados por un tabique (probablemente grueso). La zona (4) muestra el dormitorio 1 junto al salón, y muestra un nivel de señal muy similar. Este dormitorio solo está separado del salón por un tabique (probablemente delgado). La zona (5) es el dormitorio 2 junto al dormitorio 1. La zona (6) es el dormitorio 3 (el más alejado del salón), y

la zona (7) es el baño 2 junto al dormitorio 3. Como vemos, los mayores problemas de cobertura se localizan en las zonas más alejadas, en las que la señal tiene que atravesar muchos tabiques para llegar. Especialmente dramática es la situación en el baño 2, donde el RSSI apenas si supera los $-80\ dBm$. El valor umbral que permite la conexión de un equipo a una red (la llamada *sensibilidad* del equipo) suele estar en el entorno de los $-80\ dBm$.

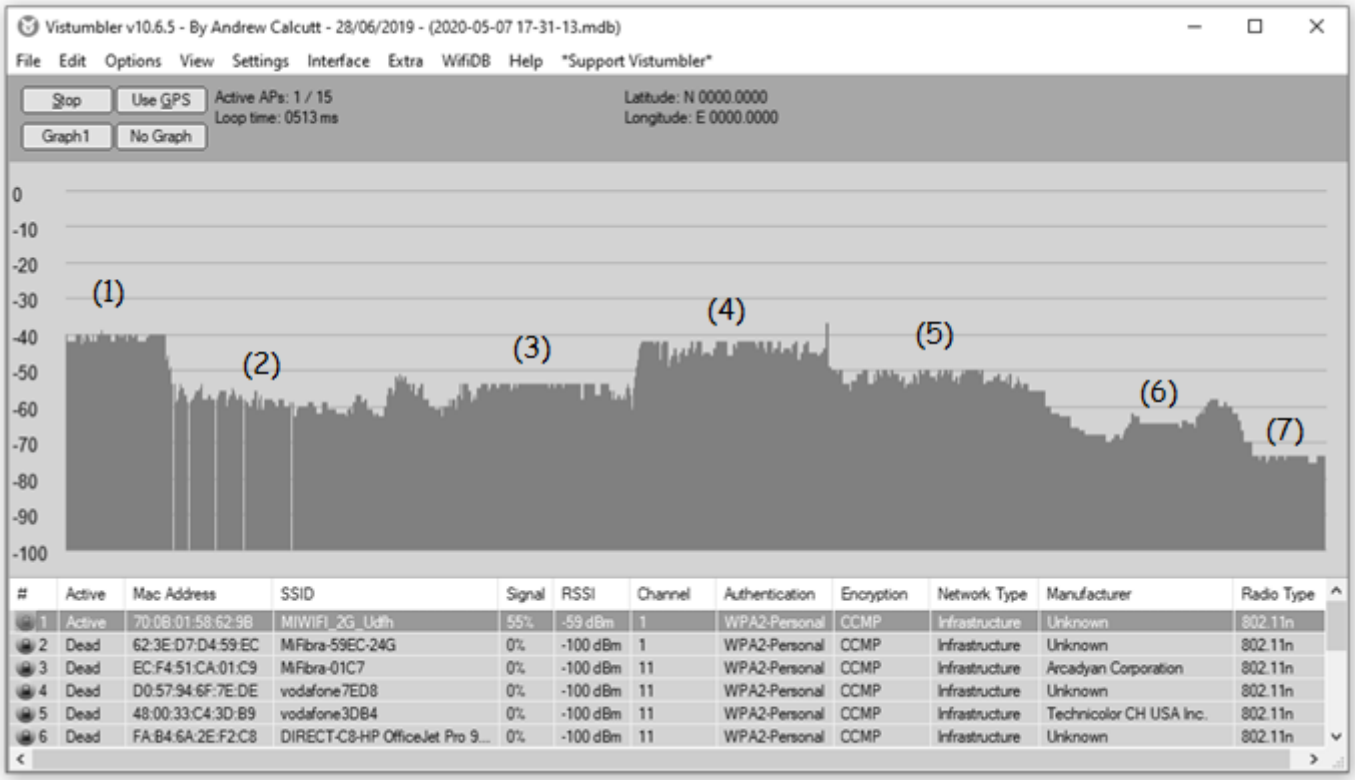


Figura L.5. Estudio de cobertura.

L.3. MANEJO BÁSICO DE VISTUMBLER.

1) Instala Vistumbler en tu portátil (https://www.vistumbler.net/downloads.html).

2) Abre el programa Vistumbler recién instalado, acude al menú "Interface", y asegúrate de tener seleccionado el interfaz WiFi de tu portátil.

3) A continuación, acude al menú "Settings" (Ajustes), y en la lista desplegable selecciona la opción "Misc Setting" para cambiar algunos ajustes de Vistumbler. En la ventana de ajustes, en la pestaña "Misc", bajamos el valor del parámetro "Refresh loop time" a $500\ ms$ para que el programa tome registros más frecuentes de los niveles de señal recibidos. Después, en la pestaña "Columns" elegimos qué datos vamos a mostrar en la ventana principal de Vistumbler. Asegúrate de habilitar los datos marcados en la figura L.6. A continuación pulsa en el botón "Apply", y en el botón "Ok" para salir de los ajustes.

4) De vuelta a la ventana principal de Vistumbler, pulsa en el botón "Scan APs" para empezar a monitorizar los puntos de acceso disponibles. La cuadrícula central empezará a llenarse de datos. Si tienes WiFi en casa, probablemente será tu red la que aparezca en primera posición, y con el mejor estado para la conexión. Y tengas o no tengas WiFi en casa, Vistumbler también mostrará todos los puntos de acceso que el adaptador de red es capaz de detectar en tu vecindario.

5) Para cada red detectada, Vistumbler muestra algunos datos como el SSID (nombre de la red), dirección MAC del punto de acceso, RSSI (intensidad de la señal recibida), Signal (Calidad del enlace), Channel (canal en el que opera la red), método de autenticación (WEP, WPA, abierta), fabricante, etc.

6) A continuación, y si tienes WiFi en casa, selecciona tu red. (De no tener WiFi en casa, selecciona el punto de acceso que recibas con una mayor intensidad y calidad de señal). Después pulsa en el botón "Graph1" para obtener un gráfico del nivel de señal recibida a corto plazo, como el mostrado en la figura L.4. Para volver a la lista de APs, vuelve a pulsar en el botón previo, que ahora muestra el texto "No Graph". Si ahora pulsas en el botón "Graph2" obtendrás un gráfico del nivel de señal a largo plazo como el de la figura L.5. Notar que ambos gráficos van añadiendo una nueva muestra cada vez que transcurre el tiempo fijado en el ajuste "Refresh loop time". Por ejemplo, si has fijado un valor de $500\,ms$, los gráficos añaden una nueva muestra cada medio segundo.

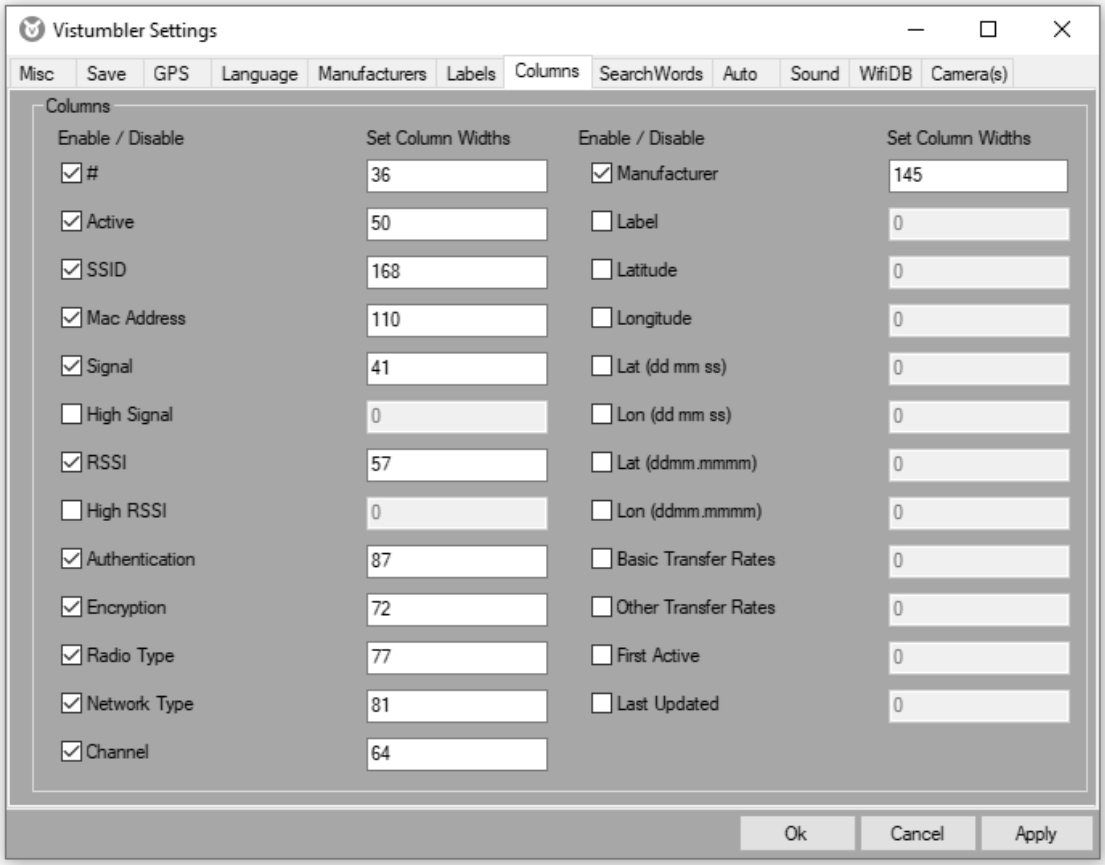


Figura L.6. Datos a mostrar.

7) Ahora echa un vistazo a la zona izquierda del programa, donde verás usa serie de listas desplegables llamadas "Authentication", "Channel", "Encryption", "Network Type", y "SSID". Vamos a explicar para qué sirven tomando como ejemplo la lista "Channel".

   En la banda de $2{,}4\ GHz$ (protocolos 802.11 b/g/n/ax), el estándar define 14 canales, de los cuales solo se usan del 1 al 13. El punto de acceso debe configurarse para transmitir en uno de estos canales. Por supuesto, si un canal está muy ocupado por las transmisiones de muchos puntos de acceso, las señales interferirán unas con otras, y la calidad de la recepción se verá afectada. Deberíamos configurar nuestro punto de acceso en un canal que esté lo menos ocupado posible.

   Para consultar la ocupación de los canales, despliega la lista "Channel". En esta lista aparecen todos los canales que están siendo usados por los puntos de acceso detectados por Vistumbler, y en qué canal está transmitiendo cada punto de acceso (ver figura L.7). Como vemos, esta lista permite consultar de un solo vistazo cómo se reparten las redes más cercanas entre los distintos canales. Tu red debería configurarse para usar uno de los canales menos ocupados.

   Además de la lista desplegable de canales, Vistumbler también proporciona listas para consultar el tipo de autenticación usado por cada red (Abierta, WEP, o WPA), el tipo de encriptado de cada red, la clase de red (ad-hoc o infraestructura), etc. Por ejemplo, podrías usar esta información para averiguar qué

redes WiFi de tu vecindario están abiertas (esto es, son accesibles sin clave), o están protegidas mediante clave WEP (una clave muy fácil de romper).

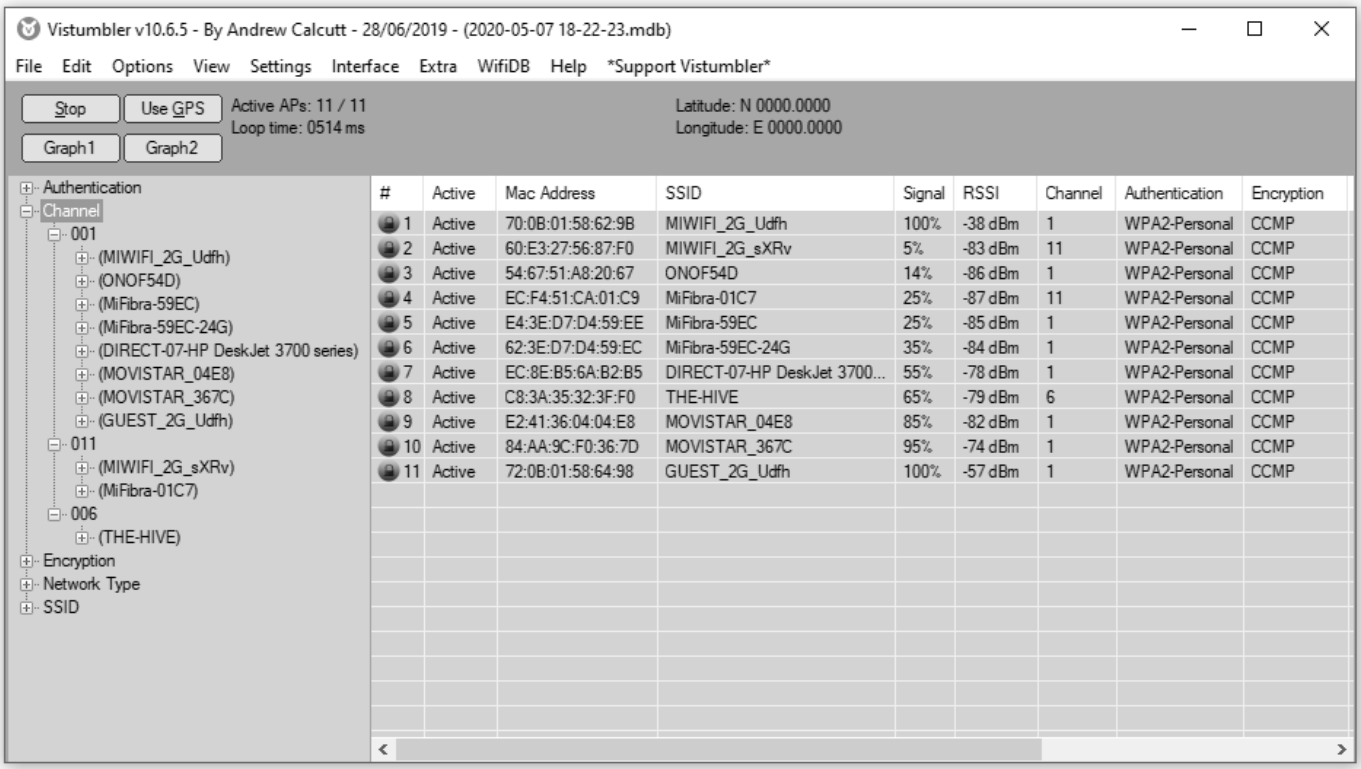


Figura L.7. Canales ocupados por las distintas redes detectadas.

L.4. EXPORTAR Y PROCESAR LA INFORMACIÓN DE VISTUMBLER.

EXPORTAR LOS DATOS A EXCEL.

A continuación vamos a mostrar cómo podemos exportar los datos capturados por Vistumbler, para poder procesarlos después con una hoja de cálculo tipo Excel. Para ello, seguimos los siguientes pasos:

a) Tras haber pulsado el botón "Scan APs" y haber estado un tiempo capturando datos, vamos a exportarlos. Para exportar los datos de todos los APs detectados, acudimos al menú "File" → "Export" → "Export to CSV", y elegimos la opción "All APs" (o en su caso, solo los datos de los APs en los que estemos interesados). En la ventana emergente, seleccionamos la carpeta donde queremos guardar el archivo resultante, activamos el botón "Detailed Comma Delimited file", y pulsamos en "Ok" (ver figura L.8). De esta forma, habremos exportado los datos a un archivo de texto CSV (Comma Separated Values), en el que los datos están separados por comas.

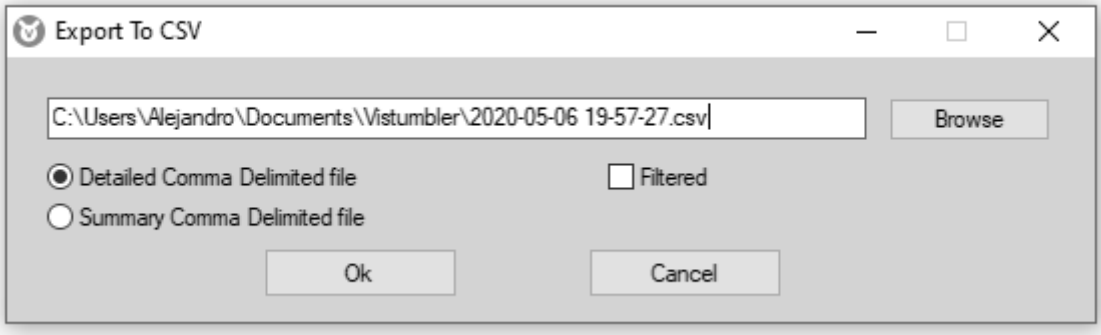


Figura L.8. Exportar datos a un archivo CSV.

b) A continuación, acudimos a la ubicación del archivo CSV guardado, y lo abrimos con Excel. El archivo debería tener una apariencia similar a la mostrada en la figura L.9. Tal y como está, este archivo no puede usarse en Excel, porque cada celda incluye muchos datos separados por comas. Por consiguiente, necesitamos separar los datos en celdas independientes para poder procesarlos. Para ello, y ya dentro de Excel, acudimos al menú "Datos", y en el bloque "Obtener datos externos", pulsamos en el botón "Desde texto", ver figura L.10.

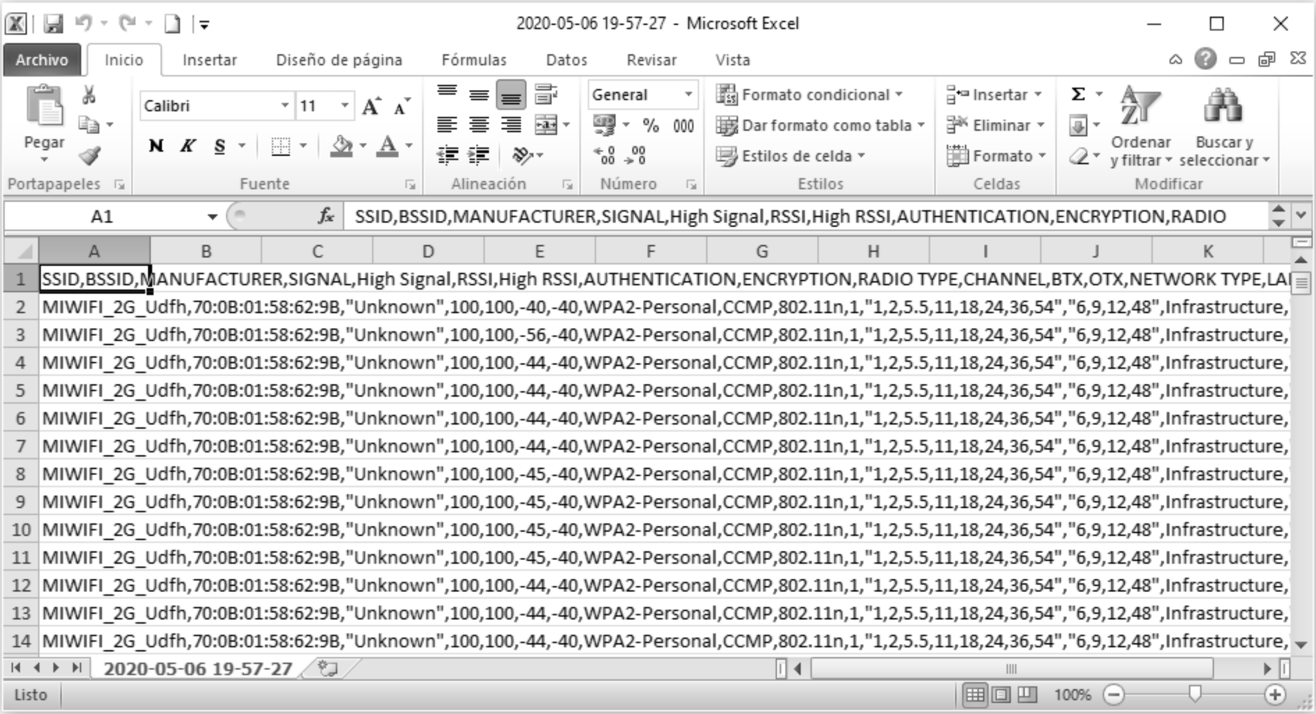


Figura L.9. Archivo CSV abierto directamente en Excel.

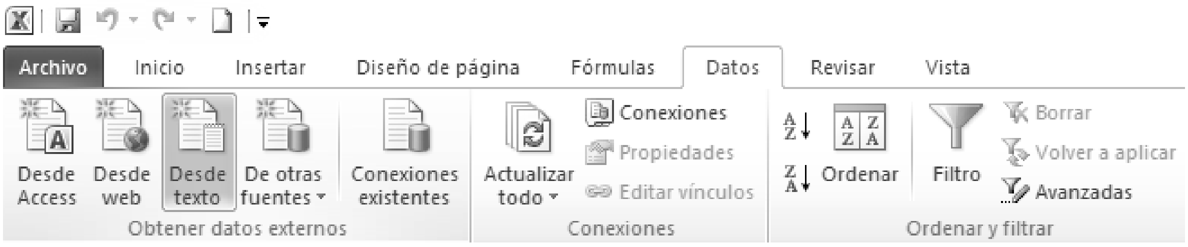


Figura L.10. Importar datos externos desde Excel.

c) En la ventana emergente, acudimos a la ubicación del archivo CSV y pulsamos en el botón "Importar". Se abrirá un asistente para importar archivos de texto a Excel. El proceso consta de tres pasos:

- En el paso 1 seleccionamos la opción "Delimitados", ver figura L.11. Pulsamos en "Siguiente".

- En el paso 2 seleccionamos "Coma" como separador. Veremos que las comas de los datos desaparecen, y que cada dato se almacena en una celda, como muestra la figura L.12. Pulsamos en "Siguiente".

- En el paso 3, el asistente nos permite definir el tipo de los datos almacenados en cada columna. Podríamos ir columna a columna indicando si los datos son texto, números, etc. Aquí simplemente aplicaremos el formato "General" a los datos de todas las columnas (ver figura L.13). Pulsamos en "Finalizar".

Para terminar, indicamos que deseamos situar los datos en una nueva hoja de cálculo y pulsamos "Aceptar", ver figura L.14. Si todo ha ido bien, deberíamos terminar con un archivo Excel similar al

mostrado en la figura L.15, el cual está formateado para poder trabajar con él. Para conservar este archivo, lo guardamos con una extensión .xslx (libro de Excel).

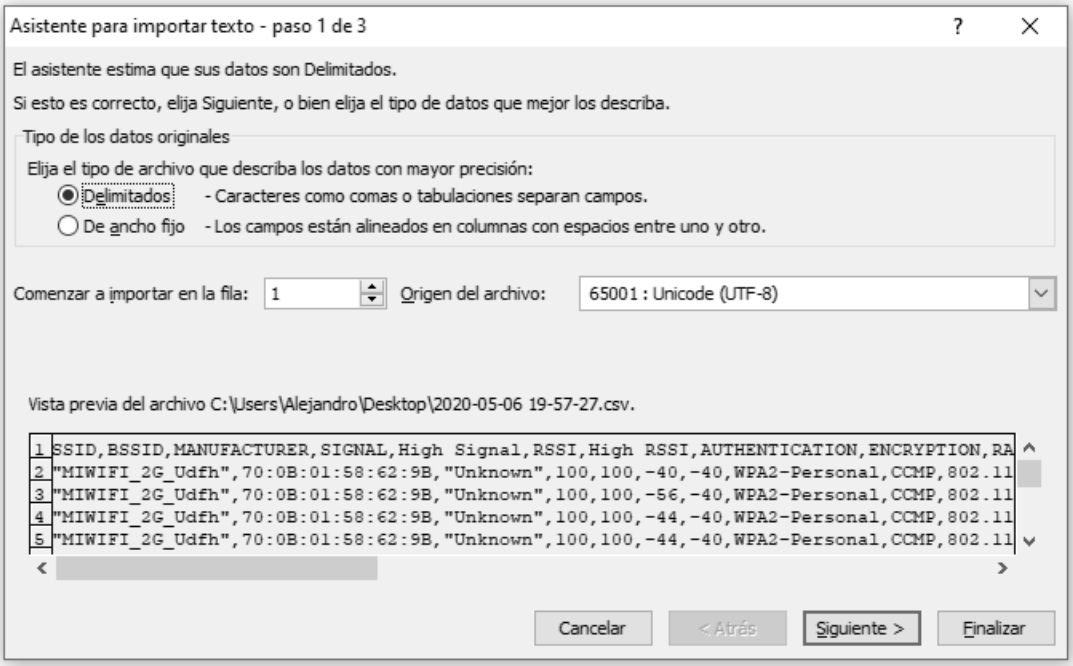


Figura L.11. Asistente para importar archivos CSV (1).

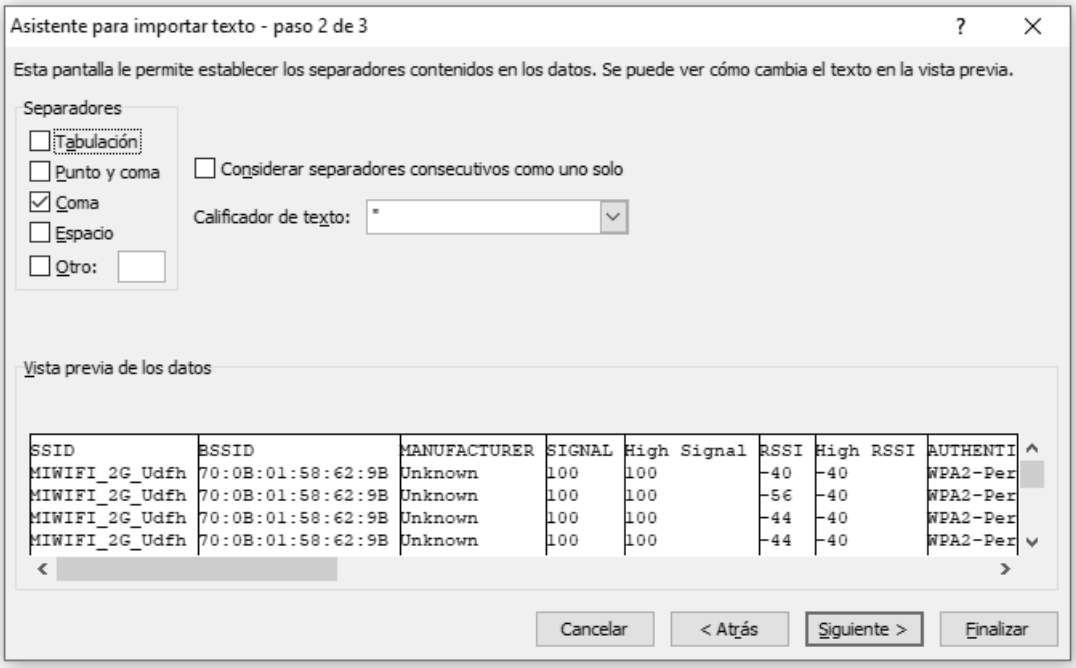


Figura L.12. Asistente para importar archivos CSV (2).

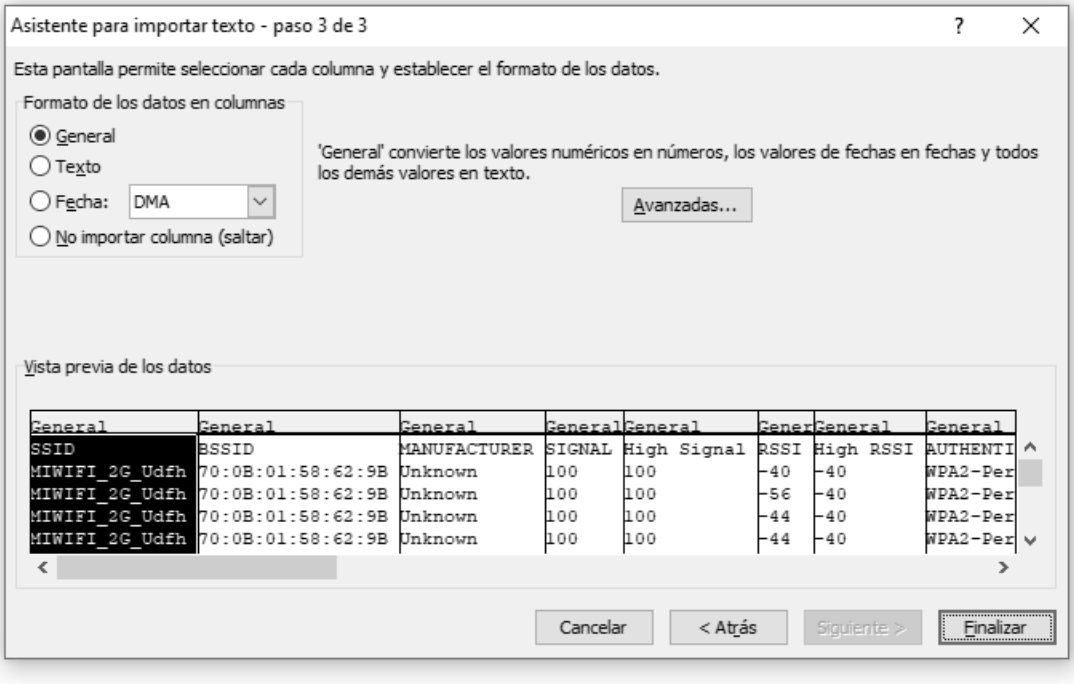


Figura L.13. Asistente para importar archivos CSV (3).

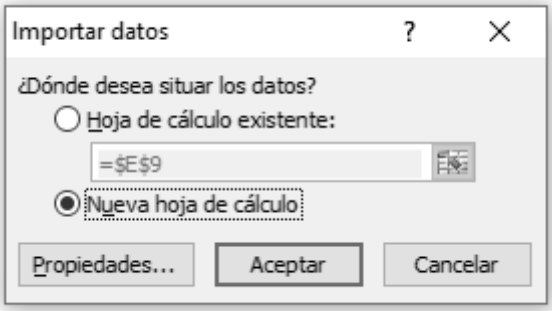


Figura L.14. Finalizar la importación de un archivo CSV.

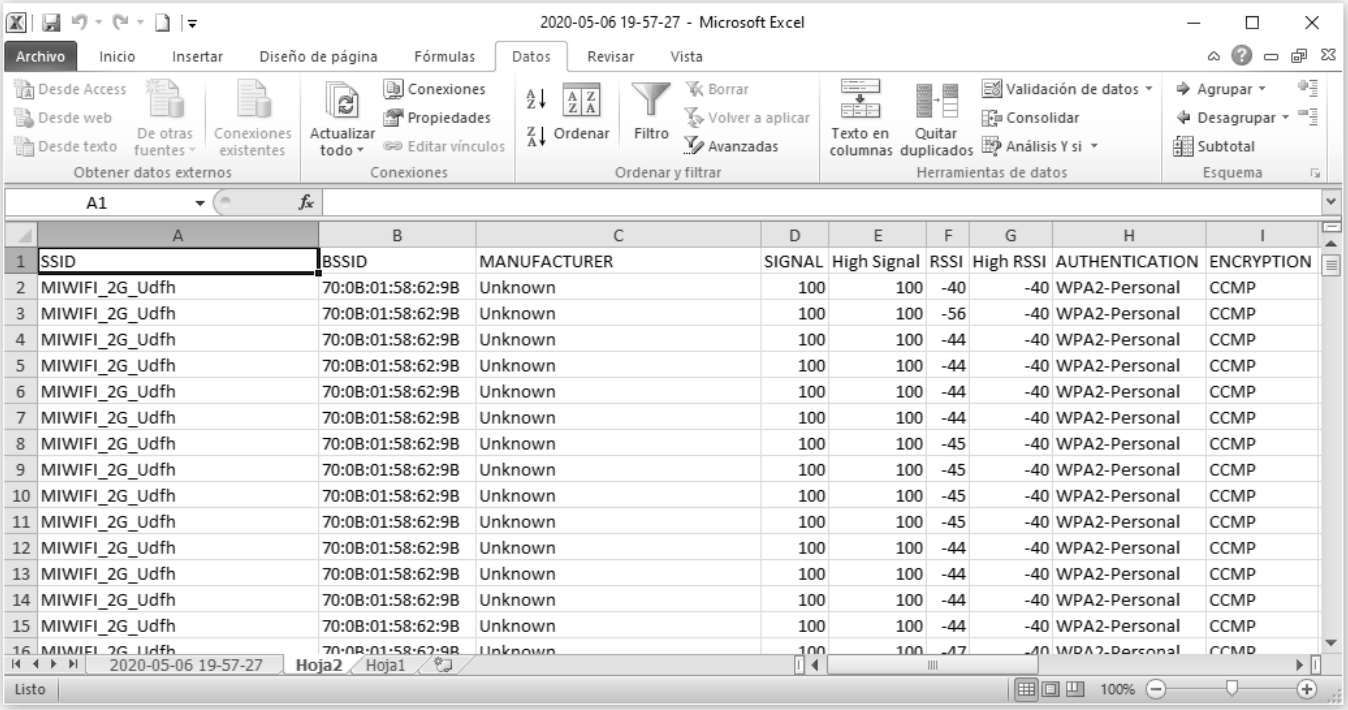
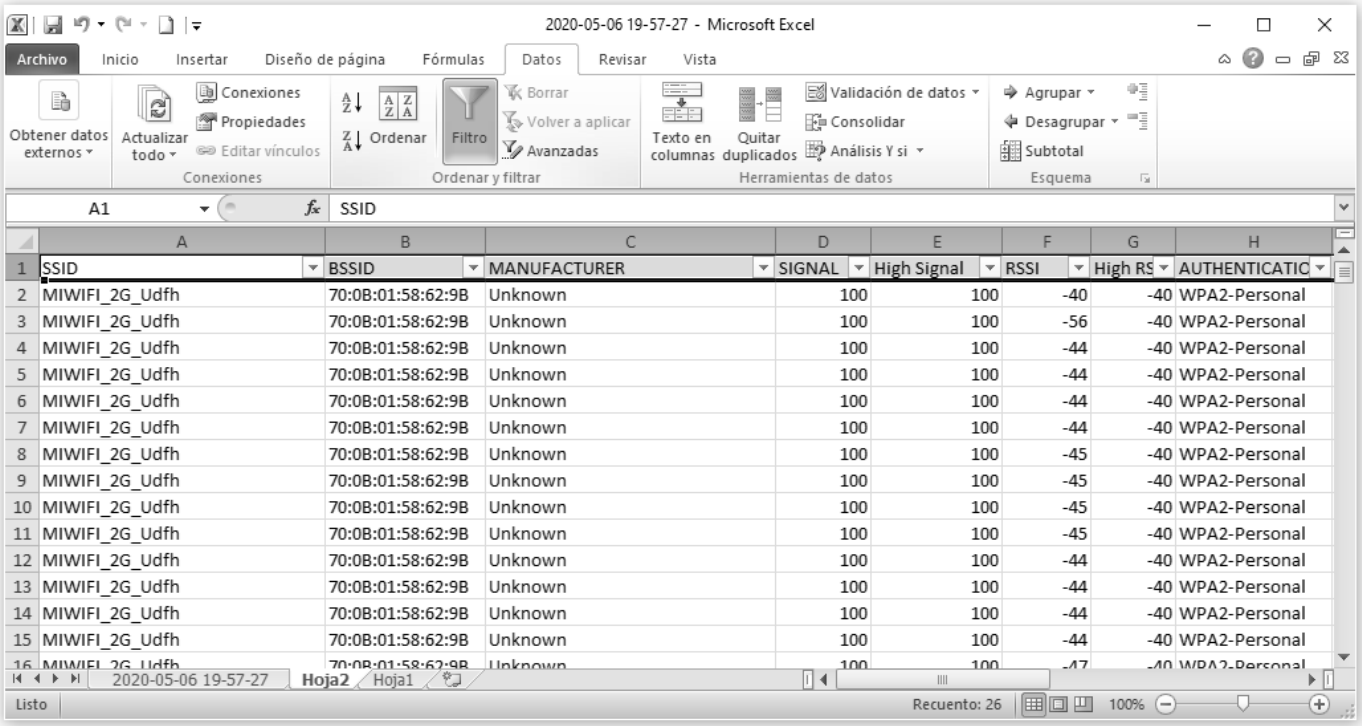


Figura L.15. Datos importados en Excel.

PROCESAR LOS DATOS EN EXCEL.

Filtros.

Una vez tenemos archivo Excel con los datos debidamente formateados, podemos comenzar a procesarlos. Para empezar, podemos aplicar un **filtro** a los datos. Para hacerlo, pinchamos en el "1" junto a la primera línea de la tabla Excel (donde están las etiquetas de los datos capturados: SSID, BSSID, Manufacturer, SIGNAL, RSSI, etc.) para seleccionarla, y a continuación, acudimos al menú "Datos" y seleccionamos "Filtro", ver figura L.16.



	A	B	C	D	E	F	G	H
1	SSID	BSSID	MANUFACTURER	SIGNAL	High Signal	RSSI	High RS	AUTHENTICATIC
2	MIWIFI_2G_Udfh	70:0B:01:58:62:9B	Unknown	100	100	-40	-40	WPA2-Personal
3	MIWIFI_2G_Udfh	70:0B:01:58:62:9B	Unknown	100	100	-56	-40	WPA2-Personal
4	MIWIFI_2G_Udfh	70:0B:01:58:62:9B	Unknown	100	100	-44	-40	WPA2-Personal
5	MIWIFI_2G_Udfh	70:0B:01:58:62:9B	Unknown	100	100	-44	-40	WPA2-Personal
6	MIWIFI_2G_Udfh	70:0B:01:58:62:9B	Unknown	100	100	-44	-40	WPA2-Personal
7	MIWIFI_2G_Udfh	70:0B:01:58:62:9B	Unknown	100	100	-44	-40	WPA2-Personal
8	MIWIFI_2G_Udfh	70:0B:01:58:62:9B	Unknown	100	100	-45	-40	WPA2-Personal
9	MIWIFI_2G_Udfh	70:0B:01:58:62:9B	Unknown	100	100	-45	-40	WPA2-Personal
10	MIWIFI_2G_Udfh	70:0B:01:58:62:9B	Unknown	100	100	-45	-40	WPA2-Personal
11	MIWIFI_2G_Udfh	70:0B:01:58:62:9B	Unknown	100	100	-45	-40	WPA2-Personal
12	MIWIFI_2G_Udfh	70:0B:01:58:62:9B	Unknown	100	100	-44	-40	WPA2-Personal
13	MIWIFI_2G_Udfh	70:0B:01:58:62:9B	Unknown	100	100	-44	-40	WPA2-Personal
14	MIWIFI_2G_Udfh	70:0B:01:58:62:9B	Unknown	100	100	-44	-40	WPA2-Personal
15	MIWIFI_2G_Udfh	70:0B:01:58:62:9B	Unknown	100	100	-44	-40	WPA2-Personal
16	MIWIFI_2G_Udfh	70:0B:01:58:62:9B	Unknown	100	100	-47	-40	WPA2-Personal

Figura L.16. Aplicar un filtro en Excel.

Una vez aplicado el filtro, podemos mostrar únicamente los datos que nos interesen. Por ejemplo, si queremos visualizar únicamente los datos correspondientes a la red "MIWIFI_2G_Udfh", pinchamos sobre la flecha junto a la etiqueta SSID, y en la ventana de selección, desactivamos la opción "(Seleccionar todo)", y activamos la opción "MIWIFI_2G_Udfh" (ver figura L.17). Cuando hay un filtro aplicado a una de las columnas de una tabla Excel, la flechita del filtro de esa columna incluye el símbolo de un embudo (ver figura L.18).

Otro ejemplo sería aplicar un filtro para mostrar únicamente los datos de aquellas redes que operen en los canales entre el 3 y el 6, como muestra la figura L.19. Incluso podemos aplicar varios filtros simultáneamente, como por ejemplo mostrar los datos correspondientes a la red "MIWIFI_2G_Udfh", pero únicamente aquellos en los que el valor del RSSI es mayor que -50.

Para eliminar los filtros aplicados, basta con volver a seleccionar la primera línea de la tabla Excel, y deseleccionar la opción "Filtro" del menú "Datos".

Gráficos dinámicos.

Todos sabemos hacer gráficos en Excel, pero una opción muy interesante es crear **gráficos dinámicos**. Para crear un gráfico dinámico basta con acudir al menú "Insertar" → "Tabla dinámica", y elegir la opción "Gráfico dinámico".

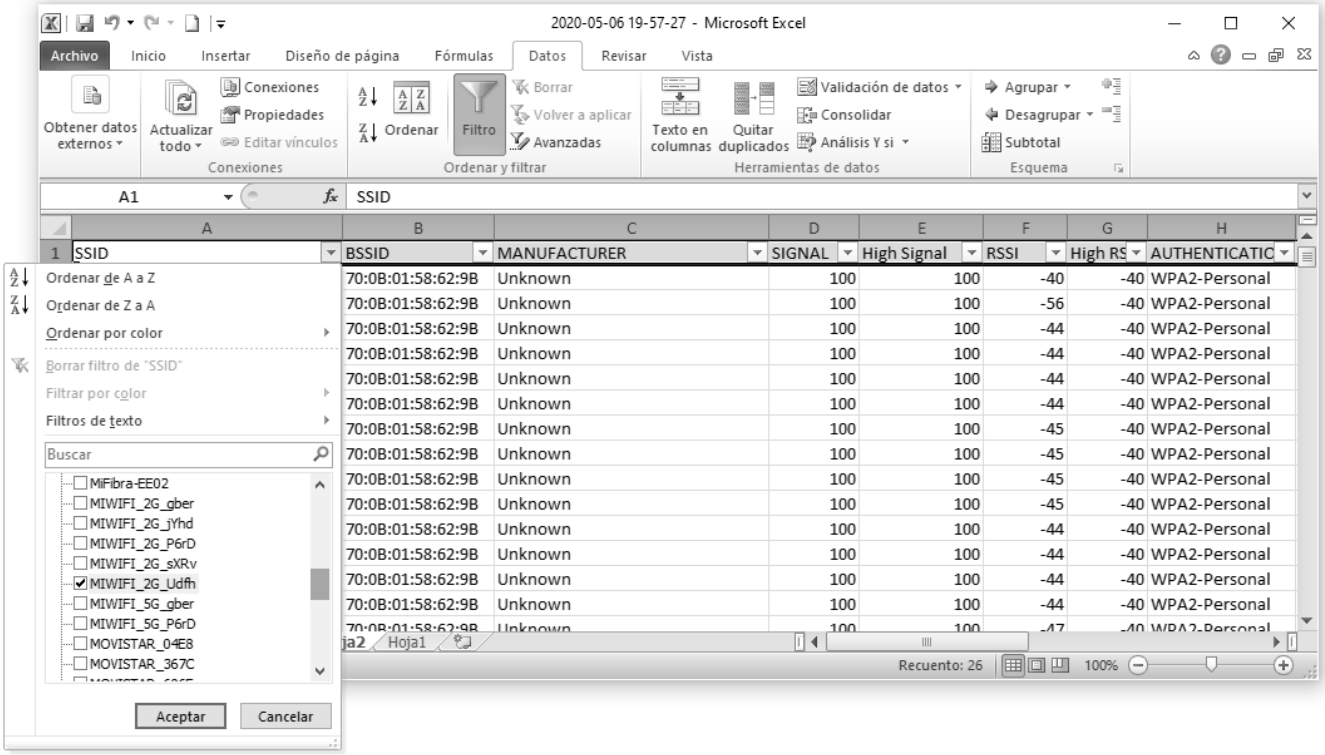


Figura L.17. Filtrado de los datos para la red con SSID "MIWIFI_2G_Udfh".

	A	B	C	D
1	SSID	BSSID	MANUFACTURER	SIGNAL
2	MIWIFI_2G_Udfh	70:0B:01:58:62:9B	Unknown	100
3	MIWIFI_2G_Udfh	70:0B:01:58:62:9B	Unknown	100
4	MIWIFI_2G_Udfh	70:0B:01:58:62:9B	Unknown	100
5	MIWIFI_2G_Udfh	70:0B:01:58:62:9B	Unknown	100

Figura L.18. La columna SSID tiene un filtro definido.

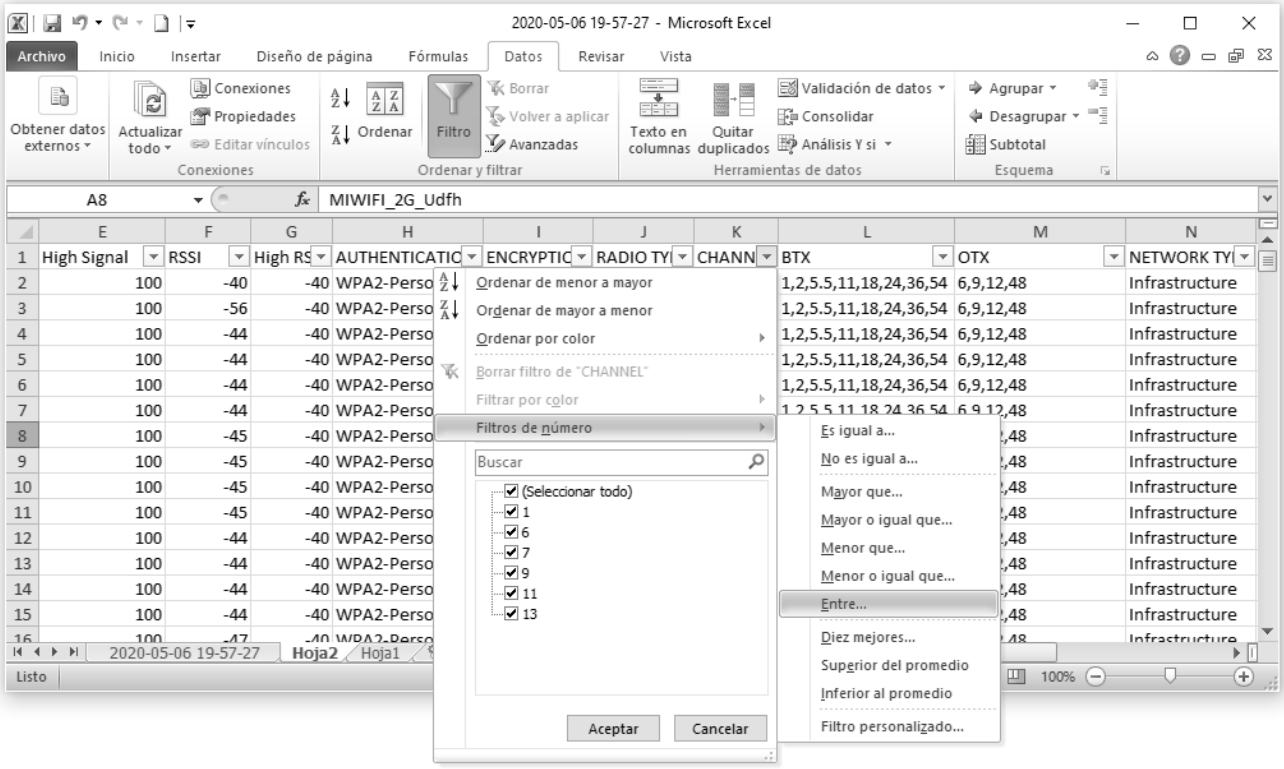


Figura L.19. Filtro numérico sobre la columna "Channel".

Se abrirá una ventana para elegir con qué datos queremos crear el gráfico dinámico (por defecto, se construirá con todos los datos de la hoja actual), y dónde colocarlo (por defecto, se colocará en una nueva hoja). Nosotros seleccionamos las opciones por defecto (ver figura L.20), y pulsamos en "Aceptar".

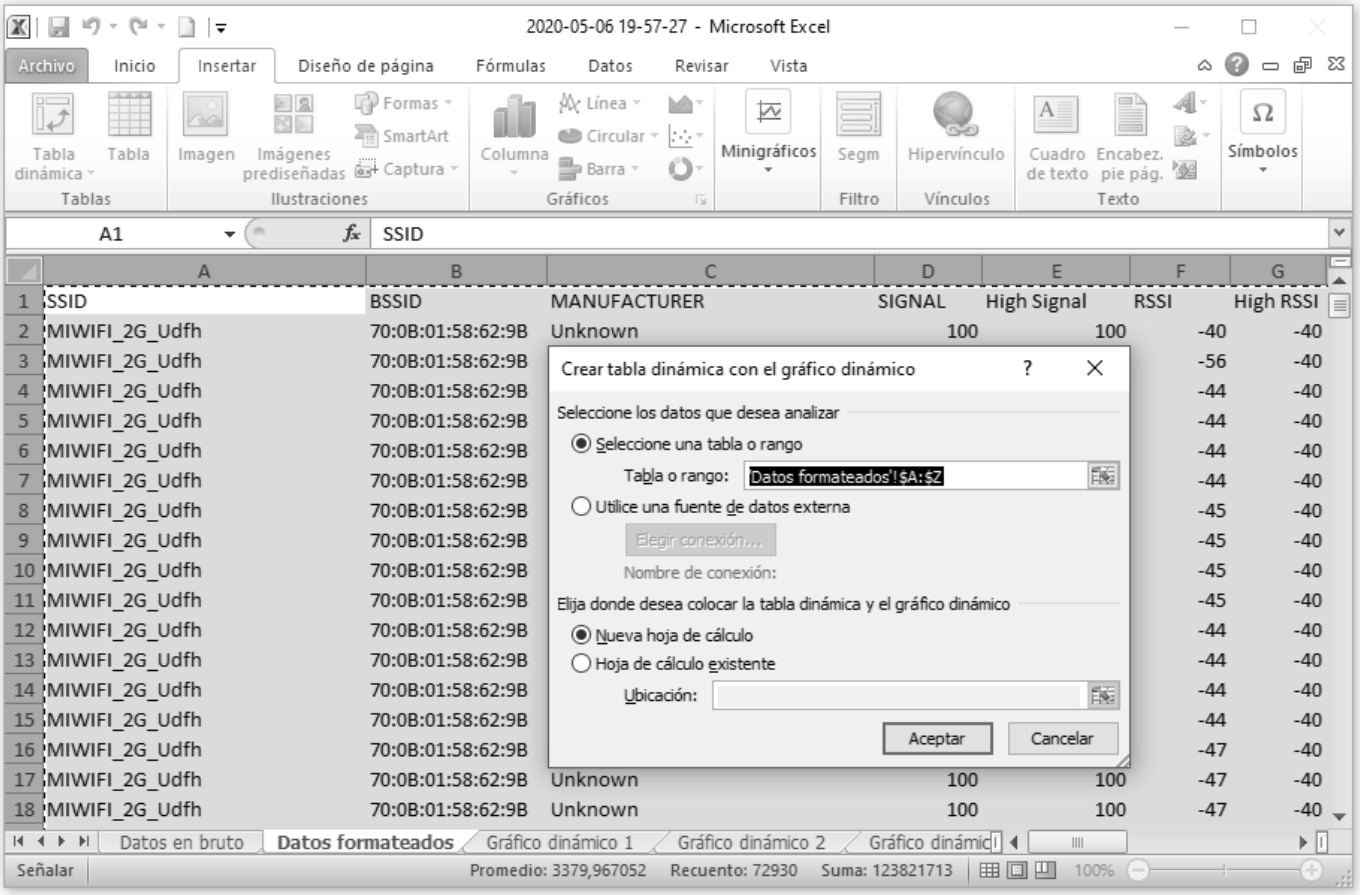


Figura L.20. Crear un gráfico dinámico.

Nada más crear el gráfico dinámico se abre la hoja que lo contiene, donde pasamos a diseñarlo (ver figura L.21). El diseño de gráficos dinámicos es un poco complejo, así que vamos a hacer uno paso a paso a modo de ejemplo.

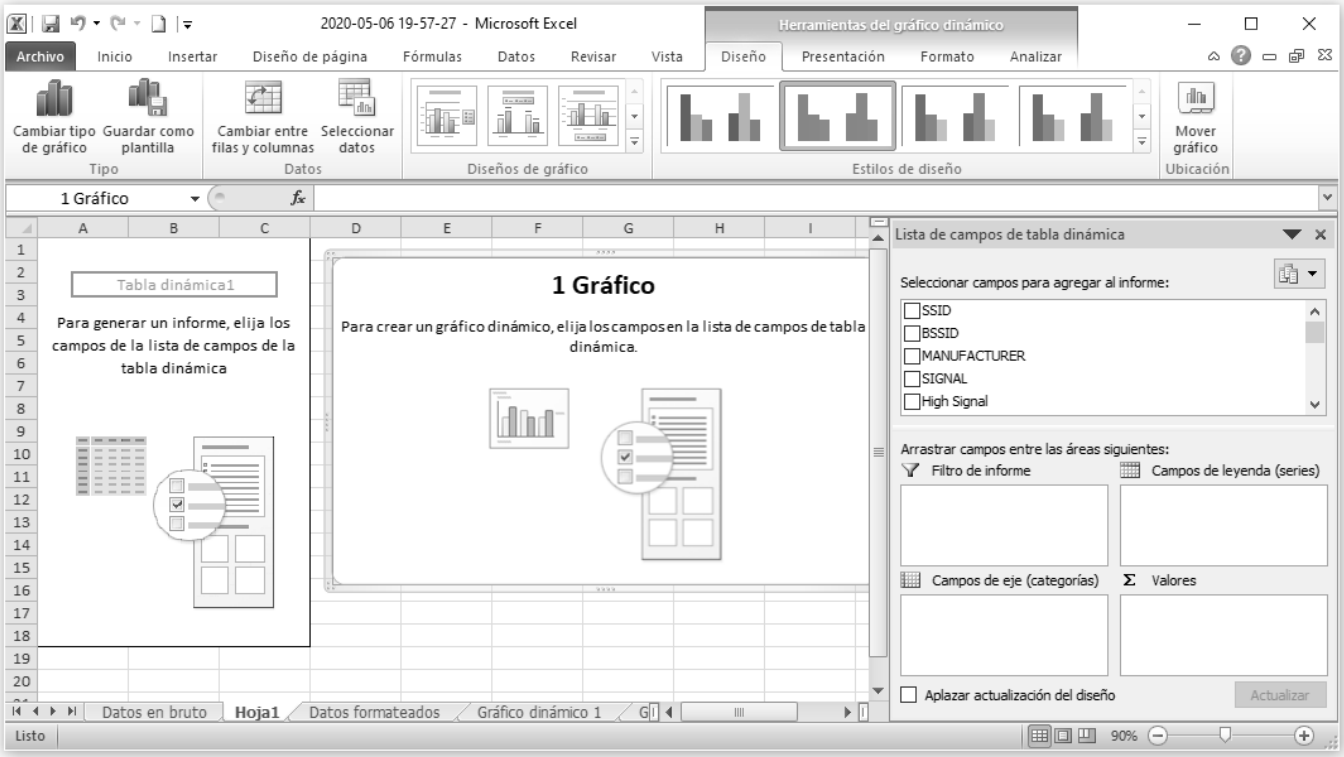


Figura L.21. Ventana de diseño de un gráfico dinámico.

Para empezar, en la "lista de campos de tabla dinámica", seleccionamos el campo SSID, y lo arrastramos y soltamos en la ventana "Campos del eje (categorías)", ver figura L.22. Veremos que el gráfico dinámico (y su tabla dinámica asociada) se van construyendo poco a poco. Por el momento, el gráfico sigue vacío.

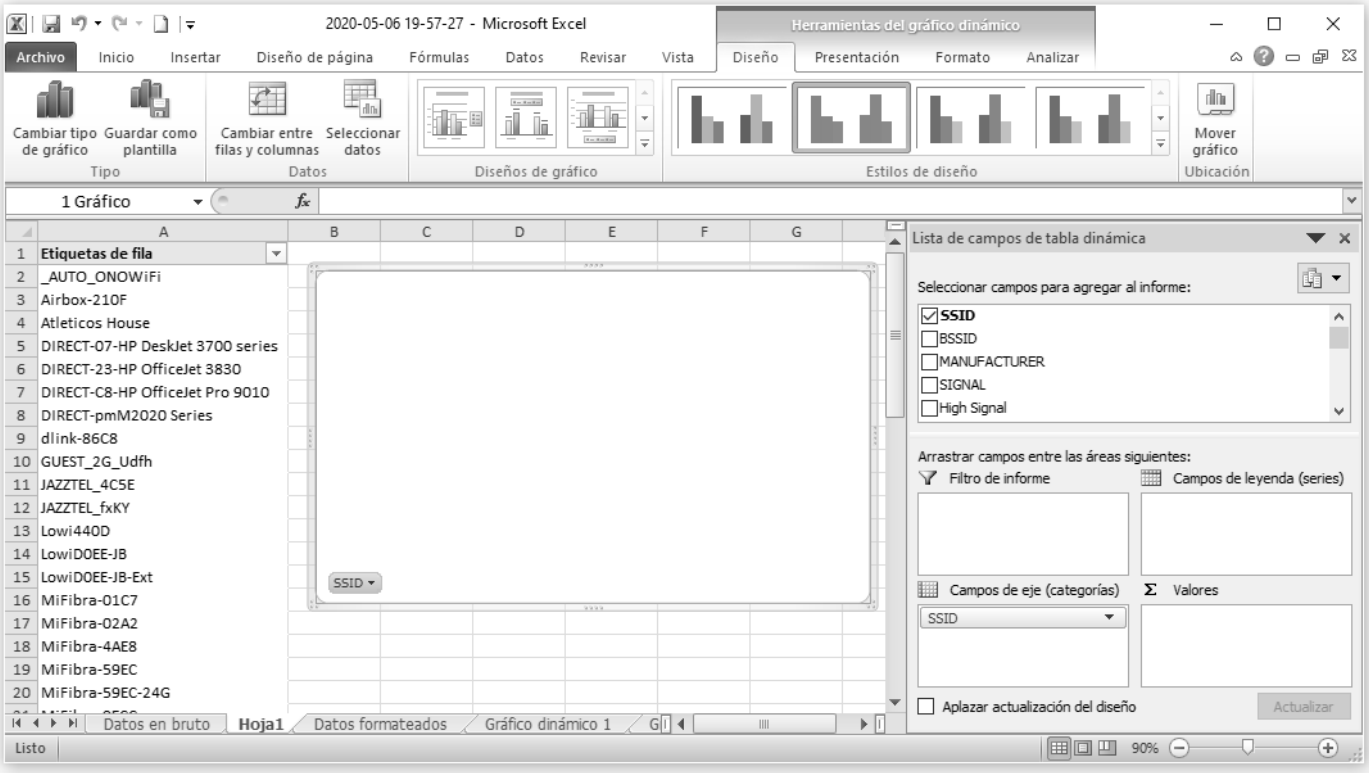


Figura L.22. Construyendo un gráfico dinámico (1).

Ahora arrastramos y soltamos el campo CHANNEL en la ventana "Campos de leyenda (series)", ver figura L.23.

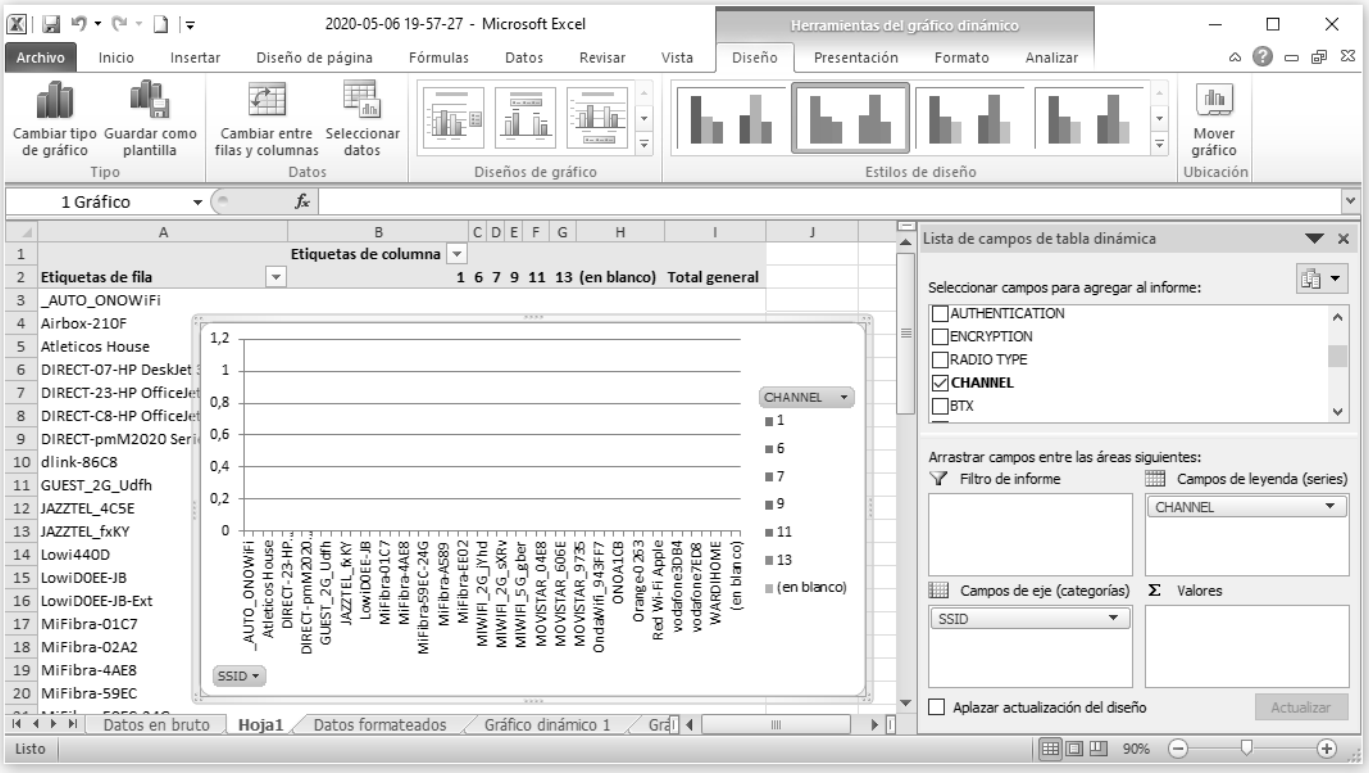


Figura L.23. Construyendo un gráfico dinámico (2).

A continuación arrastramos y soltamos el campo RSSI a la ventana " ∑ Valores". Veremos que dentro de esta ventana aparece el desplegable "Cuenta de RSSI". Pinchamos en la flechita de ese desplegable, y en la lista de menús emergente, elegimos la opción "Configuración de campo de valor…", ver figura L.24. En la ventana de configuración, cambiamos la "Cuenta de RSSI" por "Promedio de RSSI", y pulsamos en Aceptar (ver figura L.25). Tras hacer esto, el gráfico está terminado, pero muestra un aspecto bastante confuso, como ilustra la figura L.26.

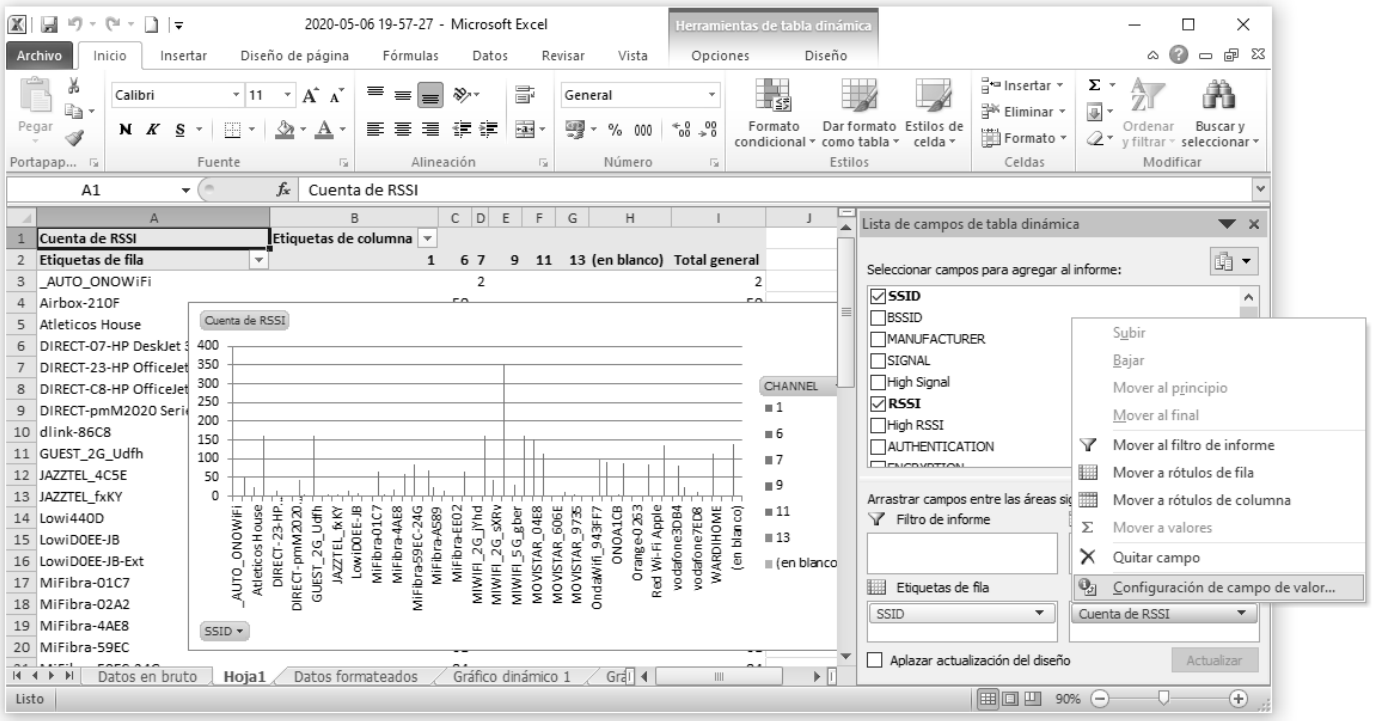


Figura L.24. Construyendo un gráfico dinámico (3).

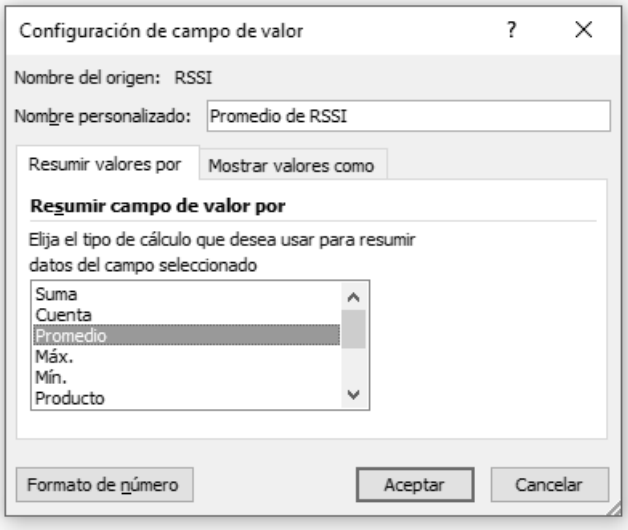


Figura L.25. Construyendo un gráfico dinámico (4).

Para mejorar el aspecto del gráfico, simplemente vamos a editarlo como cualquier otro gráfico de Excel. Primero, vamos a cambiar el formato del eje vertical: Para ello, pinchamos con el botón derecho del ratón sobre el eje vertical, y en el menú desplegable seleccionamos "Dar formato al eje…". En la ventana de formato, fijamos los ajustes mostrados en la figura L.27. Una vez hecho esto, el gráfico adoptará la apariencia de la figura L.28.

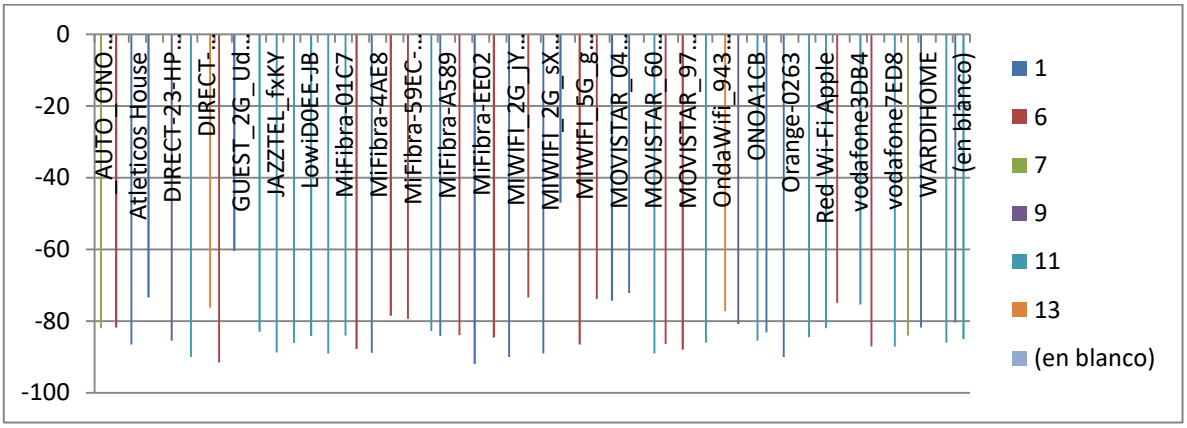


Figura L.26. Construyendo un gráfico dinámico (4).

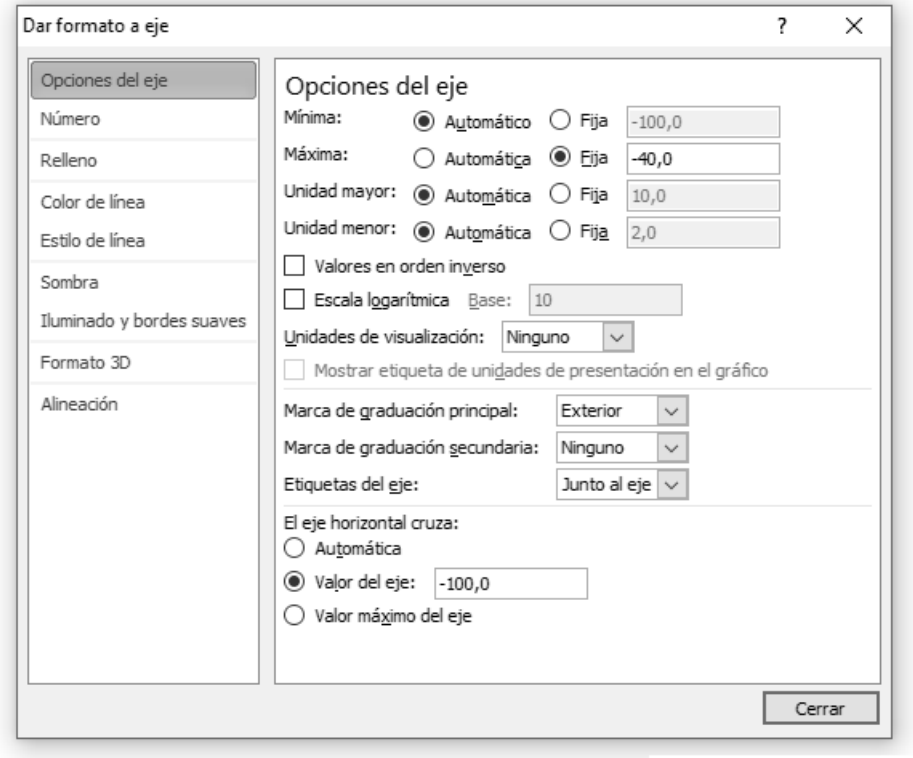


Figura L.27. Construyendo un gráfico dinámico (5).

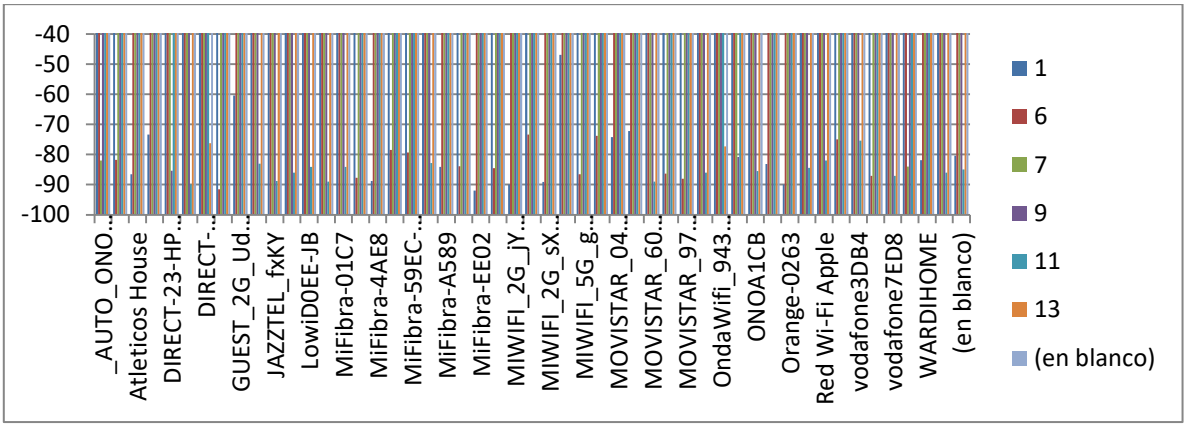


Figura L.28. Construyendo un gráfico dinámico (6).

El gráfico sigue siendo un poco confuso, debido a que mi captura de datos incluía muchísimas redes (muchos SSIDs). Si ese es también tu caso, vamos a filtrar datos en el propio gráfico para mostrar solo unas pocas

redes. Para ello, pulsamos en el botón SSID del gráfico dinámico, y en la ventana de filtrado, seleccionamos unas cuantas redes, ver figura L.29.

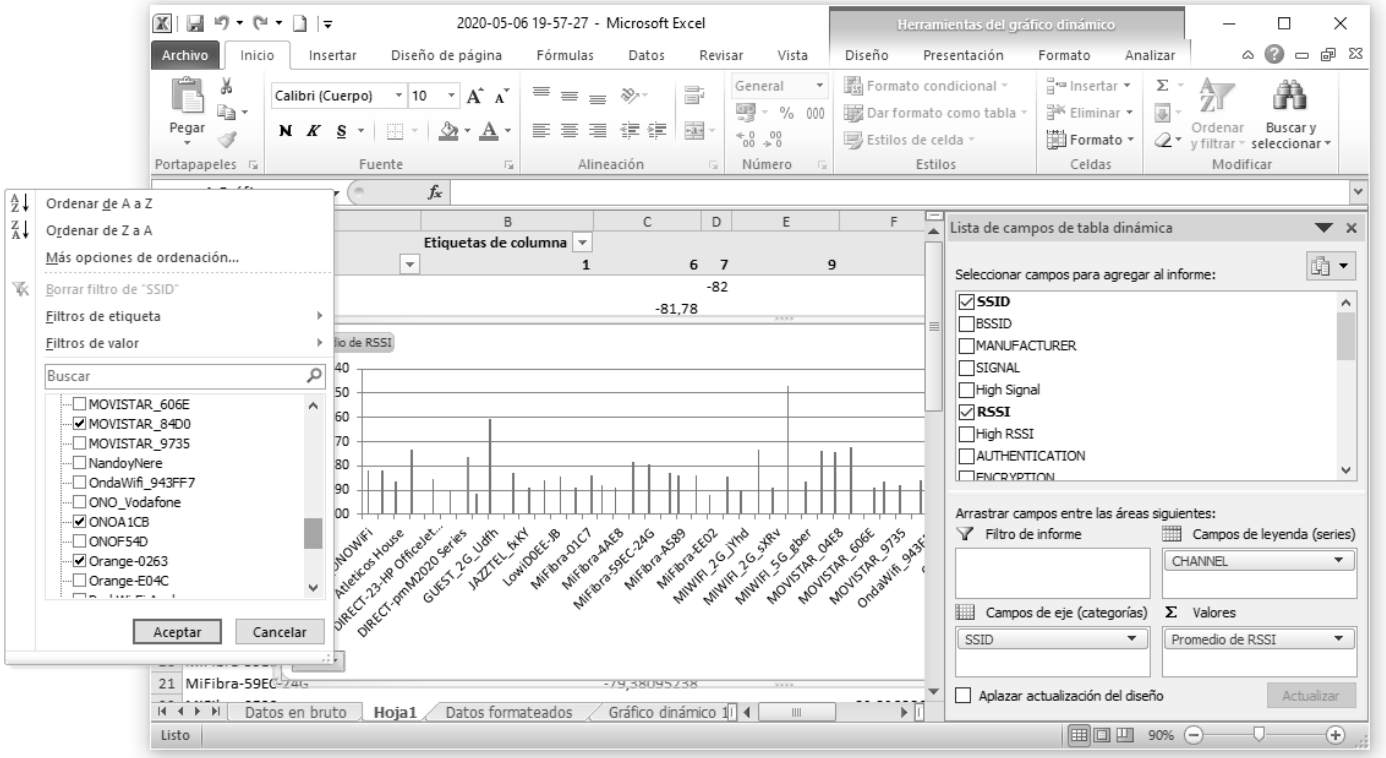


Figura L.29. Construyendo un gráfico dinámico (7).

Por último, añadimos unos rótulos a los ejes vertical y horizontal, cambiamos el color de las series, cambiamos la posición de la leyenda, y el gráfico queda como muestra la figura L.30.

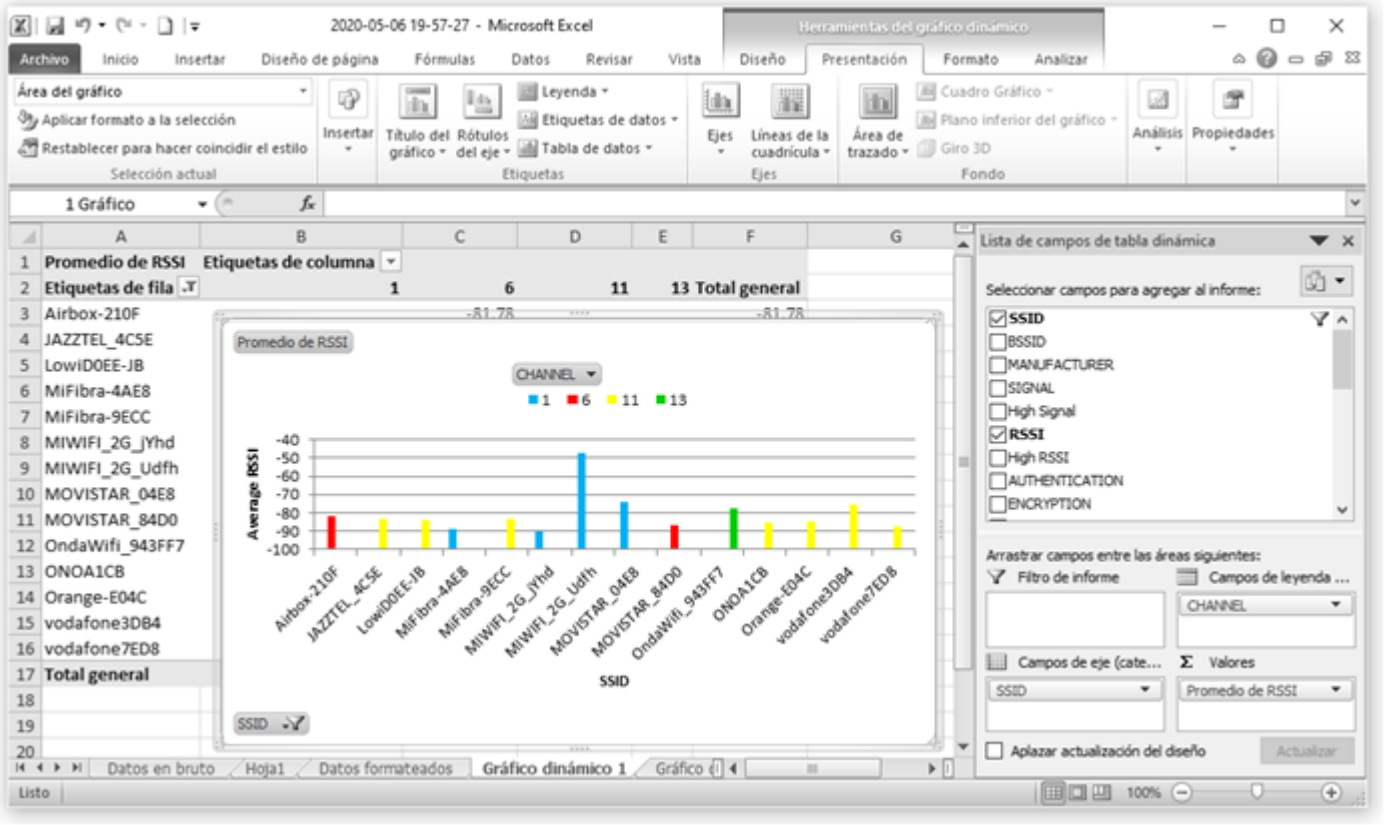


Figura L.30. Aspecto final del gráfico dinámico.

Es importante saber interpretar este gráfico: Para cada red en el eje horizontal, el gráfico nos informa de dos cosas: El eje vertical indica el valor promedio del nivel de señal recibido (RSSI), y el color de la barra correspondiente nos dice el canal en el que opera, según el color mostrado en la leyenda.

A modo de ejercicio, otros gráficos dinámicos que podemos probar a construir son los siguientes:

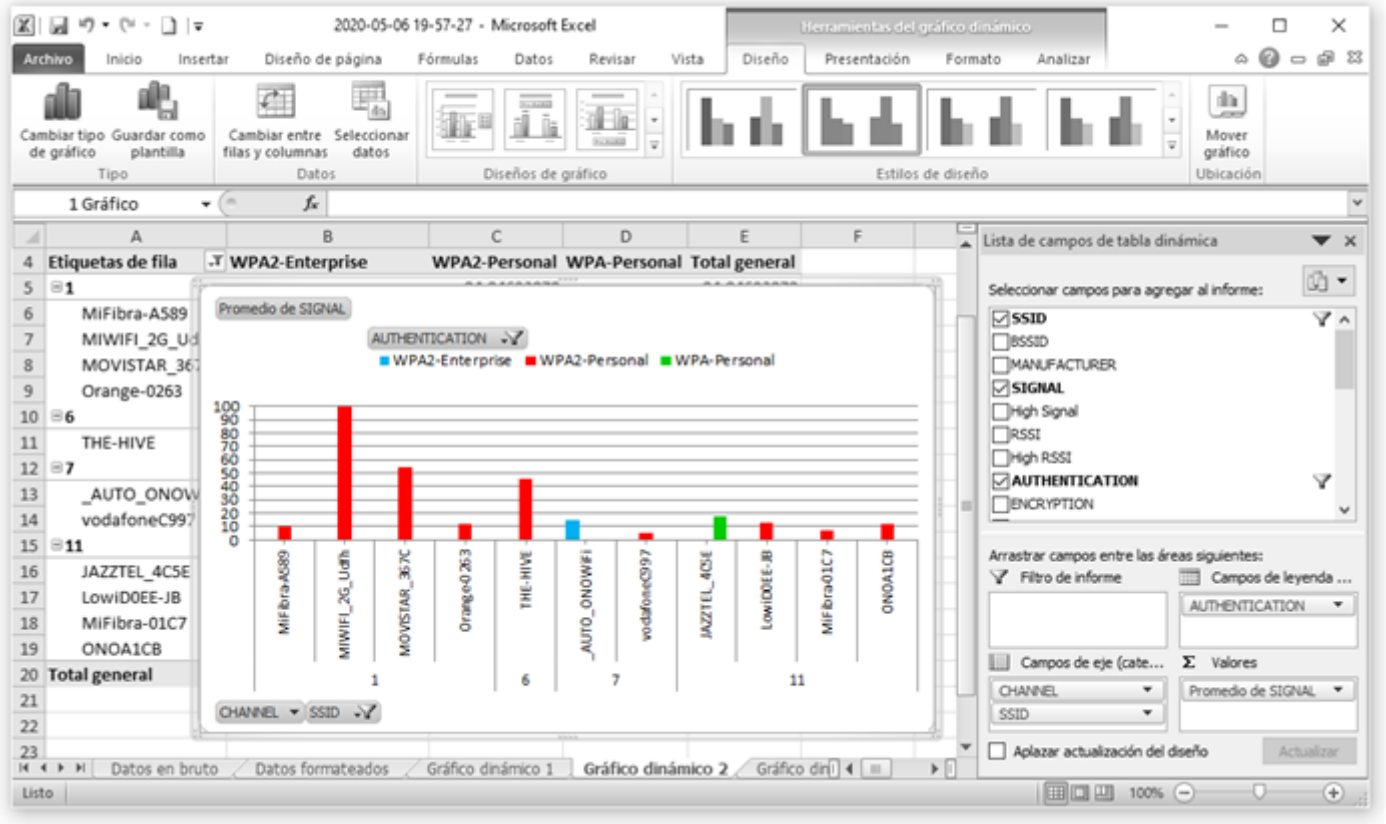


Figura L.31. Ejemplo de gráfico dinámico (1).

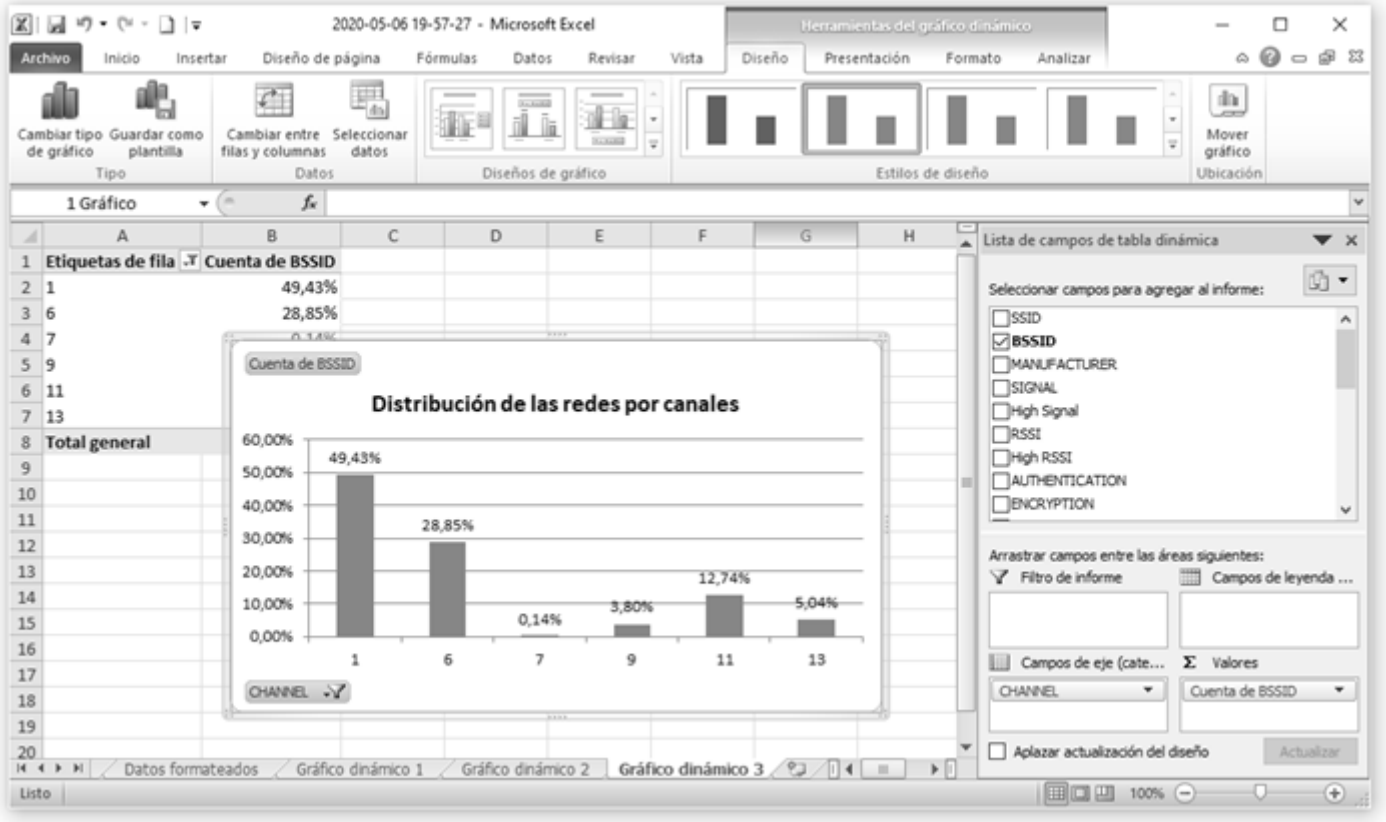


Figura L.32. Ejemplo de gráfico dinámico (2).

PRÁCTICA M. SIMULACIÓN DE REDES LAN INALÁMBRICAS CON PACKET TRACER.

M.1. BSS EN MODO INFRAESTRUCTURA.

En esta primera parte de la práctica vamos a montar un conjunto de servicios básico (BSS) con un punto de acceso (AP). Recordemos que a esta arquitectura se le denomina **modo infraestructura**.[17] La topología final de la red a montar es la mostrada en la figura M.1. Aquí vamos a construirla paso a paso, porque hay algunos procedimientos nuevos que debemos aprender.

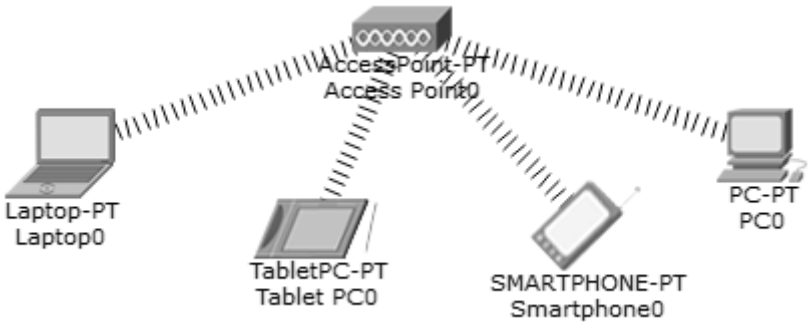


Figura M.1. BSS en modo infraestructura.

AÑADIR LOS DISPOSITIVOS.

En esta práctica vamos a conectar cuatro dispositivos a un AP. Lo primero que vamos a hacer es ubicar en la zona de trabajo los cinco componentes. El portátil (Laptop0), el ordenador de sobremesa (PC0), la tableta (Tablet PC0), y el teléfono móvil (Smartphone0) los encontraremos en la bandeja "End Devices". Por su parte, el AP está disponible en la bandeja "Wireless Devices". Nosotros usaremos el AP genérico denominado "AccessPoint PT".

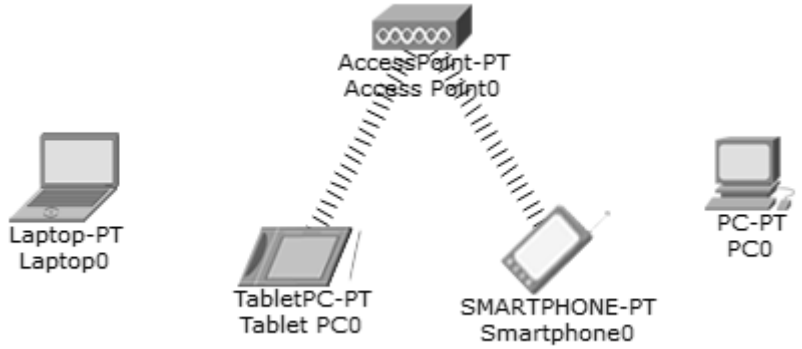


Figura M.2. La topología de la red tras añadir los cinco dispositivos.

Tras ubicar todos los dispositivos en la zona de trabajo, y al cabo de un par de minutos, veremos que la tableta y el teléfono móvil establecen automáticamente una conexión inalámbrica con el AP. Sin embargo, el portátil y el PC permanecen desconectados (ver figura M.2). La razón de ello es que el portátil y el PC no disponen por defecto de una tarjeta de red inalámbrica; debemos añadírsela nosotros.

CONFIGURACIÓN FÍSICA DEL PORTÁTIL Y DEL PC.

Si pinchamos con el ratón sobre la tableta o el teléfono móvil y acudimos a la pestaña "Config", veremos que estos dispositivos no disponen de un interfaz "FastEthernet0". En lugar de ello tienen un interfaz

[17] Por alguna razón, Packet Tracer no permite montar redes WLAN en modo ad-hoc.

"Wireless0" para conectarse a redes LAN inalámbricas, y un interfaz "3G/4G Cell1" para conectarse a redes de telefonía móvil, ver figura M.3.

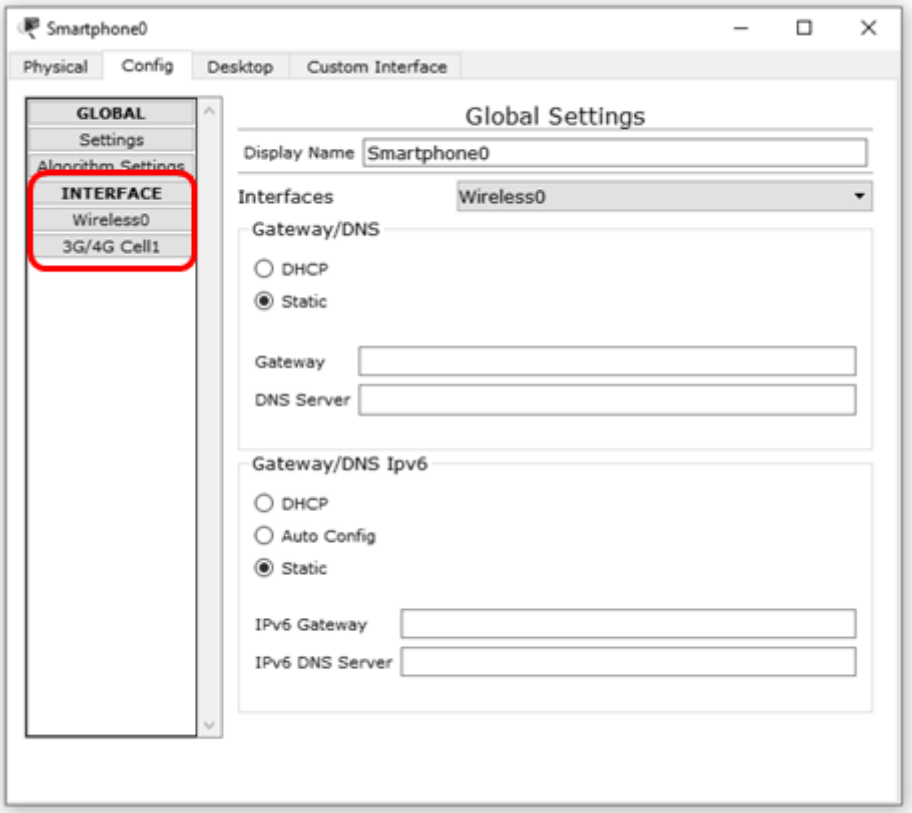


Figura M.3. Interfaces del teléfono móvil Smartphone0.

Por el contrario, el portátil y el PC disponen de un interfaz "FastEthernet0", pero no de un interfaz "Wireless0", ver figura M.4.

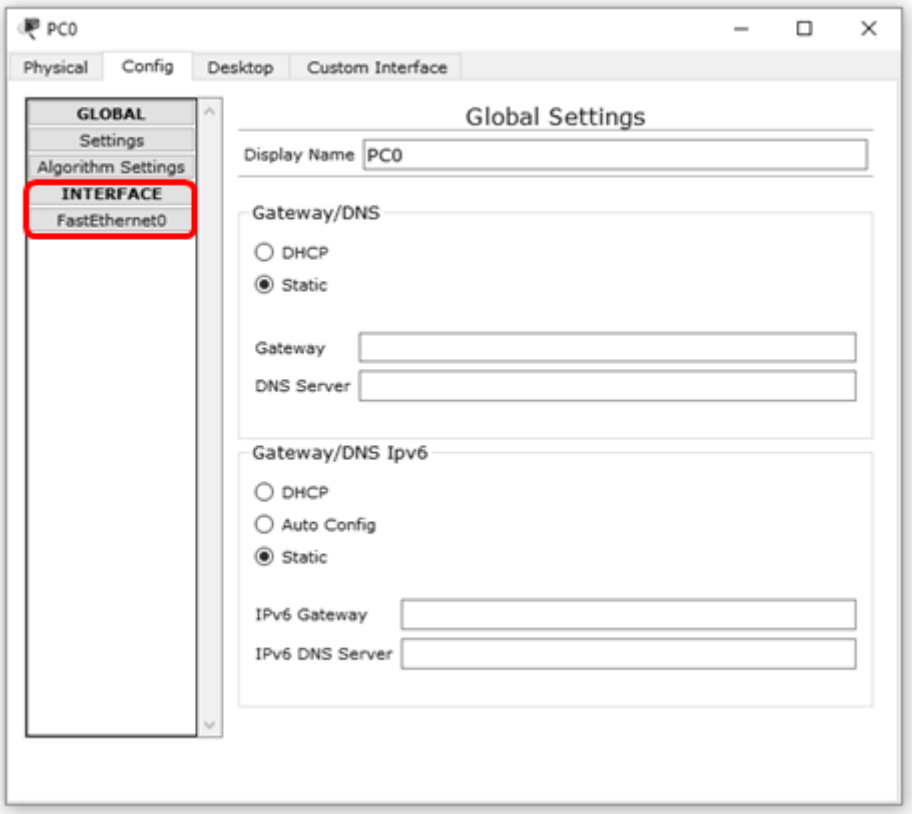


Figura M.4. Interfaces del ordenador PC0.

Esta es la razón por la que el portátil y el PC no se conectan automáticamente al AP. Para solventar este problema, debemos modificar el equipamiento hardware de estos dos dispositivos.

Para ello, pinchamos en el portátil, y acudimos a la pestaña "Physical". Aquí es donde podemos cambiar la configuración hardware del equipo, ver figura M.5.

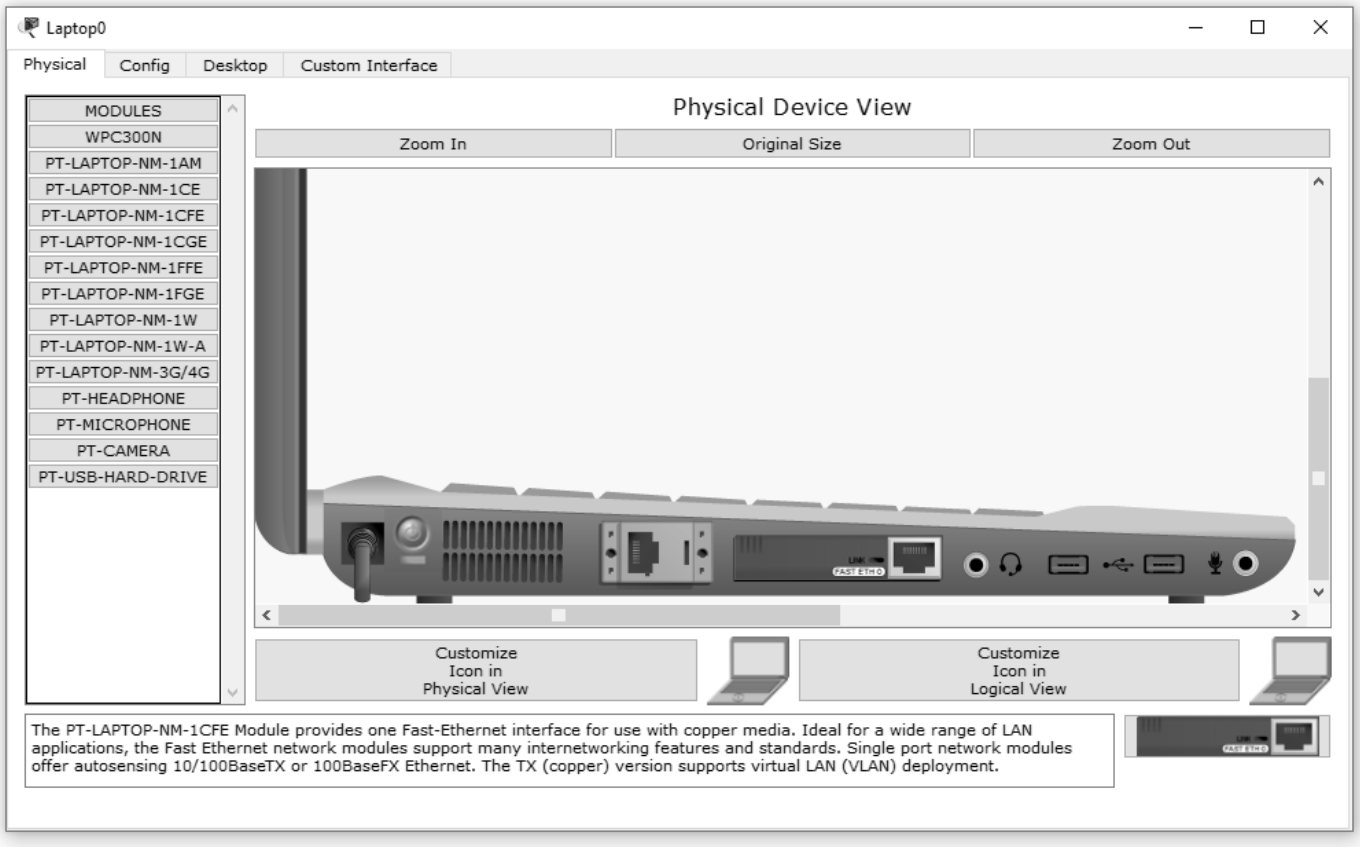


Figura M.5. Configuración física del portátil Laptop0.

Por defecto, los portátiles vienen equipados con una tarjeta de red "PT-LAPTOP-NM-1CFE" que les proporciona el interfaz de tipo FastEthernet. Nosotros vamos a cambiar este módulo por una tarjeta de red inalámbrica. Para ello, seguimos estos pasos:

1) Apagamos el portátil Laptop0 pulsando en el botón a la izquierda del portátil, junto al cable de alimentación. Al apagar el portátil, el LED indicador verde debería apagarse.
2) Pinchamos sobre la tarjeta de red Ethernet "PT-LAPTOP-NM-1CFE" insertada en la parte central del portátil (junto al conector de los auriculares) y lo arrastramos fuera del portátil. Nos deshacemos de ella soltándola en la lista de módulos a la izquierda de la pantalla de configuración física. La ranura donde estaba conectada la tarjeta debería quedar vacía, ver figura M.6.

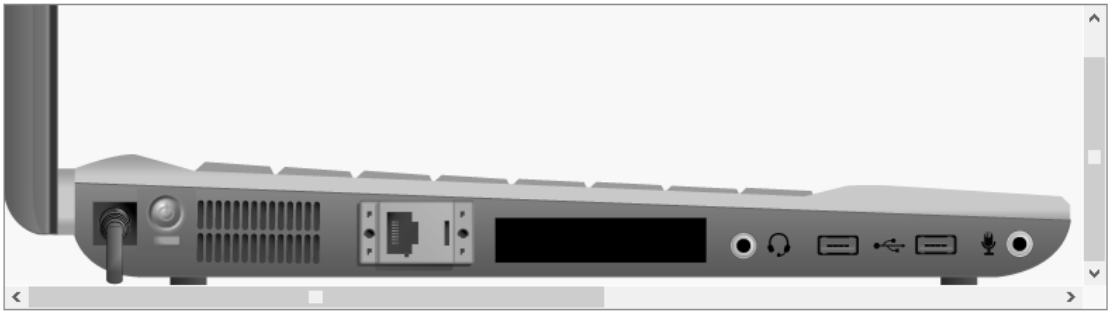


Figura M.6. Desinstalación de la tarjeta de red FastEthernet del portátil Laptop0.

3) En la lista de módulos, tomamos una tarjeta de red inalámbrica. Podemos elegir entre las tarjetas "Linksys-WPC300N" o "PT-LAPTOP-NM-1W". Ambas proporcionan una interfaz inalámbrica a $2{,}4\ Ghz$. Nosotros hemos elegido la tarjeta "Linksys-WPC300N", y la hemos arrastrado hasta conectarla a la ranura vacía donde estaba la antigua tarjeta Fast Ethernet, ver figura M.7.

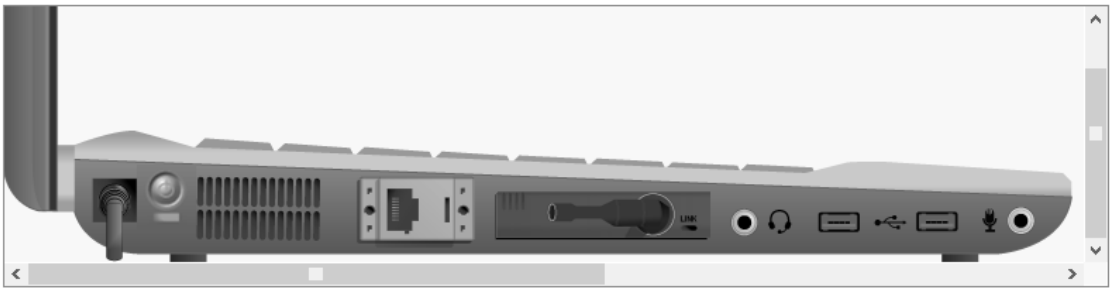


Figura M.7. Instalación de una tarjeta de red inalámbrica en el portátil Laptop0.

4) Por último, volvemos a encender el portátil presionando en el botón de encendido. El LED indicativo debería brillar en verde.

Al cerrar la ventana de configuración física, deberíamos ver que el portátil ya es capaz de conectarse inalámbricamente al AP, ver figura M.8.

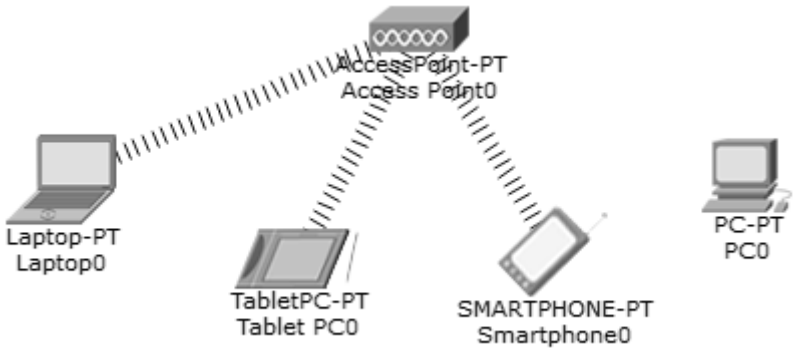


Figura M.8. Conexión del portátil al AP.

Ahora debemos repetir el mismo proceso con el ordenador PC0. De nuevo, pinchamos sobre él y acudimos a la pestaña "Physical". Apagamos el ordenador pulsando en el botón rojo. Extraemos la tarjeta de red FastEthernet, y en su lugar instalamos una tarjeta inalámbrica "WPM300N". (La figura M.9 muestra la apariencia del PC0 antes y después de la reconfiguración física). Para terminar, volvemos a pulsar en el botón de encendido del ordenador. Al cabo de unos instantes, deberíamos ver que el PC0 se conecta inalámbricamente al AP, como muestra la figura M.1 al principio de la práctica.

CONFIGURACIÓN LÓGICA DE LOS EQUIPOS.

Una vez establecida la configuración hardware de los equipos que la necesitaban, vamos a realizar la configuración lógica de los distintos dispositivos de la red.

Configuración del AP.

Pulsamos sobre el AP, y en la ventana emergente, seleccionamos la pestaña "Config". En la sección de Interfaces, pulsamos sobre "Port0". Aquí podemos configurar el ancho de banda ($10\ Mbps$ o $100\ Mbps$) y el modo de transmisión (Half - dúplex o full - dúplex). Nosotros vamos a configurarlo para que seleccione ambas opciones automáticamente. Además, nos aseguramos de que el puerto esté habilitado (casilla de "Port Status" activada), ver figura M.10.

A continuación, pulsamos en "Port1". Aquí es donde definimos el nombre identificador de la red (BSSID), el canal de transmisión (del 1 al 11), y el método de autenticación (WEP, WPA, desactivado, etc.), y el tipo de encriptado. El BSSID por defecto en Packet Tracer es "Default"; nosotros lo cambiaremos, por ejemplo, a "My_WiFi_Network". Además, dejamos el canal por defecto (en mi caso, el 6), y deshabilitamos la autenticación y el encriptado. La figura M.10 muestra cómo debería la ventana de configuración del AP.

Figura M.9. Desinstalación de la tarjeta FastEthernet e instalación de la tarjeta inalámbrica en el ordenador PC0.

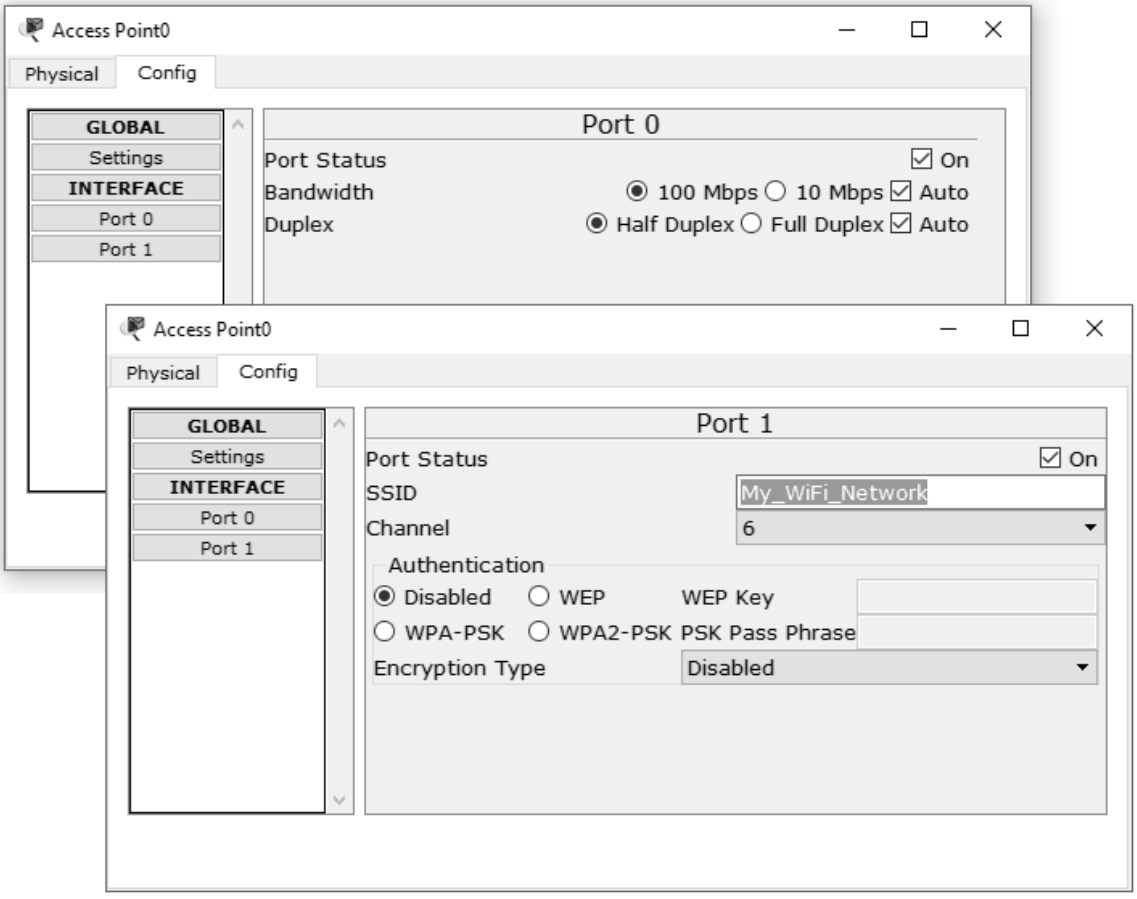


Figura M.10. Configuración del AP.

Configuración de los dispositivos terminales.

Pulsamos en uno de los dispositivos conectados al AP, por ejemplo, el portátil Laptop0. En la ventana emergente, seleccionamos la pestaña "Config". Dentro de esta pestaña acudimos al interfaz "Wireless0", y escribimos el SSID de la red a la que nos queremos conectar (cambiamos "Default" por "My_WiFi_Network"). A continuación vamos a realizar la configuración IP: Primero, cambiamos la asignación dinámica mediante DHCP por una asignación estática, para que la dirección IP asignada al portátil siempre sea la misma. Nosotros vamos a darle al portátil la dirección IP privada 192.168.1.101. Después de ello, pulsamos sobre la máscara de subred, que automáticamente tomará el valor 255.255.255.0. La figura M.11 muestra cómo debería quedar finalmente la configuración lógica del portátil Laptop0.

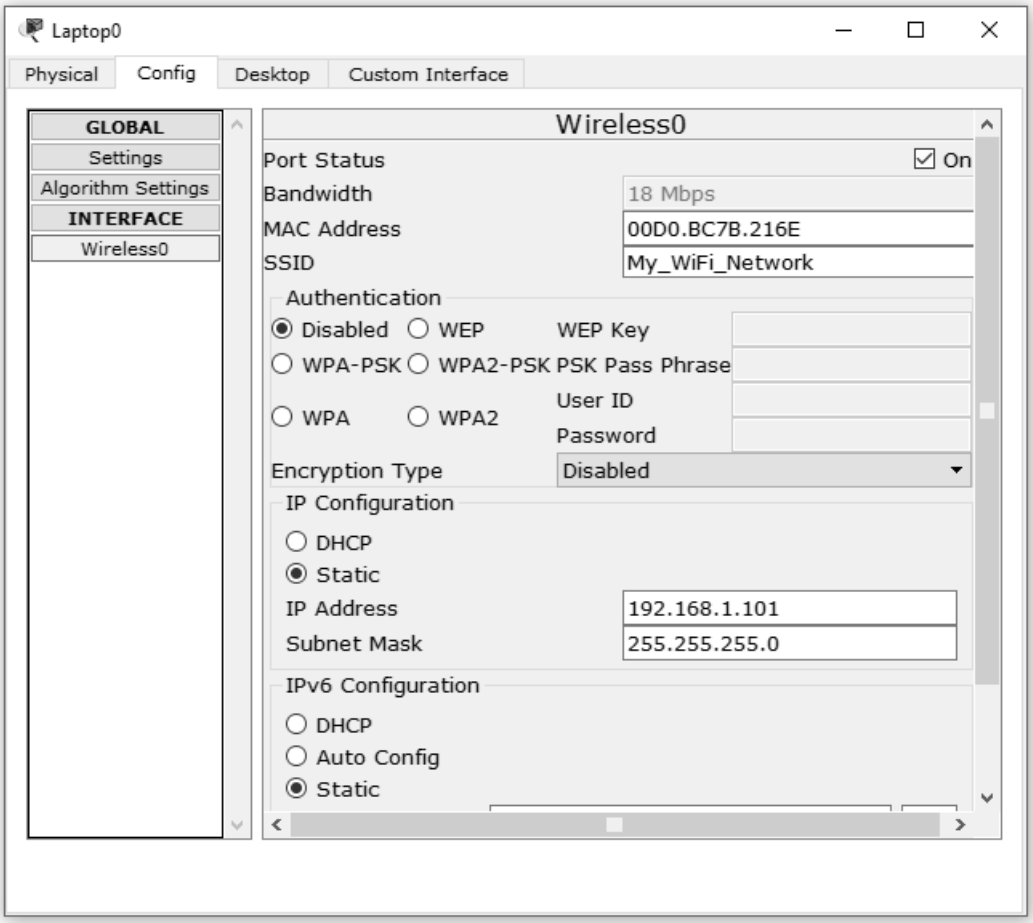


Figura M.11. Configuración lógica del portátil Laptop0.

Ahora debemos hacer lo mismo para el resto de dispositivos finales de la red. Tan solo debemos tener la precaución de asignar una dirección IP distinta a cada dispositivo, pero todas pertenecientes a la misma red. Por ejemplo, a la tableta le asignamos la dirección 192.168.1.102, al teléfono la dirección 192.168.1.103, y al PC la dirección 192.168.1.104. Como vemos, todos ellos pertenecen a la red 192.168.1.0.

Para terminar vamos a comprobar la conectividad entre los dispositivos de la red. Para ello, hacemos un `ping` (a través de la consola de comandos, o de la ventana de simulación) desde el portátil al teléfono móvil. Si el portátil recibe respuestas positivas, la red está bien configurada y los dispositivos pueden comunicarse, ver figura M.12.

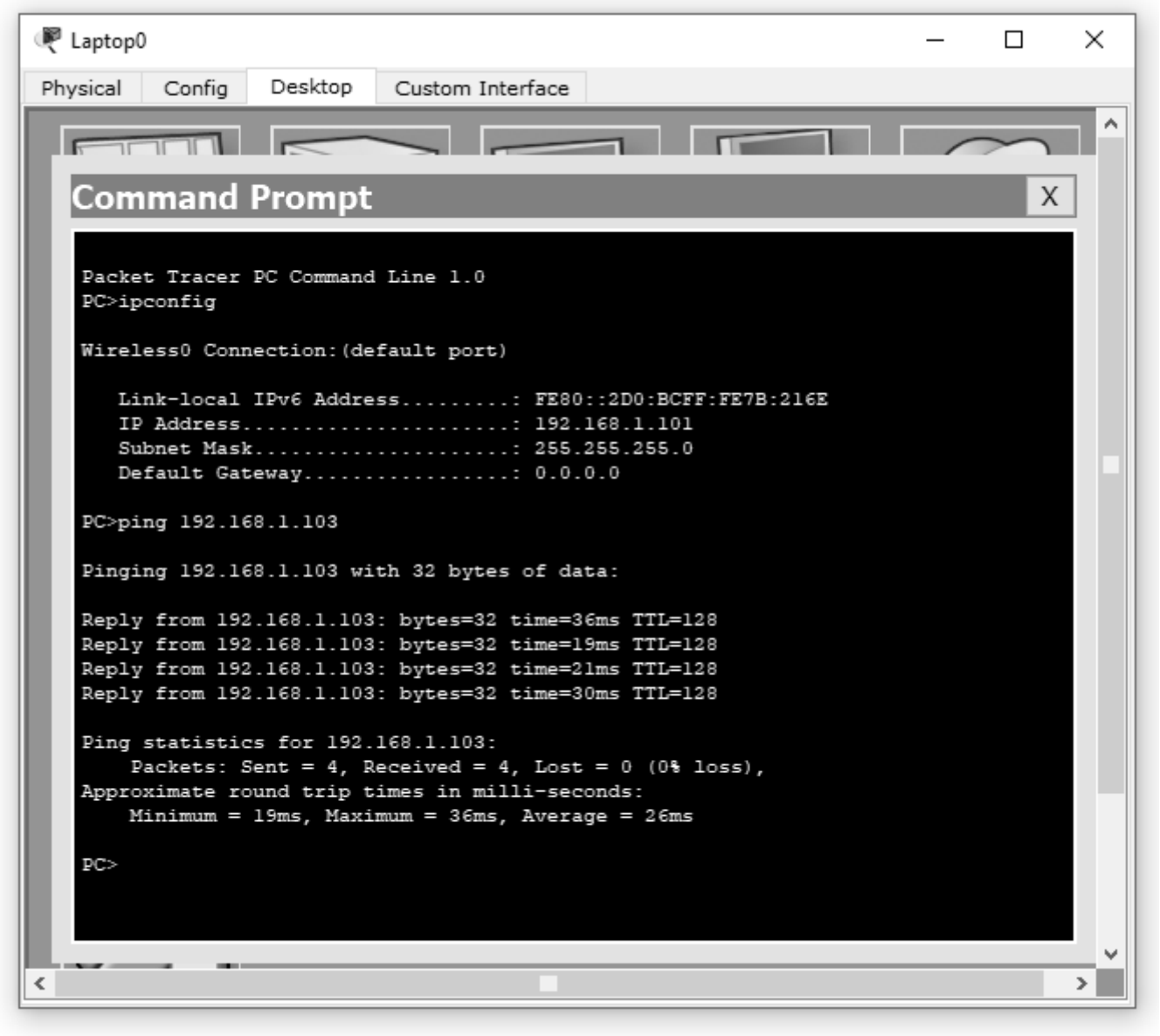


Figura M.12. Resultado de un `ping` desde el portátil Laptop0 al teléfono móvil Smartphone0.

M.2. UN ESS CON DOS WLANs.

A continuación, vas a montar un conjunto de servicios ampliado (ESS) formado por dos BSSs (BSS-A y BSS-B) con sus respectivos APs. La figura M.13 muestra la topología de esta red.

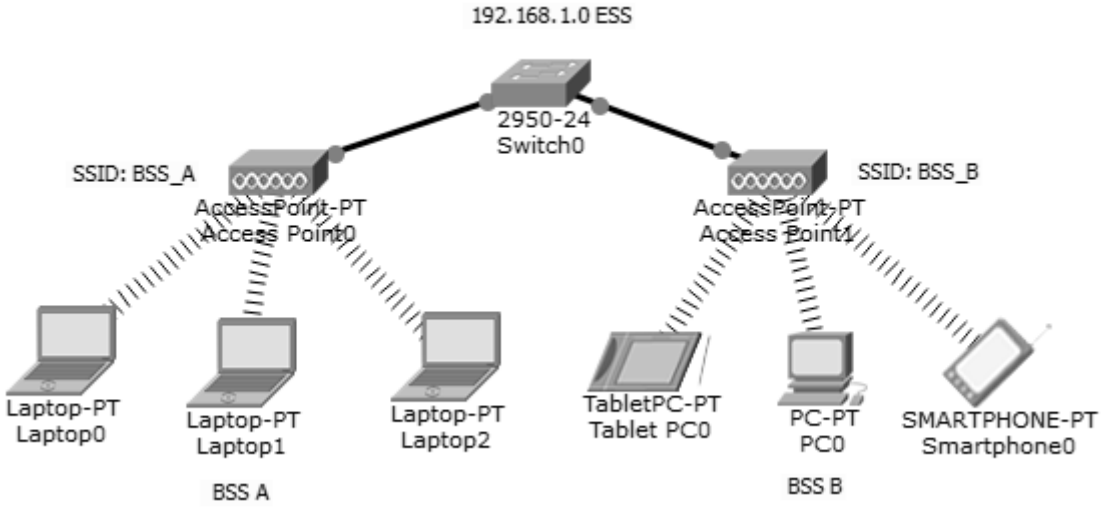


Figura M.13. ESS formado por dos BSSs.

Monta la topología física, modifica el hardware de aquellos equipos que así lo necesiten, configura el SSID de los APs, conecta cada máquina a su AP correspondiente, y asígnales una dirección IP privada fija (por ejemplo, el Laptop0 tendrá la IP 192.168.1.101, el Laptop1 la IP 192.168.1.102, etc., terminando con el Smartphone0, que tendrá la IP 192.168.1.106).

Para comprobar la conectividad de la red, haz un `ping` entre dos estaciones de la misma BSS, y entre dos estaciones en diferentes BSSs.

M.3. INTERRED FORMADA POR DOS ESSs.

Ahora vamos a montar una interred formada por dos ESSs. Imaginemos que una empresa tiene dos sedes, una en Londres y otra en París. La empresa desea configurar una interred que conecte las redes de sus dos sedes. La sede de Londres dispone de dos redes inalámbricas, la del edificio 1 (cuyo SSID es LONDON_WLAN1) y la del edificio 2 (con SSID LONDON_WLAN2). Ambas redes inalámbricas están interconectadas mediante un conmutador para constituir una única red que usa el rango de direcciones IP 192.168.1.0. La sede de París solo dispone de un edificio, donde únicamente hay una red inalámbrica cuyo SSID es PARIS_WLAN1. La red de París usa el rango de direcciones IP 192.168.2.0. Como vemos, se trata de dos redes independientes, que deben interconectarse mediante un rúter. La topología de la interred a montar se muestra en la figura M.14.

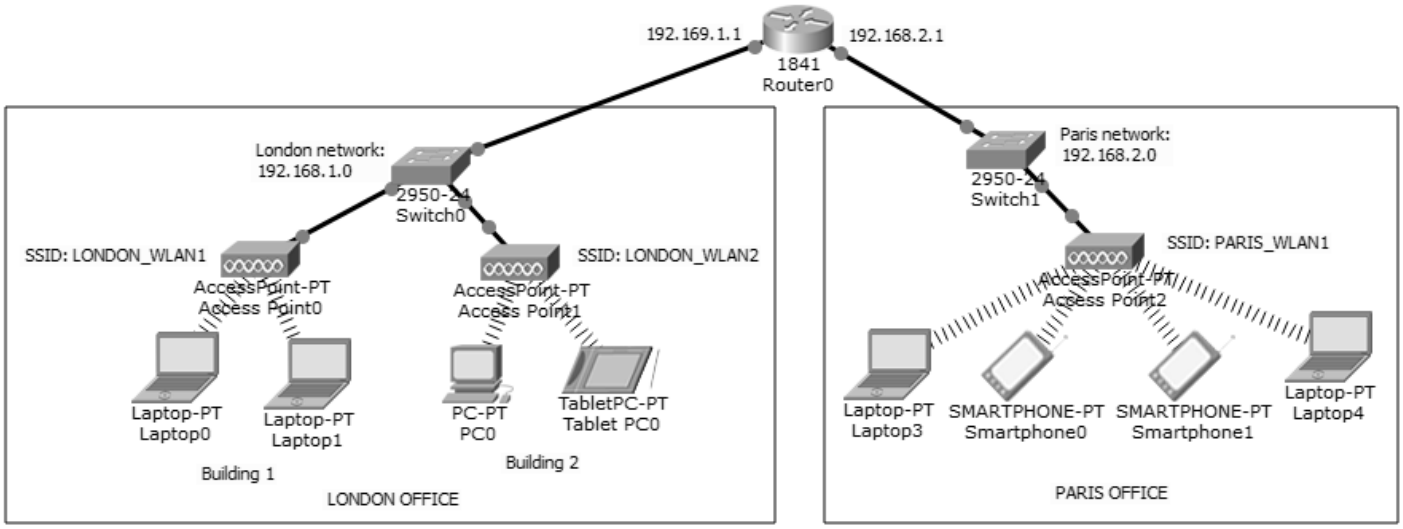


Figura M.14. Interred formada por dos ESSs .

Acuérdate de:

1) Configurar adecuadamente el SSID de cada AP.
2) Configurar adecuadamente el SSID del AP al que debe conectarse cada equipo.
3) Asignar una dirección IP fija adecuada a cada equipo de las dos redes.
4) Asignar una dirección IP correcta a cada uno de los interfaces del rúter (ver práctica J).
5) Indicar la dirección IP de la puerta de enlace (Gateway) que les permite los equipos de una red enviar paquetes a los equipos de la otra red (ver práctica J).

M.4. RED LAN MIXTA CON CONEXIÓN A INTERNET.

Para terminar, vamos a montar una red LAN mixta donde algunos equipos se conectan mediante cable a un hub, y donde otros se conectan inalámbricamente a un AP. Esta red también proporciona conexión a Internet a todos los equipos que tiene conectados.

La figura M.15 muestra la topología de esta red LAN mixta. Notar que hemos simulado la Internet global con un ordenador externo a la red. En la figura podemos encontrar todos los datos que necesitamos para montar la topología de red. Configura los distintos dispositivos (ordenadores, portátiles, teléfonos, AP, rúter, etc.) para asegurarte de que la red funciona debidamente. Cuando la hayas montado, comprueba (mediante `ping`'s) que existe conectividad entre los distintos equipos de la red, y entre estos e Internet.

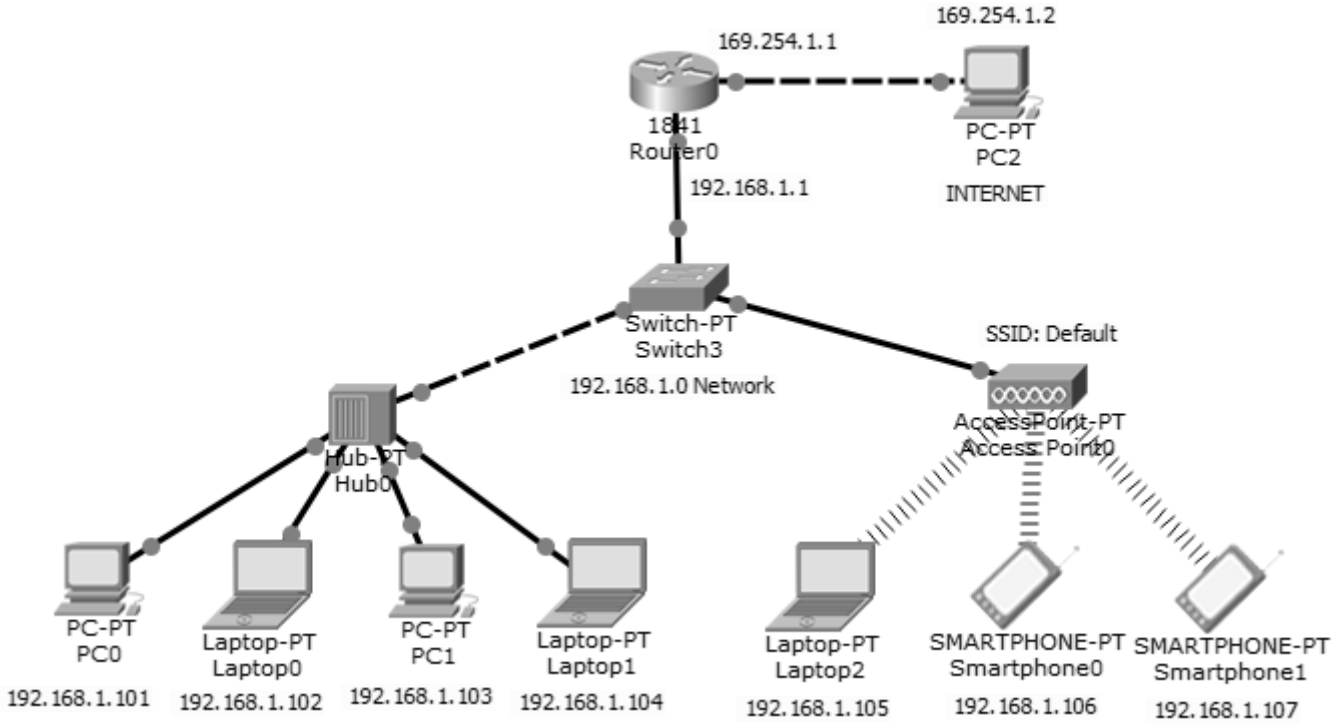


Figura M.15. Red LAN mixta con conexión a Internet.

PRÁCTICA N. RED AD-HOC.

N.1. RED AD-HOC MEDIANTE LÍNEA DE COMANDOS.

Recordemos que una WLAN 802.11 en modo ad – hoc es una red inalámbrica donde los dispositivos se interconectan entre sí sin usar un punto de acceso (AP). En esta práctica vamos a configurar una red ad – hoc usando nuestro ordenador como hot - spot de conexión a la red ad – hoc.

En Windows 10 es fácil establecer una red ad – hoc mediante la consola de comandos, siempre que la tarjeta de red inalámbrica de nuestro ordenador así lo permita. En caso contrario, existen aplicaciones para convertir nuestro ordenador en un hot - spot WiFi, como veremos en el apartado P.2 de esta misma práctica. Para establecer una red ad – hoc en Windows 10 por línea de comandos seguimos los siguientes pasos:

1) Comenzamos arrancando la consola de comandos en *modo administrador*.

2) Una vez abierta la consola de comandos (en modo administrador), lo primero que debemos hacer es averiguar si nuestra tarjeta de red inalámbrica admite una "red hospedada". Una **red hospedada** es una especie de red inalámbrica virtual que se implementa y se muestra como si de una red física real se tratase. Para ello, ejecutamos el comando `netsh wlan show drivers`. Este comando permite mostrar toda la información relacionada con el adaptador de red WLAN. Acudimos a la línea "Red hospedada admitida", y comprobamos si nuestro adaptador permite crear este tipo de redes, ver figura N.1.

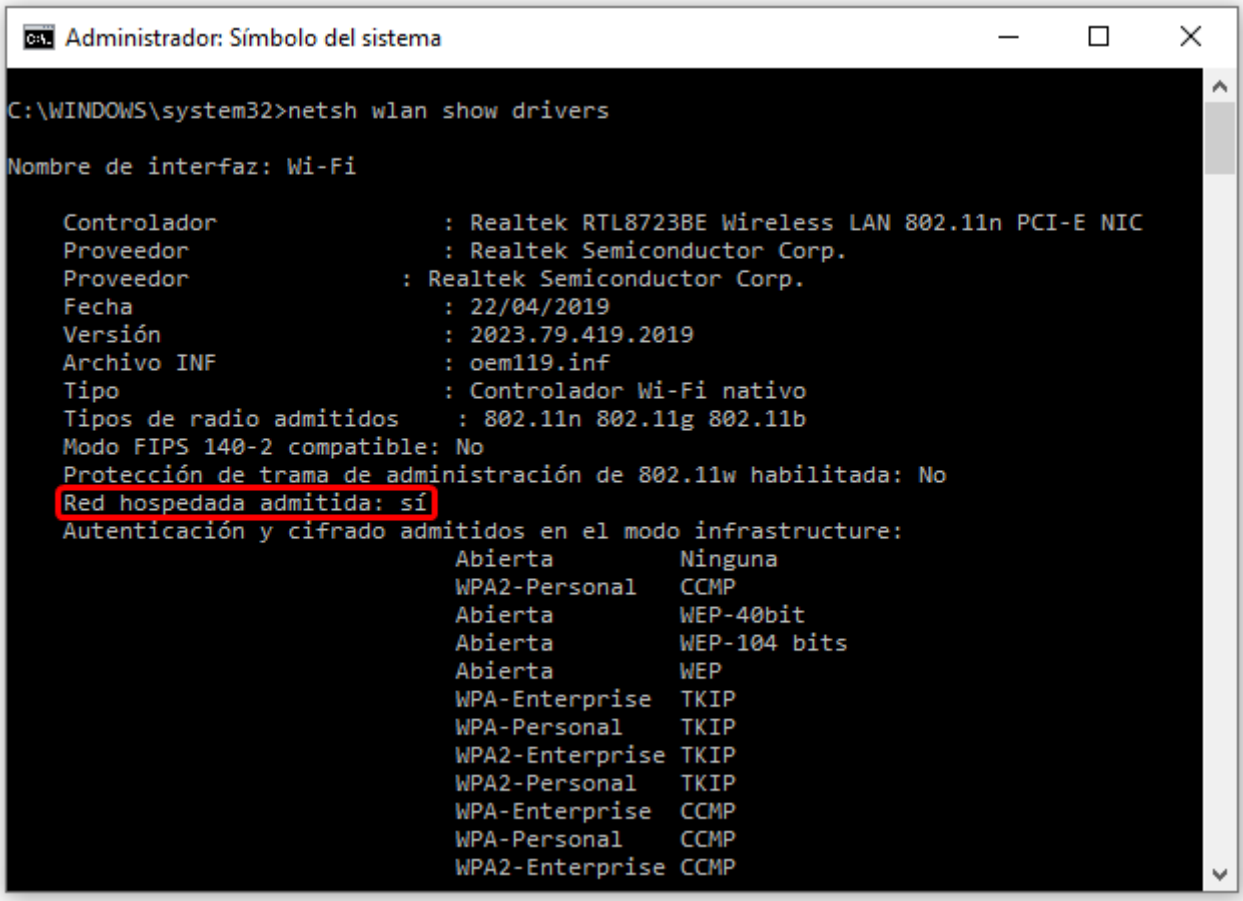


Figura N.1. Resultado del comando `netsh wlan show drivers`.

NOTA: En el caso que nuestro adaptador de red no admita una red hospedada, tal vez se deba a que el controlador (driver) del adaptador no está actualizado. Existen muchas aplicaciones que analizan

nuestro ordenador, detectan controladores no actualizados, y los descargan e instalan automáticamente. Un ejemplo es el programa DriverHub.

3) A continuación, y sin cerrar la ventana de comandos, ejecutamos el comando `netsh wlan set hostednetwork mode=allow ssid=MiRedAdHoc key=123456789` para crear una nueva red hospedada. El parámetro `ssid` es el nombre que le queremos dar a la red, y el parámetro `key` es la contraseña para acceder a ella. En nuestro caso, nuestra red se llama "MiRedAdHoc", y la contraseña es "123456789". La figura N.2 muestra el resultado de ejecutar este comando.

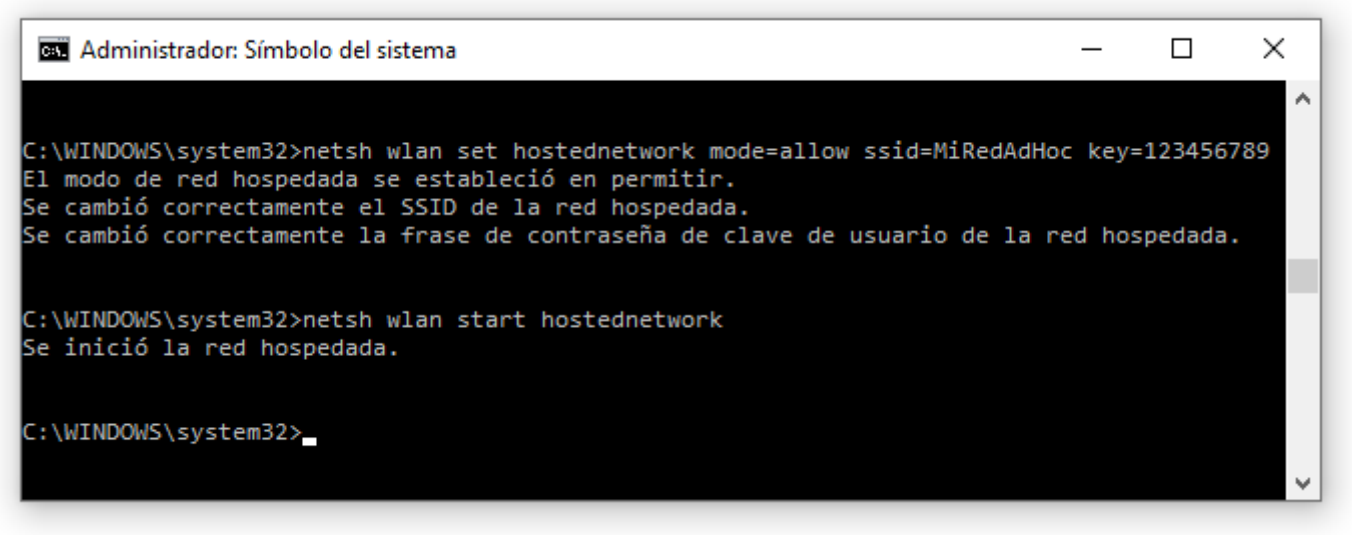


Figura N.2. Crear y levantar una red ad – hoc.

4) A continuación, ejecutamos el comando `netsh wlan start hostednetwork` para levantar la red hospedada recién creada. Si todo ha ido bien, la consola de comandos debería mostrar el mensaje "Se inició la red hospedada", ver figura N.2.

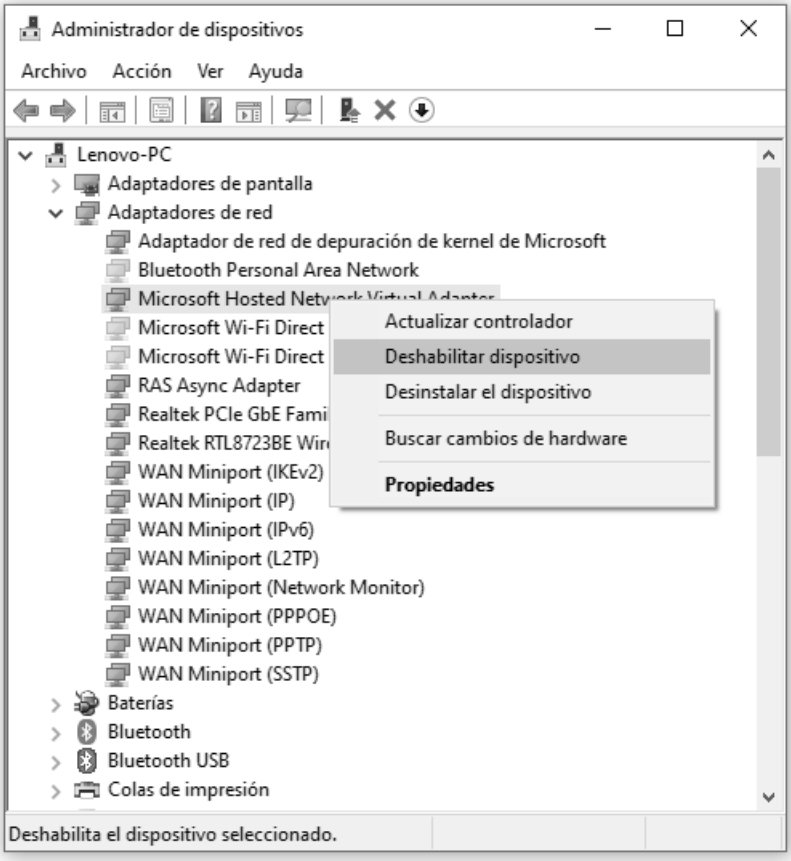


Figura N.3. Adaptador virtual de red hospedada de Microsoft.

En el caso de obtener el mensaje "No se pudo iniciar la red hospedada", hemos de acudir al **administrador de dispositivos** de Windows. Una vez allí, debemos deshabilitar el "Adaptador virtual de red hospedada de Microsoft", y volverlo a habilitar (ver figura N.3).

5) Una vez iniciada la red, vamos a comprobar que la red creada es visible y accesible desde otros equipos, como otro ordenador, una tableta, o un teléfono móvil. La figura N.4 muestra las redes detectadas por mi teléfono móvil y por otro ordenador distinto al que aloja la red hospedada. Como vemos, ambos dispositivos detectan la red "MiRedAdHoc". Probamos a conectarnos introduciendo la contraseña "123456789", y si lo conseguimos, es que todo ha ido bien.

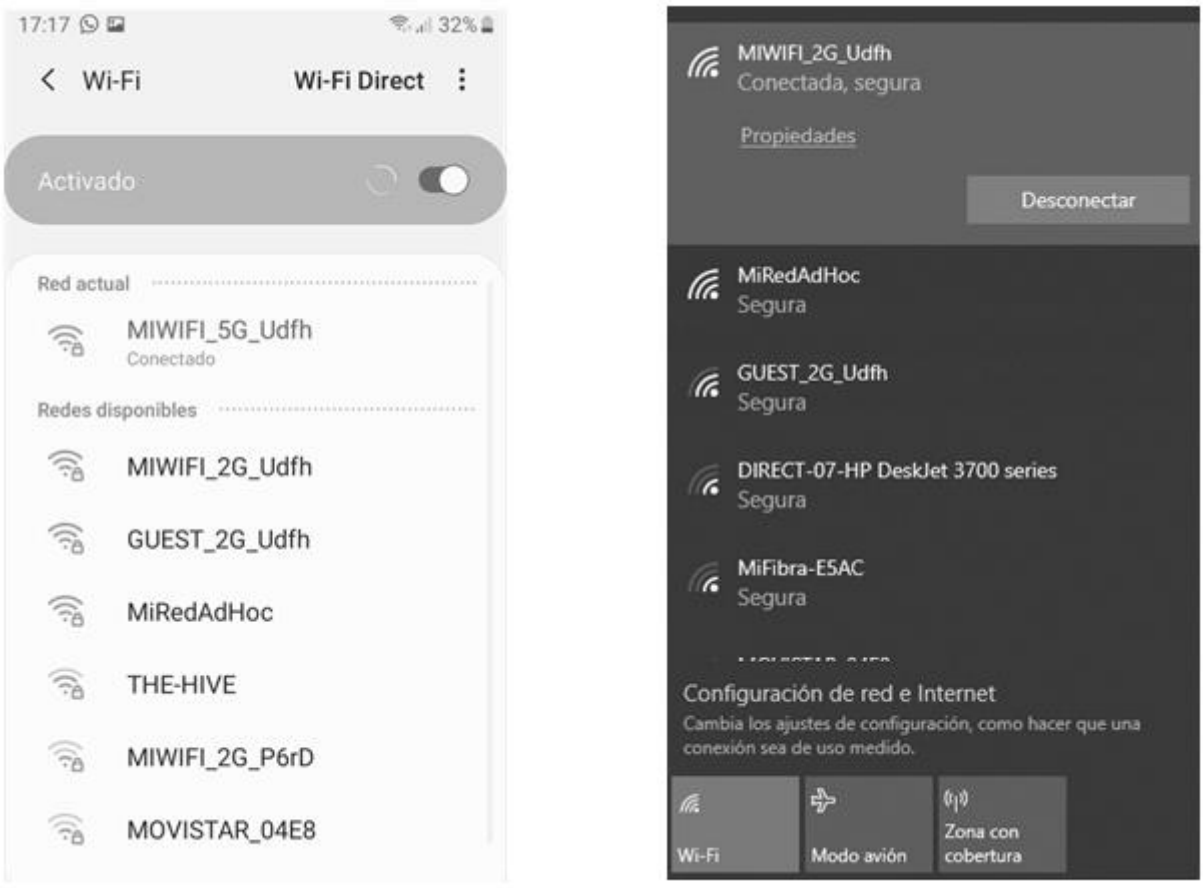


Figura N.4. La red hospedada recién creada es visible y accesible desde otros dispositivos.

6) Por supuesto, la red ad – hoc recién creada no proporciona acceso a Internet, sino únicamente conectividad entre los distintos dispositivos asociados a ella. Si queremos que nuestro ordenador comparta su conexión a Internet con el resto de dispositivos asociados a la red ad – hoc, debemos seguir estos pasos:

- Acudimos a la ventana de Configuración, y pulsamos en la opción "Red e Internet".
- En la sección de "Estado", pulsamos en la opción "Cambiar opciones del adaptador".
- Pinchamos con el botón derecho del ratón sobre la red que nos proporciona acceso a Internet, y elegimos "Propiedades".
- En la pestaña de uso compartido, activamos la casilla "Permitir que los usuarios de otras redes se conecten a través de la conexión de Internet de este equipo". Además, en la lista desplegable "Conexión de red doméstica", seleccionamos la red ad – hoc creada (ver figura N.5)
- En el botón "Configuración", seleccionamos los servicios a los que permitimos acceder a los usuarios de la red.
- Pulsamos en "Aceptar".

Si probamos a reconectar un dispositivo a la red "MiRedAdHoc", veremos que ahora dispone de acceso a Internet (a través de la conexión del ordenador que aloja la red hospedada).

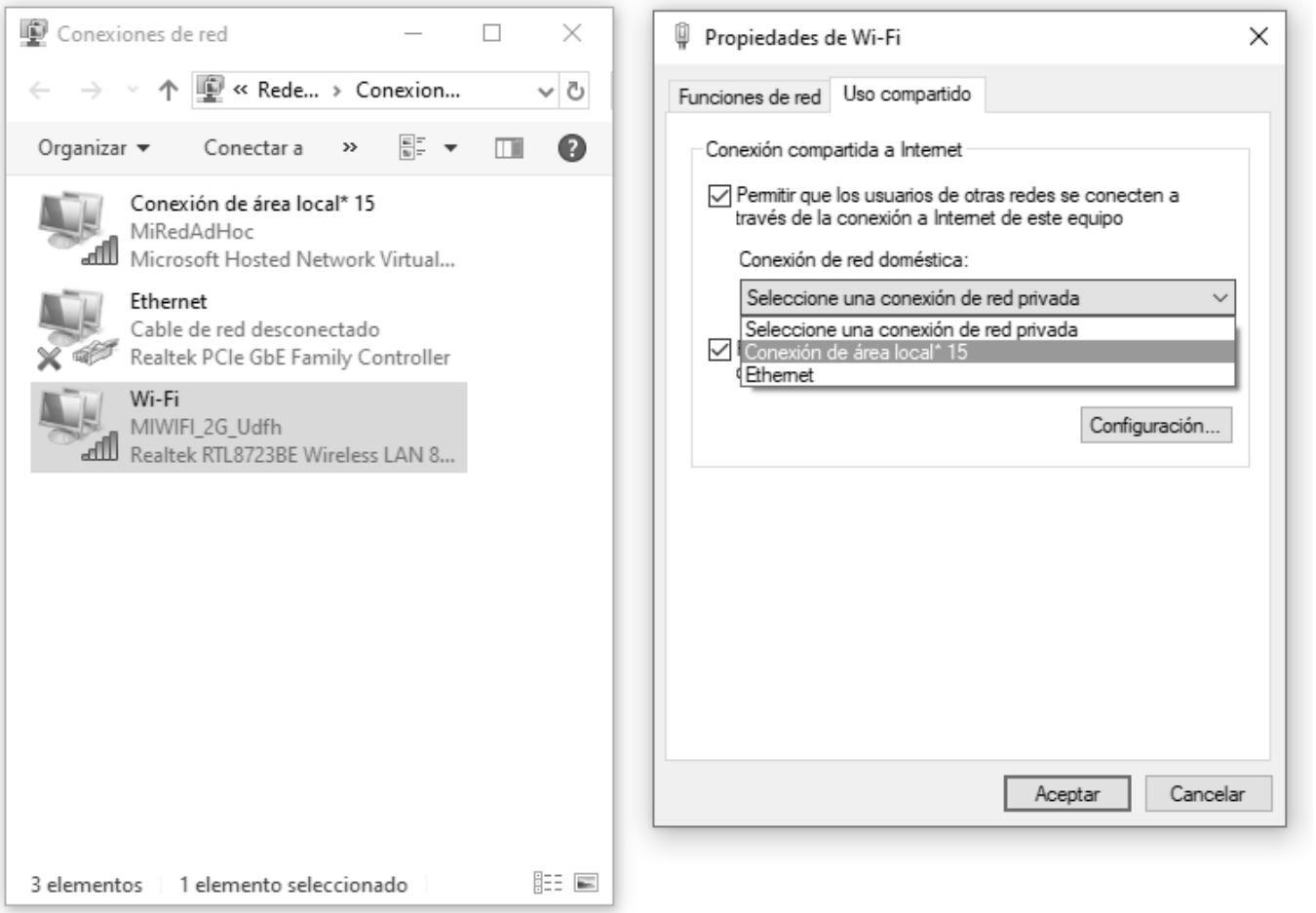


Figura N.5. Compartir la conexión a Internet.

7) Finalmente, si queremos desactivar la red ad – hoc que hemos creado, ejecutamos el comando `netsh wlan stop hostednetwork`, ver figura N.6.

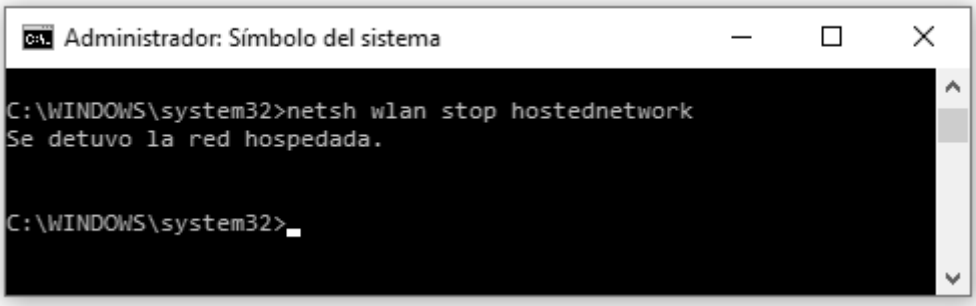


Figura N.6. Desactivar la red hospedada.

N.2. RED AD – HOC USANDO UNA APLICACIÓN DE USUARIO.

En caso de no poder levantar una red hospedada mediante la línea de comandos, existen multitud de aplicaciones que nos permiten crear un punto de acceso WiFi en nuestro ordenador. Algunos ejemplos son "NoWiFi" (disponible de forma gratuita en la tienda Microsoft Store), y "Connectify Hotspot", ver figura N.7.

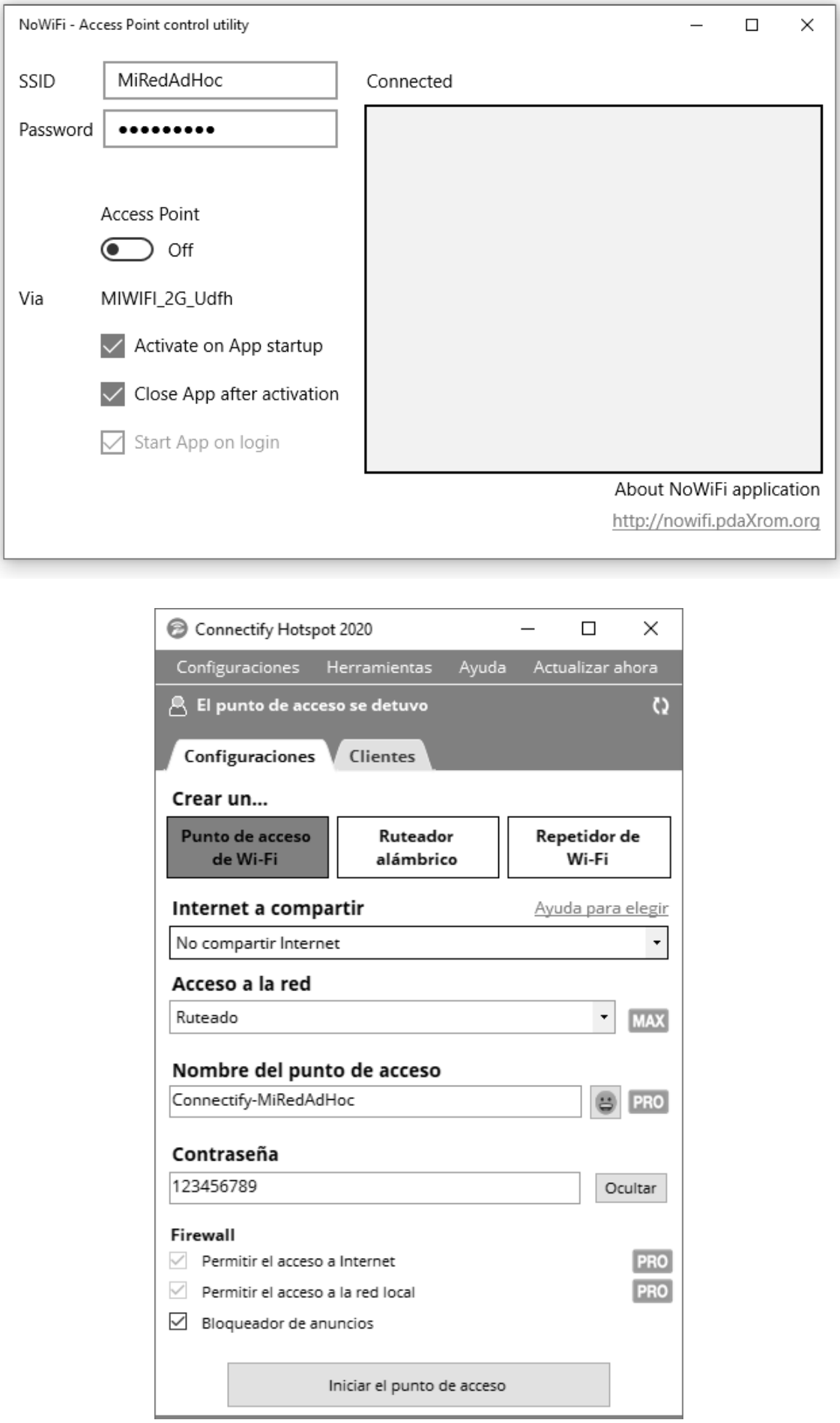


Figura N.7. Ejemplos de programas para crear hot – spots WiFi.

PRÁCTICA O. PROGRAMACIÓN DE CSMA/CA CON PYTHON.

O.1. PLANTEAMIENTO DEL PROBLEMA.

En esta práctica vamos a escribir un programa en Python que implemente una versión simplificada del método de acceso CSMA/CA del protocolo 802.11, ver sección 6.3 del capítulo previo. En este algoritmo simplificado, el ordenador que desee enviar procederá de la siguiente forma:

1) Cuando el ordenador emisor tiene datos para transmitir, inicializa el contador B de número de intentos a cero, y a continuación, comprueba si el canal está libre. En nuestro caso, el propio programa simulará la ocupación del canal decidiendo aleatoriamente si el canal está disponible (con una probabilidad de un 25% de encontrarlo libre, y una probabilidad del 75% de hallarlo ocupado). Esta parte es similar al algoritmo CSMA que ya implementamos en la práctica M. Si el canal está ocupado, la estación vuelve a comprobarlo otra vez, hasta encontrarlo libre. Si el canal está libre, la estación envía una trata RTS (Request To Send).
2) Tras enviar la trama RTS, la estación emisora queda a la espera de recibir una trama CTS (Clear To Send) de la estación destino. En nuestro caso, el programa decidirá con una probabilidad del 50% si la trama CTS llega o no a la estación emisora antes de que venza un temporizador de espera. Si la estación emisora no recibe la trama CTS, incrementa el valor de B en una unidad y se pone a escuchar nuevamente el canal para intentar transmitir de nuevo. Si la estación emisora recibe la trama CTS, entonces envía una trama de datos, y se prepara para recibir la trama de acuse de recibo ACK (Acknowledgement).
3) La trama ACK llegará a la estación emisora el 75% de las veces. Si recibe la trama ACK, la estación emisora considera que la trama de datos se envió correctamente, y el programa termina. Si la estación emisora no recibe el ACK, incrementa el valor del contador B y se pone a escuchar nuevamente el canal para intentar transmitir de nuevo.
4) Si el valor del contador B supera un cierto valor máximo (elegido por el usuario al principio del programa), la estación emisora deja de intentar transmitir de nuevo, y aborta la transmisión.

A continuación mostramos varios ejemplos de ejecución del programa:

a) Transmisión exitosa:

```
Simulación de CSMA/CA en una WLAN 802.11.
Por favor, indique el número máx. de intentos de transmisión (o pulse ENTER
para 3):

La estación de origen tiene datos para enviar.
Intento número 0.
Escuchando canal...
Canal ocupado.
Canal ocupado.
Canal ocupado.
Canal ocupado.
Canal libre.
Enviando trama RTS...
Trama CTS recibida!
Enviando trama de DATOS...
Temporizador agotado: Trama ACK no recibida.
Intento número 1.
Escuchando canal...
Canal ocupado.
```

```
Canal ocupado.
Canal ocupado.
Canal libre.
Enviando trama RTS...
Temporizador agotado: Trama CTS no recibida.
Intento número 2.
Escuchando canal...
Canal libre.
Enviando trama RTS...
Trama CTS recibida!
Enviando trama de DATOS...
Trama ACK recibida!
Datos enviados correctamente.
```

b) Transmisión fallida (1):

```
Simulación de CSMA/CA en una WLAN 802.11.
Por favor, indique el número máx. de intentos de transmisión (o pulse ENTER
para 3):

La estación de origen tiene datos para enviar.
Intento número 0.
Escuchando canal...
Canal ocupado.
Canal ocupado.
Canal libre.
Enviando trama RTS...
Temporizador agotado: Trama CTS no recibida.
Intento número 1.
Escuchando canal...
Canal libre.
Enviando trama RTS...
Temporizador agotado: Trama CTS no recibida.
Intento número 2.
Escuchando canal...
Canal ocupado.
Canal libre.
Enviando trama RTS...
Temporizador agotado: Trama CTS no recibida.
Intento número 3.
Número máximo de intentos de transmisión alcanzado: Transmisión abortada.
```

c) Transmisión fallida (2):

```
Simulación de CSMA/CA en una WLAN 802.11.
Por favor, indique el número máx. de intentos de transmisión (o pulse ENTER
para 3):

La estación de origen tiene datos para enviar.
Intento número 0.
Escuchando canal...
Canal ocupado.
Canal libre.
Enviando trama RTS...
Temporizador agotado: Trama CTS no recibida.
Intento número 1.
Escuchando canal...
Canal libre.
Enviando trama RTS...
Temporizador agotado: Trama CTS no recibida.
Intento número 2.
```

```
Escuchando canal...
Canal ocupado.
Canal ocupado.
Canal ocupado.
Canal libre.
Enviando trama RTS...
Trama CTS recibida!
Enviando trama de DATOS...
Temporizador agotado: Trama ACK no recibida.
Intento número 3.
Número máximo de intentos de transmisión alcanzado: Transmisión abortada.
```

O.2. IMPORTAR MÓDULOS Y EMPEZAR EL PROGRAMA PRINCIPAL.

En este programa necesitaremos importar los módulos `random` (para obtener enteros aleatorios) y `sys` (para terminar el programa con la función `sys.exit()`).

En el programa principal, comenzamos indicando qué hace el programa (`Simulación de CSMA/CA en una WLAN 802.11.`).

A continuación, le permitimos al usuario elegir el número máximo de intentos de transmisión. El usuario podrá responder con un entero, o simplemente presionar INTRO para elegir un valor por defecto de 3 (`Por favor, indique el número máx. de intentos de transmisión (o pulse ENTER para 3):`).

Una vez elegido el número máximo de intentos (y almacenado en una variable), informamos de que la estación de origen tiene datos para enviar (`La estación de origen tiene datos para enviar.`), y después inicializamos el contador de intentos B a cero.

Tras haber hecho todos estos pasos iniciales, el programa entra en un bucle que se repetirá mientras el número de intentos de transmisión no supere el número máximo. A cada pasada del bucle, el programa informa en qué intento está actualmente (`Intento número ?`), llama a la función `escucharCanal_enviarRTS()` para escuchar el canal y enviar un RTS cuando lo encuentre libre, y finalmente, llama a la función `B = recibirCTS_enviarDATOS (B)` para esperar la llegada de la trama CTS y enviar una trama de datos cuando la reciba.

El programa sale del bucle cuando el número de intentos supera el número máximo, momento en el que el programa aborta la transmisión, informa de este hecho (`Intento número 3. Número máximo de intentos de transmisión alcanzado: Transmisión abortada.`), y termina.

A continuación, vamos a programar las funciones que necesita el programa para funcionar.

O.3. FUNCIÓN ESCUCHARCANAL_ENVIARRTS().

La función `escucharCanal_enviarRTS()` implementa el procedimiento de escucha del canal (CSMA) que ya escribimos en la práctica M. Lo primero que hace es indicar que la estación va a comprobar el estado del canal (`Escuchando canal...`). A continuación, la función obtiene un número aleatorio entre 1 y 100. (Con este número aleatorio simularemos la probabilidad de encontrarlo libre u ocupado). Mientras el número aleatorio sea mayor o igual que 25 (esto es, el 75% de las veces), la función indicará que el canal está ocupado (`Canal ocupado.`), y volverá a obtener un nuevo número aleatorio. En el momento en el que el número aleatorio sea menor que 25 (el 25% de las veces), la función indicará que el canal se encuentra libre (`Canal libre.`), y que la estación envía la trama RTS (`Enviando trama RTS...`).

El esquelelo de la función es el siguiente:

```
def escucharCanal_enviarRTS():
    # Rellena aquí el código de la función
```

O.4. RECIBIRCTS_ENVIARDATOS ().

Ahora vamos a escribir la función `recibirCTS_enciarDATOS()` que pone a la estación emisora en espera de recibir una trama CTS desde la estación destino. Esta función recibirá como argumento el número actual de intentos (B), y devolverá el número de intentos actualizado.

Para empezar, la función obtiene un número aleatorio entre 0 y 1. Si el número aleatorio obtenido es 0, asumiremos que la trama CTS no llegó antes de agotarse un temporizador. En ese caso, la estación informa de este hecho (`Temporizador agotado: Trama CTS no recibida.`), e incrementa el número de intentos B en una unidad.

Por el contrario, si el número aleatorio es 1 supondremos que la trama CTS se recibió correctamente. En ese caso, la estación informa de este hecho (`Trama CTS recibida!`), indica que pasa a enviar una trama de datos (`Enviando trama de DATOS...`), y llama a la función `B = recibirACK(B)` que pone a la estación emisora en espera de recibir la trama de confirmación ACK.

Por último, la función termina devolviendo el valor actualizado del número de intentos.

```
def recibirCTS_enciarDATOS (B):
    # Rellena aquí el código de la función
```

O.5. FUNCIÓN RECIBIRACK().

La función `recibirACK()`, a la que se llega desde la función `recibirCTS_enciarDATOS()`, pone a la estación emisora en espera de recibir una trama ACK tras haber enviado una trama de datos. Esta función recibirá como argumento el número actual de intentos (B), y devolverá el número de intentos actualizado.

La función empieza obteniendo un número aleatorio entre 1 y 100. (Con este número aleatorio simularemos la recepción o no de la trama ACK). Si el número aleatorio es mayor o igual que 25 (esto es, el 75% de las veces), la función indicará la llegada de la trama ACK (`Trama ACK recibida!`), e informará de que la trama de datos se recibió correctamente en el destino (`Datos enviados correctamente.`) Por el contrario, si el número aleatorio es menor que 25 (el 25% de las veces), la función considerará que la trama ACK no se recibió dentro del plazo de espera (`Temporizador agotado: Trama ACK no recibida.`), incrementará el número de intentos B en una unidad, y devolverá el número de intentos actualizado.

```
def recibirACK(B):
    # Rellena aquí el código de la función
```

Con esto el programa está terminado y debería funcionar correctamente.

5. PROGRAMACIÓN DE APLICACIONES CLIENTE – SERVIDOR.

5.1. MODELO CLIENTE – SERVIDOR.

La **capa de aplicación** se encarga de dar servicio al usuario final de Internet. La comunicación entre las capas de aplicación en el host origen y en el host destino utiliza una *conexión lógica*, lo que significa que las dos capas de aplicación asumen que hay una conexión directa imaginaria a través de la cual pueden enviar y recibir los mensajes que desean intercambiar.

Para poder usar los servicios de Internet se necesitan dos programas de aplicación que interactúen mutuamente, cada uno de ellos ejecutándose en un ordenador ubicado en algún lugar del mundo. Sin embargo, la comunicación entre estos dos programas puede llevarse a cabo en base a dos modelos bien distintos: El tradicional modelo cliente – servidor y el nuevo modelo de igual a igual (peer to peer).

En el **modelo cliente – servidor**, el proveedor del servicio es un programa de aplicación al que se denomina **proceso servidor**. El proceso servidor se está ejecutando ininterrumpidamente, a la espera de que otro programa de aplicación, llamado **proceso cliente**, establezca una conexión a través de Internet para solicitar un servicio.

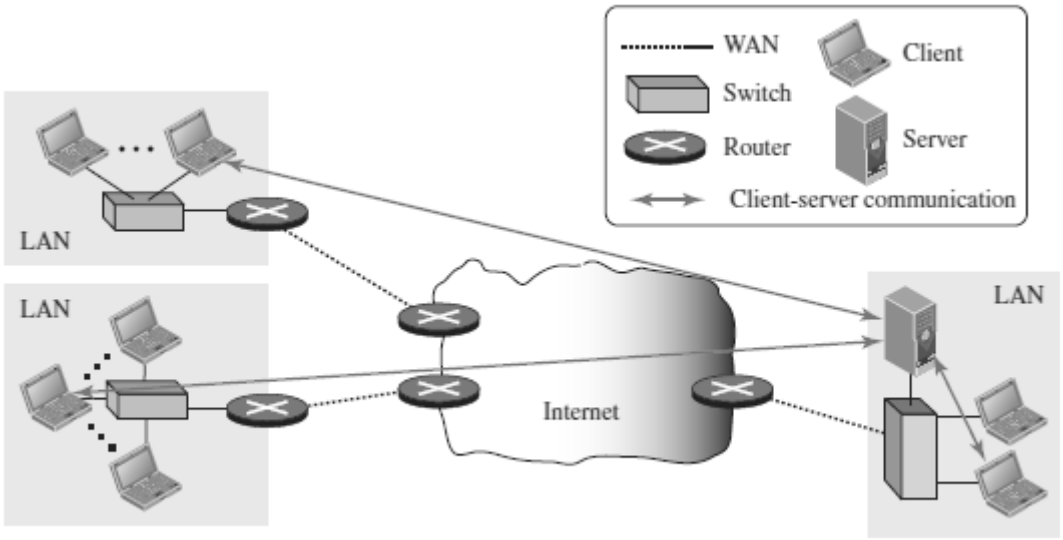


Figura 5.1. Modelo cliente – servidor.

Aunque la comunicación en el modelo cliente – servidor se produce entre dos procesos de aplicación, el cometido de cada uno de ellos es totalmente distinto. En otras palabras, no es posible ejecutar un programa servidor como si fuese un programa cliente, y viceversa. Más adelante, cuando hablemos sobre la programación de clientes y servidores, veremos que siempre se necesita escribir dos programas para cada tipo de servicio: Un programa cliente, y un programa servidor. La figura 5.1 muestra un ejemplo de comunicación cliente – servidor en el que tres clientes solicitan servicios a un servidor.

Una de las mayores desventajas de este modelo es que un solo servidor debe ser capaz de proveer servicio a muchos clientes, lo que implica que el servidor ha de ser un ordenador muy potente. Incuso un ordenador de mucha potencia puede verse desbordado si muchos clientes intentan conectarse con él al mismo tiempo.

Algunos de los servicios más tradicionales de Internet usan este modelo de comunicación, incluyendo la World Wide Web (WWW) y su protocolo HTTP (HyperText Transfer Protocol), el protocolo de transferencia de archivos FTP (File Transfer Protocol), el correo electrónico, etc.

En los últimos años ha surgido el nuevo **modelo de igual a igual** (P2P = *peer to peer*), ideado para responder a las necesidades de algunas aplicaciones emergentes. En este modelo no se necesita un proceso servidor que esté ejecutándose todo el tiempo en espera de las solicitudes de los clientes. Esta responsabilidad se comparte entre todos los equipos. En un momento dado, un ordenador conectado a Internet podría dar servicio a otro ordenador, y en otro momento, recibir servicio de un tercero. Ese mismo ordenador incluso podría proporcionar y recibir servicio al mismo tiempo.

Aunque el modelo de igual a igual ha demostrado ser fácilmente escalable y muy eficiente en cuanto a costes (ya que elimina la necesidad de tener costosos servidores funcionando todo el tiempo), también presenta algunas dificultades. La principal tiene que ver con la seguridad, ya que es mucho más difícil establecer comunicaciones seguras en este tipo de sistemas distribuidos que en los sistemas controlados por un servidor. Otra desventaja es que no todas las aplicaciones se adaptan bien a este modelo. Por ejemplo, no todos los usuarios estarían dispuestos a participar en un modelo de igual a igual si algún día se decidiese que la Web debe implementarse de esta forma. Sin embargo, hay algunas aplicaciones como BitTorrent, Skype, IPTV, y la telefonía por Internet que sí que empelan este modelo.

5.2. PROGRAMACIÓN CLIENTE – SERVIDOR: SOCKETS.

En el modelo cliente – servidor, la comunicación en la capa de aplicación se produce entre dos programas en ejecución denominados **procesos**. Estos dos procesos son el **cliente** y el **servidor**. El cliente es un programa de aplicación que inicia la comunicación enviando una solicitud; el servidor es otro programa de aplicación que espera la solicitud del cliente. El servidor gestiona la solicitud del cliente, prepara un resultado, y se lo envía de vuelta al cliente. Esta definición implica que el servidor debe de estar ejecutándose cuando recibe la solicitud del cliente. Por el contrario, el cliente solo debe ejecutarse cuando lo necesita.

Si dos ordenadores están conectados el uno al otro mediante este modelo, uno de ellos debe estar ejecutando el proceso cliente, y el otro el proceso servidor. Sin embargo, debemos tener la precaución de arrancar el proceso servidor antes que el proceso cliente. El ciclo de operación del servidor es infinito: Después de arrancar debe estar ejecutándose todo el rato, a la espera de las solicitudes de los clientes. Por el contrario, el ciclo de operación del cliente es finito: Normalmente envía un número limitado de solicitudes al servidor correspondiente, recibe las respuestas, y termina.

INTERFAZ SOCKET.

¿Cómo puede un proceso cliente comunicarse con un proceso servidor? Bueno, recordemos que un programa no es más que un conjunto de instrucciones que le dicen a un ordenador lo que tiene que hacer. Los programas se escriben usando instrucciones para efectuar operaciones matemáticas, para manipular cadenas, para realizar accesos de entrada y/o salida, etc. Si queremos escribir un programa que pueda comunicarse con otro programa ejecutándose en otra máquina, necesitamos un nuevo conjunto de instrucciones que nos permita decirle a la capa de transporte que abra la conexión, que envíe y reciba datos del otro extremo, y que cierre la conexión. A este conjunto de instrucciones se le suele llamar **interfaz**. En el mercado existen distintas interfaces de comunicación (Transport Layer Interface (TLI), STREAM, etc.), pero aquí trabajaremos con la **interfaz socket**.

Para entender el concepto de interfaz socket, primero debemos considerar la relación entre el sistema operativo instalado en la máquina, y la pila de protocolos TCP/IP (un asunto que hemos estado ignorando hasta ahora). La figura 5.2 muestra esta relación.

El conjunto de instrucciones que constituye la interfaz socket se localiza entre el sistema operativo y la capa de aplicación. Para que una aplicación pueda acceder a los servicios prestados por la pila de protocolos TCP/IP necesita usar las instrucciones definidas en el interfaz socket.

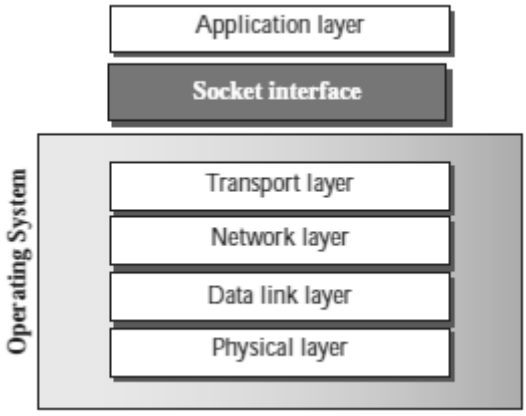


Figura 5.2. Relación entre el sistema operativo y los protocolos de la pila TCP/IP.

¿Qué es un socket?

Un **socket** (conector o toma) es una abstracción software que simula las tomas físicas que usamos en nuestra vida cotidiana (por ejemplo, la toma telefónica a la que conectamos un teléfono fijo). Para poder usar el canal de comunicación, el programa de aplicación (ya sea un cliente o un servidor) debe pedirle al sistema operativo que cree un socket. Una vez creado, el programa puede enchufarse a ese socket para enviar y recibir datos. Para que pueda haber transmisión de datos, cada extremo de la comunicación necesita crear su propio socket. La figura 5.3 ilustra esta abstracción mediante las tomas y los enchufes que usamos habitualmente. En la práctica, un socket es en realidad una estructura de datos software, tal y como veremos a continuación.

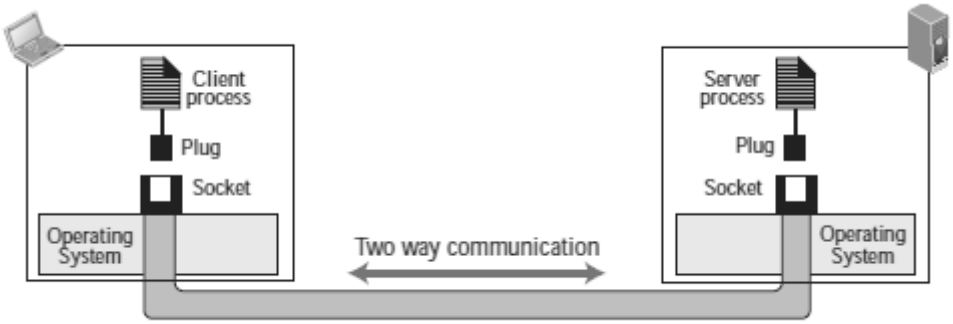


Figura 5.3. Concepto de socket.

SOCKETS EN PYTHON.

Módulo socket.

El formato de la estructura de datos que implementa un socket depende del lenguaje de programación usado para escribir los procesos de aplicación. En este capítulo usaremos el lenguaje Python para escribir nuestros programas.

Para usar sockets en Python, lo primero que debemos hacer es importar el módulo `socket`:

```
import socket
```

Este módulo contiene todas las funciones básicas para escribir procesos de aplicación que puedan comunicarse entre sí.

Funciones.

Los procesos de aplicación interactúan con su sistema operativo mediante un conjunto de funciones predeterminadas, que pasamos a listar a continuación. En secciones posteriores veremos cómo utilizarlas para escribir los programas cliente y servidor.

1) Función `socket()`. El sistema operativo define la estructura de un socket, pero no lo crea a menos que se así se lo indique el programa de aplicación. Tanto el proceso cliente y como el proceso servidor utilizan la función `socket()` para ordenar a los sistemas operativos de sus máquinas que creen sus respectivos sockets:

```
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

   La llamada a la función `socket()` necesita recibir dos parámetros: El primer parámetro sirve para indicar el esquema de direccionamiento a utilizar. `AF_INET` especifica que el direccionamiento es IPv4, mientras que `AF_INET6` indica que el direccionamiento es IPv6. El segundo parámetro se usa para especificar el tipo de servicio solicitado a la capa de transporte. `SOCK_STREAM` sirve para usar un servicio fiable orientado a conexión (protocolo TCP), mientras que `SOCK_DGRAM` especifica un servicio no fiable sin conexión (protocolo UDP).

2) Función `gethostname()`. Esta función le permite obtener el nombre de host de la máquina en la que se está ejecutando el programa de aplicación:

```
client_name = socket.gethostname()

server_name = socket.gethostname()
```

3) Función `gethostbyname()`. Con esta función se puede obtener la dirección IP de una máquina a partir de su nombre de host. Esta función suele usarse conjuntamente con la función `gethostname()`.

```
client_ip = socket.gethostbyname(client_name)

server_ip = socket.gethostbyname(server_name)
```

   Como veremos, estas dos funciones son muy útiles cuando queremos ejecutar los programas cliente y servidor en una misma máquina.

4) Función `bind()`. La función `socket()` le permite al proceso servidor crear un socket, pero no lo define de forma completa. Además, el proceso servidor necesita llamar a la función `bind()` para asignar el socket al host servidor y a un número de puerto:

```
server_socket.bind((server_ip, server_port))
```

   Al contrario que las funciones previas, la función `bind()` se llama sobre el socket del servidor, y no sobre el módulo `socket`. Todas las demás funciones que veremos a continuación también se llaman de esta forma.

   La función `bind()` recibe una tupla con la dirección IP y el número de puerto del servidor. De esta forma, cuando un cliente envía un paquete a la dirección IP y al puerto del servidor, el paquete se redirige automáticamente al socket del servidor.

5) Función `connect()`. Cuando un cliente quiere enviar o recibir datos de un servidor usando un socket TCP (esto es, orientado a conexión), primero se necesita establecer una conexión entre el cliente y el servidor. La función `connect()` le permite al cliente establecer una conexión TCP con el servidor:

```
client_socket.connect((server_ip, server_port))
```

La función `connect()` necesita recibir una tupla con la dirección IP y con el número de puerto del servidor. Después de haberse ejecutado esta función, la capa de transporte realiza la negociación en tres fases (three - way handshake) que permite establecer la conexión TCP entre el cliente y el servidor.

6) Función `listen()`. Esta función pone a un servidor TCP a escuchar las solicitudes de conexión enviadas desde un cliente:

```
server_socket.listen(1)
```

El parámetro especifica el número máximo de solicitudes que el servidor pondrá en espera si ya está atendiendo otra solicitud de conexión.

7) Función `accept()`. Cuando un proceso servidor recibe una solicitud de conexión TCP de un cliente, invoca la función `accept()` sobre el socket servidor (`server_socket`) para crear un nuevo socket dedicado a ese cliente en particular. A continuación, el cliente y el servidor completan la negociación y crean una conexión TCP entre el socket del cliente (`client_socket`) y ese nuevo socket en el servidor (`conn_socket`). Una vez establecida, tanto el cliente como el servidor pueden enviar y recibir datos a través de la conexión.

```
conn_socket, addr = server_socket.accept()
```

Notar que la llamada a dicha función devuelve el nuevo socket creado en el servidor (`conn_socket`), así como la dirección del cliente (`addr`) con el que se ha establecido la conexión.

8) Función `sendto()`. La función `sendto()` le permite a un proceso (cliente o servidor) enviar datos a otro proceso remoto (servidor o cliente) usando los servicios del protocolo UDP.

```
client_socket.sendto(client_message,(server_ip, server_port))

server_socket.sendto(server_message,(client_ip, client_port))
```

Como UDP es un protocolo sin conexión, esta función necesita recibir como argumentos el mensaje a enviar, y la dirección de destino (esto es, la tupla formada por la dirección IP del host remoto y el número de puerto del proceso remoto). La dirección de origen también se adjunta al mensaje enviado, pero esto no hay que programarlo porque lo hace automáticamente el sistema operativo subyacente. Después de enviar el mensaje, el proceso emisor queda a la espera de recibir datos del proceso remoto.

9) Función `recvfrom()`. Esta función le permite a un proceso (cliente o servidor) recibir datos de otro proceso remoto (servidor o cliente) usando los servicios de UDP.

```
server_message, server_address = client_socket.recvfrom(2048)

client_message, client_address = server_socket.recvfrom(2048)
```

Tomemos como ejemplo la función `server_message, server_address = client_socket.recvfrom(2048)`. Cuando el cliente recibe un paquete desde el servidor remoto a través de su socket (`client_socket`), ese paquete se guarda en la variable `server_message`, y la

dirección de origen en la variable `server_address`. La variable `server_address` es una tupla que contiene la dirección IP y el número de puerto del servidor. La función `recvfrom()` recibe como argumento el tamaño máximo del búfer que almacenará los datos recibidos (en este caso, 2048 bytes).

10) Función `send()`. La función `send()` le permite a un proceso (cliente o servidor) enviar datos a otro proceso remoto (servidor o cliente) usando el protocolo TCP:

```
client_socket.send(client_message)

conn_socket.send(server_message)
```

La función `send()` asume que ya hay una conexión TCP establecida entre el proceso origen y el proceso destino, por lo que no necesita como parámetro la dirección de destino. Esta función simplemente toma el mensaje a enviar, y pone los bytes en la tubería TCP que conecta con el proceso remoto. A continuación, el proceso emisor queda a la espera de recibir bytes del proceso remoto.

11) Función `recv()`. La función `recv()` le permite a un proceso (cliente o servidor) enviar datos a otro proceso remoto (servidor o cliente) usando el protocolo TCP:

```
server_message = client_socket.recv(2048)

client_message = conn_socket.recv(2048)
```

Tomemos como ejemplo la función `server_sentence = client_socket.recv(2048)`. Cuando el cliente va recibiendo los datos enviados por el servidor remoto, estos se van alojando alojan en la cadena `server_sentence`.

12) Función `close()`. Esta función le permite a un proceso (cliente o servidor) cerrar su socket. Si el socket es de tipo TCP, esta función también cierra la conexión TCP entre cliente y servidor, lo que implica el envío de un mensaje TCP desde la capa de transporte de la máquina origen a la capa de transporte de la máquina remota.

```
client_socket.close()

conn_socket.close()
```

CLIENTE – SERVIDOR CON SOCKETS UDP.

En esta subsección vamos a escribir unos sencillos programas cliente y servidor que usen UDP. En la siguiente sección escribiremos unos programas similares que usen TCP.

Recordemos que dos procesos ejecutándose en dos máquinas distintas pueden enviarse mensajes entre sí a través de sus respectivos sockets. En este caso, vamos a utilizar sockets UDP. Como UDP es un protocolo sin conexión, el proceso emisor debe adjuntar la dirección de destino al paquete que desea enviar. Después de que el paquete pase a través del socket del proceso emisor, Internet usará esta dirección de destino para encaminar el paquete al socket del proceso receptor. Cuando el paquete llega al proceso receptor a través de su socket, este proceso recupera el paquete y lo inspecciona para extraer su carga útil y realizar la operación apropiada sobre los datos recibidos.

Recordemos que esta dirección de destino incluye tanto la dirección IP de la máquina destino como el número de puerto del proceso de destino. La dirección IP permite encaminar el paquete a través de Internet al host de destino. Y como en el host de destino puede haber múltiples procesos en ejecución, el número de host permite entregar el paquete al proceso adecuado. (La dirección de origen también se añade

al paquete enviado, pero esto no lo hace el proceso emisor, sino que lo hace automáticamente el sistema operativo subyacente).

Tanto para UDP como para TCP, los programas cliente y servidor que vamos a escribir realizarán las siguientes tareas:

a) El cliente lee una línea de caracteres del teclado, y envía estos datos al servidor.
b) El servidor recibe los datos y convierte los caracteres a mayúsculas.
c) El servidor envía los datos modificados de vuelta al cliente.
d) El cliente recibe los datos modificados del servidor y los muestra por pantalla.

A continuación tenemos el código completo del proceso cliente. Abre un editor de archivos de Python, copia el código, y guárdalo como `Client_UDP.py`:

```
import socket
import time

server_name = socket.gethostname()
server_ip = socket.gethostbyname(server_name)
server_port = 12345

client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

text = input('Input lowercase message: ')
client_message = text.encode()
print ('Message sent to server.')

client_socket.sendto(client_message,(server_ip, server_port))

server_message, server_address = client_socket.recvfrom(2048)
data = server_message.decode()
print ('Message received from server: ' + str(data))

time.sleep(10)
client_socket.close()
```

El programa es muy sencillo. En primer lugar, importamos los módulos necesarios (en este caso, el módulo `socket` y el módulo `time`). En nuestro caso, vamos a ejecutar los procesos cliente y servidor en una misma máquina. Por esta razón, obtenemos el nombre de host del ordenador en la que se está ejecutando el cliente, y lo guardamos como nombre de host del servidor. A continuación obtenemos la dirección IP del servidor a partir de su nombre de host, e indicamos el puerto del servidor al que nos vamos a conectar (en este caso, el puerto 12345). Después creamos el socket del cliente, especificando que el esquema de direccionamiento es IPv4, y que el protocolo de capa de transporte es UDP. Seguidamente, solicitamos que el usuario escriba un texto en minúsculas, y usamos la función `encode()` para convertir esa cadena de caracteres en bytes. (La función `sendto()` envía bytes, y no cadenas de texto). Una vez convertidos los caracteres del mensaje en bytes, usamos la función `sendto()` para enviar esos bytes al servidor. Notar que, como UDP es un protocolo sin conexión, debemos proporcionar la dirección IP y el número de puerto del servidor. A continuación, el cliente utiliza la función `recvfrom()` para recibir los bytes devueltos desde el servidor. Una vez recibidos los bytes, el cliente utiliza la función `decode()` para convertir esos bytes en la cadena modificada por el servidor, que pasa a imprimir por pantalla. Finalmente, y tras una pausa de 10 segundos, el cliente cierra su socket y termina el programa.

Ahora vamos a escribir el programa del servidor. Abre un editor de archivos de Python, copia el código mostrado a continuación, y guárdalo como `Server_UDP.py`:

```
import socket

server_name = socket.gethostname()
server_ip = socket.gethostbyname(server_name)
server_port = 12345

server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind((server_ip, server_port))
print ('The server is ready to receive.')

while True:
    client_message, client_address = server_socket.recvfrom(2048)
    data = client_message.decode()
    print ('Message received from client: ' + str(data))
    modified_message = data.upper()
    print ('Message returned to client: ' + str(modified_message))
    server_message = modified_message.encode()
    server_socket.sendto(server_message, client_address)
```

¿Cómo funciona el proceso servidor UDP? Después de importar el módulo `socket`, el proceso obtiene el nombre de host del ordenador en el que se está ejecutando, y a partir de él, su dirección IP. A continuación, el servidor reserva el puerto 12345 para comunicarse con el cliente. Tras crear el socket UDP, el proceso asigna este socket al host servidor y al puerto del servidor, e indica por pantalla que está listo para recibir solicitudes de un cliente. Seguidamente, el servidor entra en un bucle infinito, donde recibe los bytes enviados por el cliente, decodifica los bytes para obtener la cadena del mensaje, muestra por pantalla el mansaje recibido, lo convierte a mayúsculas, imprime el mensaje modificado, codifica el mensaje para convertirlo en bytes, y envía esos bytes de vuelta al cliente.

Vamos a probar los programas. Recordar estamos usando el mismo ordenador para ejecutar los procesos cliente y servidor. Además, es importante enfatizar que siempre debemos ejecutar el programa servidor en primer lugar. Pinchamos en el icono del programa servidor (se abrirá una ventana de comandos), y a continuación, pinchamos en el icono del programa cliente (se abrirá una segunda ventana de comandos). El programa cliente nos pide que escribamos un texto en minúsculas. Lo escribimos y pulsamos ENTER. La salida de ambos programas debería ser similar a la mostrada en la figura:

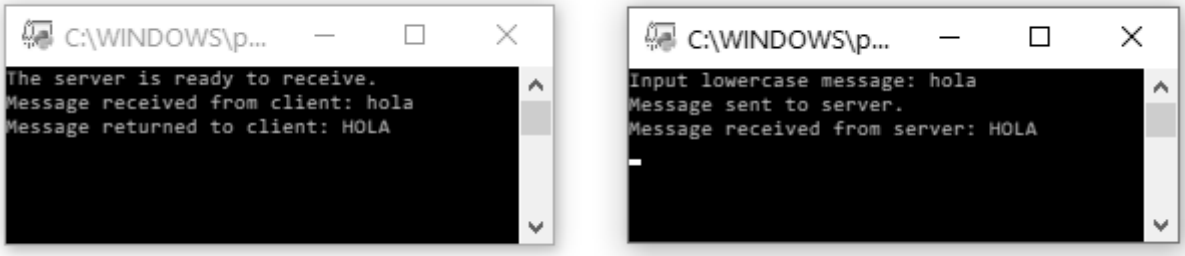


Figura 5.4. Salidas de los procesos servidor (izquierda) y cliente (derecha).

CLIENTE – SERVIDOR CON SOCKETS TCP.

Al contrario que UDP, TCP es un protocolo orientado a conexión. Esto significa que, antes de que el cliente y el servidor puedan empezar a enviarse datos, primero deben negociar y establecer una conexión TCP. Un extremo de la conexión TCP estará enchufado al socket del cliente, y el otro extremo al socket del servidor. Esta conexión TCP tiene asociada la dirección de socket del cliente (dirección IP y número de puerto) y la dirección de socket del servidor (dirección IP y número de puerto). Una vez establecida la conexión TCP, si un extremo quiere enviar datos al otro extremo, el emisor solo debe poner los datos en la

conexión TCP a través de su socket. Esto difiere de UDP, donde el emisor debía adjuntar al paquete la dirección de destino antes de enviarlo por el socket.

Ahora consideremos la interacción entre los programas cliente y servidor en TCP. El cliente es el encargado de contactar con el servidor, pero para que el servidor pueda reaccionar al contacto inicial del cliente, debe estar preparado con antelación. Esto implica dos cosas: En primer lugar, y como en UDP, el servidor TCP debe estar ejecutándose antes de que el cliente intente contactar con él. En segundo lugar, el programa servidor debe disponer de un socket especial para recibir el contacto inicial del proceso cliente.

Si el proceso servidor está en ejecución, el proceso cliente puede iniciar una conexión TCP con él. El proceso cliente hace esto creando un socket TCP. Cuando el cliente crea su socket TCP, también especifica la dirección del socket de bienvenida del servidor, a saber, la dirección IP del host servidor y el número de puerto del socket servidor. Después de crear su socket, el cliente inicia una negociación en tres fases y establece la conexión TCP con el servidor. Esta negociación en tres fases ocurre en la capa de transporte, y es completamente invisible para los programas cliente y servidor.

Durante la negociación, el cliente contacta con el socket de bienvenida del servidor. Cuando el servidor "escucha" la solicitud del cliente, crea un nuevo socket dedicado a la conexión con ese cliente en particular. En nuestro ejemplo, el socket de bienvenida se denomina `server_socket`, y el socket dedicado al cliente que establece la conexión se llama `conn_socket`. En ocasiones, los estudiantes suelen confundir el socket de bienvenida (que es el punto inicial de contacto para todos los clientes que desean comunicarse con el servidor) con los distintos sockets que se crean posteriormente para las comunicaciones con cada cliente.

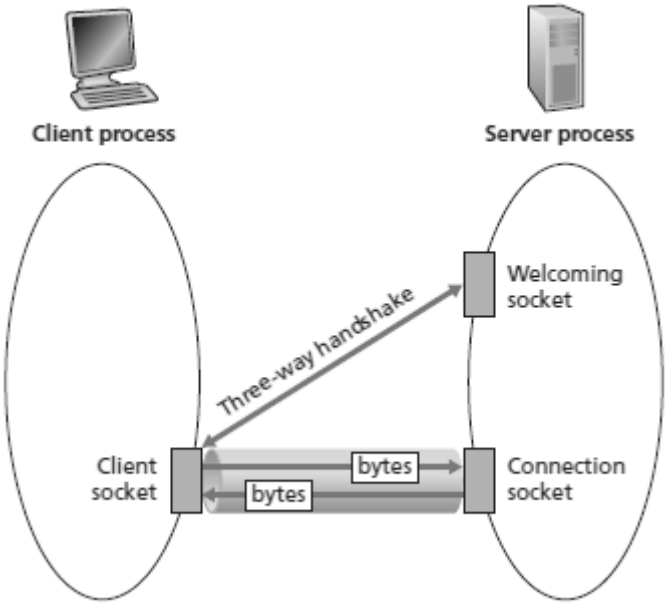


Figura 5.5. El proceso servidor TCP dispone de dos sockets.

Desde su punto de vista, la aplicación cree que el socket del cliente y el socket de conexión del servidor están directamente conectados mediante una tubería TCP. Como muestra la figura 5.5, el proceso cliente puede enviar bytes a través de su socket, y el protocolo TCP garantiza que el proceso servidor recibirá (a través de su socket de conexión) cada uno de los bytes enviados en el mismo orden en el que se enviaron. Es decir, TCP proporciona un servicio fiable entre los procesos cliente y servidor. (Esto difiere de UDP, que solo proporciona un servicio no fiable en el que los bytes pueden llegar desordenados, o incluso perderse). Y lo que es más, el cliente no solo puede enviar bytes a través de su socket, sino que también recibirlos. Análogamente, el proceso servidor no solo puede recibir bytes a través de su socket de conexión, sino que también enviarlos.

A continuación vamos a programar la misma aplicación cliente – servidor que ya escribimos en UDP: El cliente envía una línea de datos al servidor, el servidor convierte la línea a mayúsculas, y se la envía de vuelta al cliente.

Abajo tenemos el código completo del proceso cliente. Abre un editor de archivos de Python, copia el código, y guárdalo como `Client_TCP.py`:

```
import socket
import time

server_name = socket.gethostname()
server_ip = socket.gethostbyname(server_name)
server_port = 12000

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

client_socket.connect((server_ip, server_port))

text = input('Input lowercase sentence: ')
client_message = text.encode()
client_socket.send(client_message)

data = client_socket.recv(1024)
server_message = data.decode()
print ('Message received from server: ' + str(server_message))

time.sleep(10)
client_socket.close()
```

¿Cómo funciona este programa? En primer lugar, importamos los módulos necesarios. De nuevo, vamos a ejecutar los procesos cliente y servidor en el mismo ordenador (a saber, el ordenador local). Esta es la razón por la que obtenemos el nombre de host del servidor consultando el nombre de host del ordenador en el que se ejecuta el proceso cliente. A partir del nombre de host, obtenemos la dirección IP del servidor. Por último, especificamos el puerto del servidor al que nos conectaremos (en este caso, el puerto 12000).

A continuación creamos un socket TCP para el cliente, y lo usamos para mandar una solicitud de conexión a la dirección del socket del servidor. Seguidamente le pedimos al usuario cliente que escriba una línea de datos en minúsculas, que posteriormente convertimos en un flujo de bytes para poder enviársela al servidor. Notar que la función `send()` no requiere la dirección del socket servidor, porque en TCP el cliente simplemente necesita poner los bytes en la tubería TCP que le conecta con el servidor.

Tras enviar los bytes al servidor, el cliente se queda esperando hasta recibir los bytes devueltos por el servidor. Cuando los recibe, los decodifica para obtener el mensaje modificado, que pasa a imprimir por pantalla. Finalmente, y tras una pausa de 10 segundo, el cliente cierra su socket (y por consiguiente, la conexión TCP con el servidor).

Ahora vamos a escribir el programa del servidor. Abre un editor de archivos de Python, copia el código mostrado a continuación, y guárdalo como `Server_TCP.py`:

```
import socket

server_name = socket.gethostname()
server_ip = socket.gethostbyname(server_name)
server_port = 12000
```

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((server_ip, server_port))

server_socket.listen(4)
print ('The server is ready to receive.')

while True:
    conn_socket, addr = server_socket.accept()
    print ('TCP Connection established with client: ' + str(addr))
    client_message = conn_socket.recv(1024)
    data  = client_message.decode()
    print ('Message received from client: ' + str(data))
    modified_message = data.upper()
    print ('Message returned to client: ' + str(modified_message))
    server_message = modified_message.encode()
    conn_socket.send(server_message)
    conn_socket.close()
```

El programa comienza obteniendo el nombre de host del servidor (que coincide con el nombre de host de la máquina donde se ejecutarán los procesos cliente y servidor). A partir del nombre de host obtiene la dirección IP, y después, reserva el puerto 12000 para recibir las solicitudes de conexión de los clientes.

Seguidamente, el servidor crea un socket TCP de bienvenida, que utilizará para recibir todas las solicitudes de conexión de los clientes. Recordemos que, para definir completamente el socket, el servidor debe usar la función `bind()` para asignar a ese socket su dirección IP y su número de puerto. A continuación, el servidor se pone en modo escucha. Notar que, en este caso, estamos permitiendo que 4 solicitudes queden en cola si el servidor ya está ocupado atendiendo una solicitud. En este punto, el servidor ya está listo para recibir las solicitudes de los clientes a través de su socket de bienvenida. Una vez preparado, el servidor entra en un bucle infinito para aceptar las solicitudes de conexión de los clientes (momento en el que se crean los sockets de conexión con cada uno de los clientes), recibir los datos enviados por los clientes, convertirlos a mayúsculas, enviar los datos modificados de vuelta a los clientes a través del socket de conexión correspondiente, y cerrar las distintas conexiones.

Vamos a probar los programas. Ejecutamos el proceso servidor en primer lugar, y después, ejecutamos hasta cinco procesos cliente. La salida debería ser similar a la mostrada en la figura 5.6.

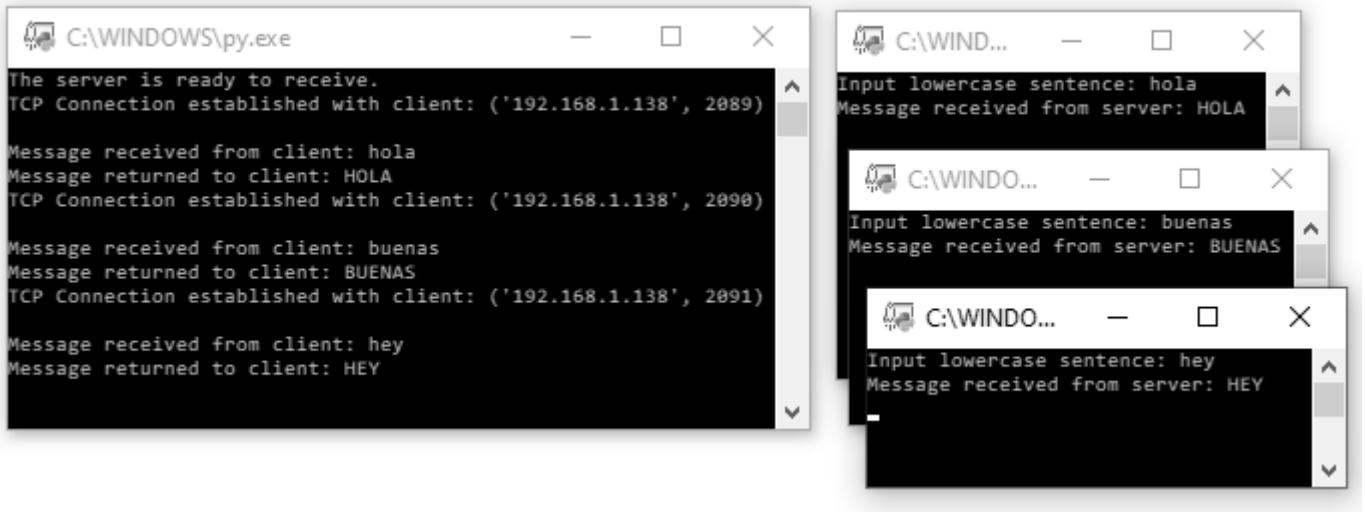


Figura 5.6. Salidas del proceso servidor (izquierda) y de tres procesos clientes (derecha).

Como habréis imaginado, ejecutar los procesos cliente y servidor en un mismo ordenador no es lo habitual. (Esto solo se hace durante la fase de pruebas de los programas). En la práctica, los procesos cliente y servidor se ejecutan en dos máquinas distintas. Vamos a ver cómo hacerlo.

Para empezar, necesitamos modificar ligeramente los programas `Client_TCP.py` y `Server_TCP.py`. (Los cambios que habría que efectuar en los programas `Client_UDP.py` y `Server_UDP.py` serían esencialmente los mismos). Modifica el programa `Client_TCP.py` como se indica a continuación y guárdalo como `Client_TCP(2).py`:

```
# server_name = socket.gethostname()  # Comment this line to avoid execution.
server_ip = '192.168.1.138'           # Write here server host IP address.
server_port = 12000                   # Write here server port.
```

Análogamente, cambia el programa `Server_TCP.py` como se indica y guárdalo como `Server_TCP(2).py`:

```
# server_name = socket.gethostname()  # Comment this line to avoid execution.
server_ip = '192.168.1.138'           # Write here local computer IP address.
server_port = 12000                   #Write here any port number (i.e. 12000).
```

Por supuesto, en vez de escribir la dirección IP 192.168.1.138, debes añadir la dirección IP del ordenador que vayas a utilizar como servidor. A continuación, guarda el programa servidor en el ordenador servidor (esto es, en el ordenador cuya IP hayas escrito en el código de los programas), y el programa cliente en otro ordenador cualquiera conectado a la misma red. En mi caso, el programa servidor lo guardé en mi ordenador, cuya IP era 192.168.1.138, y el programa cliente en un portátil con IP 192.168.1.135.

Ahora ejecuta el programa servidor en el ordenador servidor, y a continuación, al programa cliente en el ordenador cliente. Las figuras 5.7 y 5.8 muestran las salidas que yo obtuve para los programas cliente y servidor:

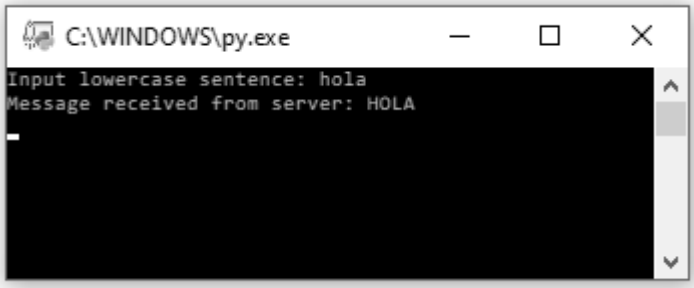


Figura 5.7. Salida del programa `Client_TCP(2).py` en el portátil cliente (IP 192.168.1.135).

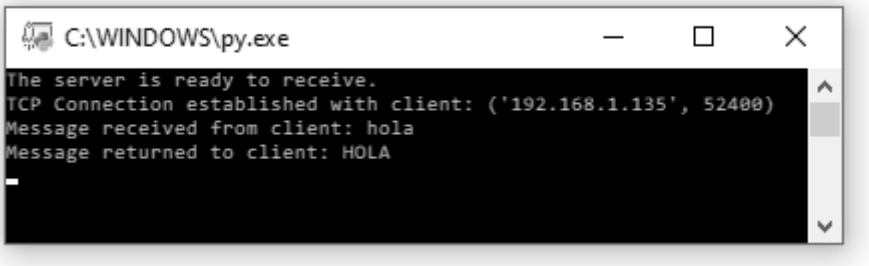


Figura 5.8. Salida del programa `Server_TCP(2).py` en el ordenador servidor (IP 192.168.1.138).

Pensemos en las implicaciones de estos resultados: Ejecuté el programa servidor en mi ordenador (IP 192.168.1.138), y el programa cliente en un segundo portátil (IP 192.168.1.135). En la ventana de comandos del ordenador cliente escribí la cadena `hola`. El ordenador cliente envió esa cadena al ordenador servidor a través de la conexión TCP previamente establecida. El ordenador servidor recibió dicha cadena, la pasó a mayúsculas, y se la envió de vuelta al portátil cliente. En otras palabras, con estos dos programas conseguí comunicar dos ordenadores a través de la red WiFi de mi casa, y enviar datos entre ellos. ¿No te parece increíble?

PRÁCTICA P. PROGRAMACIÓN DE SISTEMAS CLIENTE – SERVIDOR EN PYTHON.

P.1. SERVICIO DE CHAT.

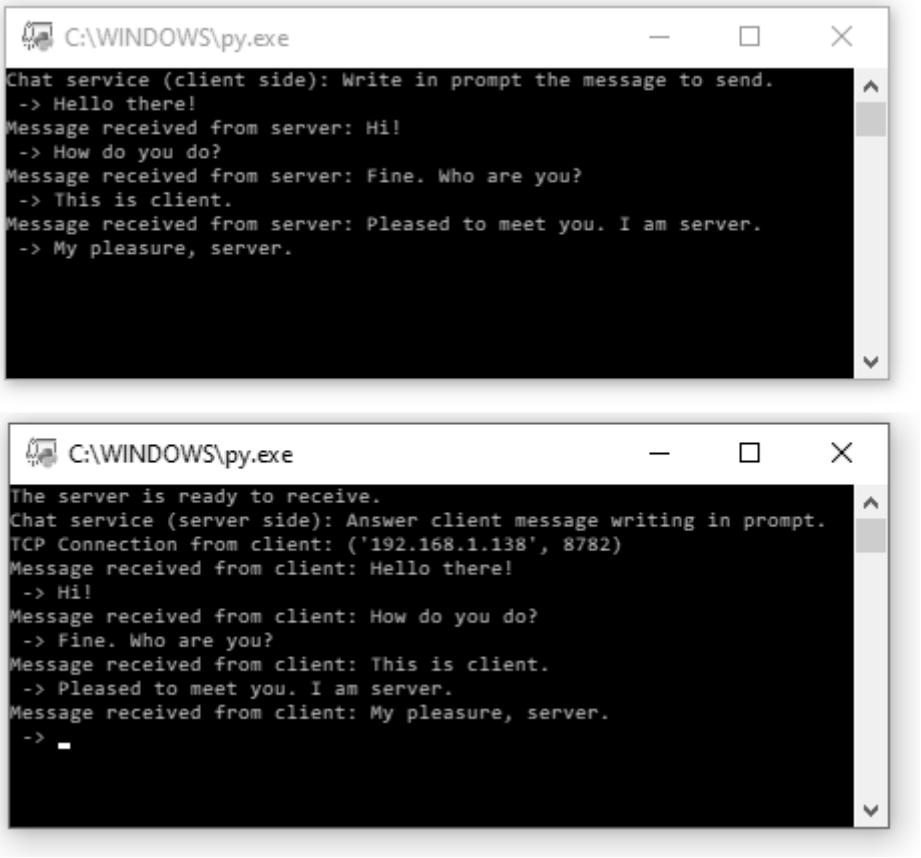


Figura S.1. Salidas del servicio de chat: Cliente (arriba) y servidor (abajo).

P.2. CONEXIÓN CON UN SERVIDOR WEB.

https://bedfordsarah.wordpress.com/2013/10/29/python-socket-programming-project-1-http-web-server/

https://docplayer.net/81346430-Socket-programming-assignment-5-icmp-pinger.html

Kurose – Ross 179, 180.

PARTE IV: OFIMÁTICA.

1. HOJAS DE CÁLCULO: EXCEL.

1.1. HOJAS DE CÁLCULO.

Las hojas de cálculo son programas informáticos que permiten realizar operaciones (cálculos) con todo tipo de datos (números, textos, fechas, monedas, porcentajes, etc.) siempre que esos datos puedan organizarse en forma de tabla.

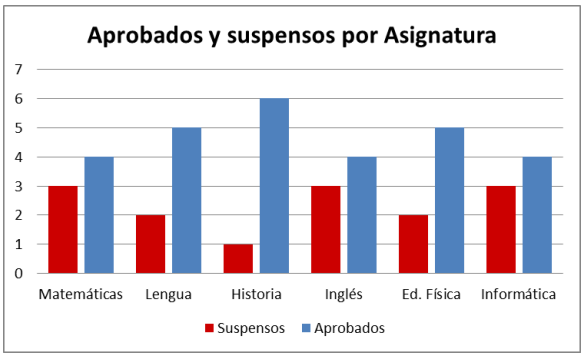
La característica más importante de las hojas de cálculo es su capacidad de **recalcular todos los valores obtenidos sin más que cambiar los datos iniciales**. Esto es importante cuando los cálculos son extensos o complicados: Imagina que estás realizando una declaración de la renta a mano y al final descubres un error en los datos de partida; en tal caso tendrías que volver a recalcularlo todo. Si esto lo haces con Excel, sólo es necesario corregir el dato erróneo.

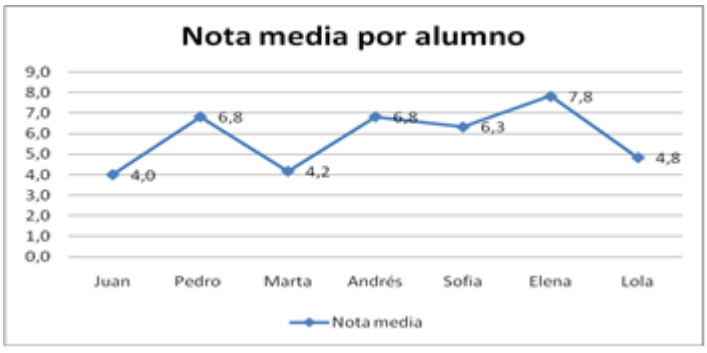

Introducción.xlsx


Introducción 2.xlsx

	Matemáticas	Lengua	Historia	Inglés	Ed. Física	Informática	**Nota media**
Juan	4	2	4	7	3	4	4,0
Pedro	7	6	8	2	9	9	6,8
Marta	2	4	8	0	5	6	4,2
Andrés	9	5	10	6	8	3	6,8
Sofia	4	6	5	9	4	10	6,3
Elena	10	8	7	7	5	10	7,8
Lola	8	5	5	3	5	3	4,8
Nota media	6,3	5,1	6,7	4,9	5,6	6,4	
Número de:							**TOTAL**
Suspensos	3	2	1	3	2	3	14
Aprobados	4	5	6	4	5	4	28

Las hojas de cálculo no son meras calculadoras que realizan operaciones con los datos, sino que además **sirven para ordenar, analizar, procesar y representar los datos**.





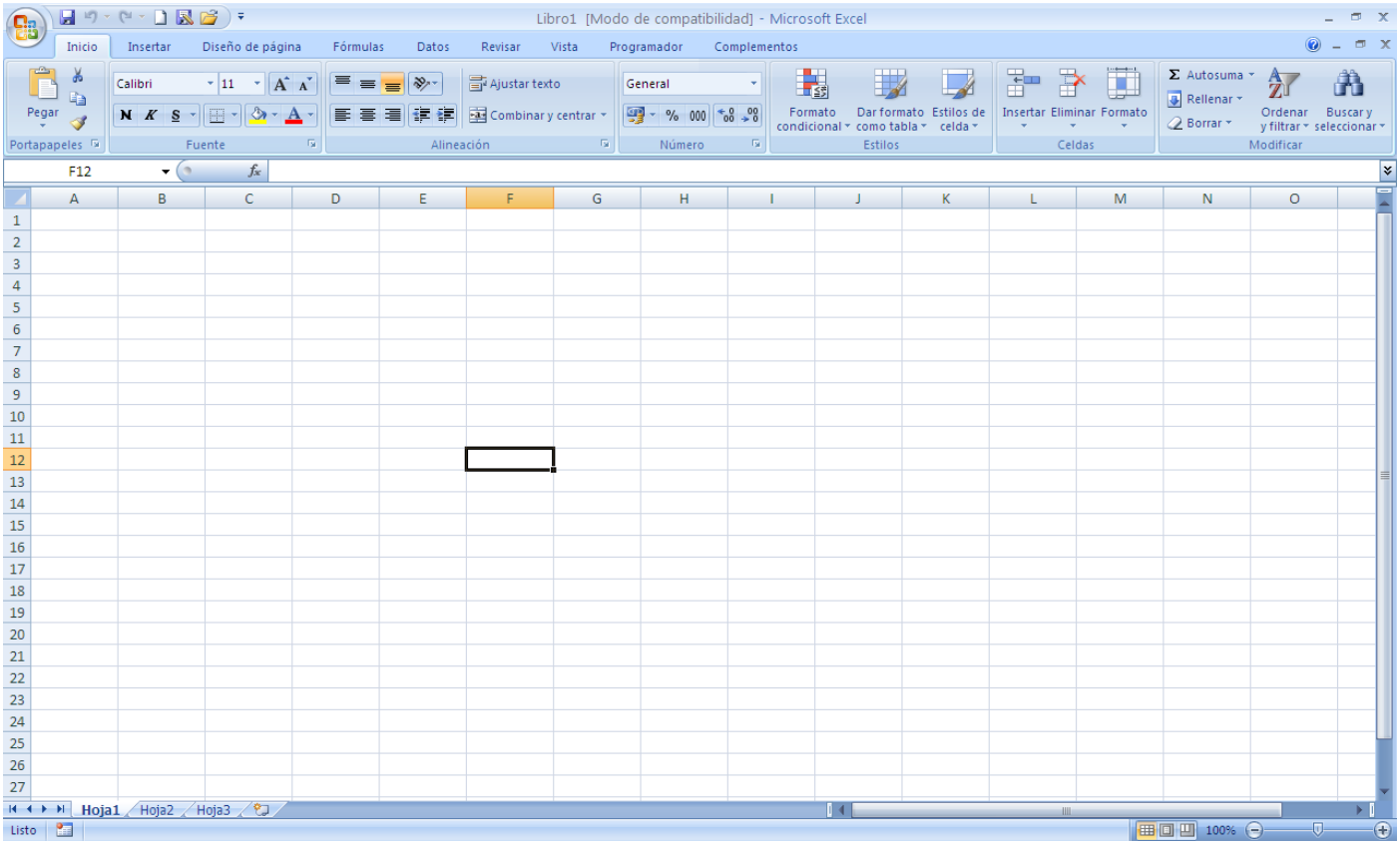
Hojas de cálculo hay muchas: Calc (OpenOffice), Gnumeric (Linux), Numbers (Apple), etc. Una de las hojas de cálculo más extendidas y utilizadas en la actualidad es **Excel**, y es la aplicación comercial que vamos a utilizar en este tema.

1.2. MICROSOFT EXCEL.

Excel es una hoja de cálculo integrada en Microsoft Office. Esto quiere decir que si ya conoces otro programa de Office, como Word, Access, Outlook, PowerPoint, etc. te resultará familiar utilizar Excel, puesto que muchos iconos y comandos funcionan de forma similar en todos los programas de Office.

CONCEPTOS BÁSICOS DE EXCEL.

Repasemos la terminología asociada al trabajo con hojas de cálculo:



Libro → Archivo de Excel (Libro1.xlsx).

Hojas → Cada archivo de Excel está formado por varias hojas. En principio consta de 3 hojas, pero se pueden ampliar hasta 255 (Hoja 1, Hoja 2, etc.). Cada hoja presenta una cuadrícula formada por muchísimas filas y columnas.

Columnas → conjunto de celdas verticales. Cada columna se nombra por letras: A, B, C,..., X, Y, Z, AA, AB, AC, AD, … , ZX, ZY, ZZ, AAA, AAB, etc.

Filas → conjunto de celdas horizontales. Cada fila se nombra con un número (1, 2, 3, 4, 5, etc.).

Celda → Las cuadrículas de Excel se llaman celdas. Una celda es la intersección de una columna y una fila en una hoja. Se nombra con el nombre de su columna y a continuación el número de su fila (B9 → celda en la columna B y la fila 9).

Celda activa → Es la celda sobre la que se sitúa el cursor, preparado para trabajar con ella. Se identifica porque aparece más remarcada que las demás (en la imagen, la celda F12).

Rango → Conjunto de varias celdas que Excel trata como una unidad. Los rangos son vitales en las hojas de cálculo, ya que todo tipo de operaciones se realizan a base de rangos. Por ejemplo, el rango B3:B8 identifica

al conjunto de celdas comprendidas entre las celdas B3 y B8 (es decir, las celdas B3, B4, B5, B6, B7, B8). El rango E4:F7 identifica al connunto de celdas comprendidas entre las celdas E4 y F7 (es decir, E4, E5, E6, E7, F4, F5, F6, F7). Como se observa, los rangos se especifican indicando la celda inicial y la celda final del rango, reparadas por dos puntos.

Algunos trucos útiles:

- Para acudir a una celda concreta (por ejemplo, la celda RV4527), basta con acudir al cuadro de la celda activa y escribir la referencia de la celda a la que queremos ir.
- Para acudir a la última columna de una hoja Excel, pulsamos el botón "Fin" y a continuación (o simultáneamente, dependiendo de la versión de Excel), pulsamos el botón de la flecha hacia la derecha.
- Para acudir a la última fila de una hoja Excel, pulsamos el botón "Fin" y a continuación (o simultáneamente, dependiendo de la versión de Excel), pulsamos el botón de la flecha hacia abajo.
- La primera celda de una hoja de Excel siempre es la celda A1.
- En la barra de menús de Excel, en la pestaña "Inicio", podremos encontrar algunas de las herramientas habituales de todos los programa de Microsoft Office, como son el tipo de fuente[18]; el tamaño de la fuente; el color de la fuente; las herramientas de negrita, cursiva, y subrayado; el color de relleno[19]; la alineción de la fuente (derecha, izquierda, o centrada); la herramienta de bordes, etc.
- La herramienta de "Combinar y centrar" nos permite combinar varias celdas en una sola celda (y además, centrar el texto que escribamos dentro de esa celda combinada).

EMPEZAR A TRABAJAR CON EXCEL.

Tipos de datos. En las celdas se pueden introducir datos de muy diverso tipo (texto, números, fechas, fórmulas, etc.). Los datos se introducen escribiéndolos directamente con el teclado en la celda donde se deseen añadir. Estos datos pueden ser:

- **Textos**, como por ejemplo Hola; Adiós; Me llamo Alejandro; Tengo 23 años; Mi dirección de correo es alex3453@yahoo.es; etc. En general, Excel trata cualquier combinación de caracteres imprimibles (letras, dígitos numéricos, y símbolos como puntos, comas almohadillas, arrobas, etc.) como textos. Notar que la alineación por defecto de todos los datos tipo texto es a la izquierda.

- **Números**, como por ejemplo, 6; -3; $2{,}5$; $-3{,}75$; etc. Los números decimales se escriben con coma, no con punto (si bien esto depende de la configuración regional del equipo). Notar que la alineación por defecto de todos los datos numéricos es a la derecha.

- **Fechas**, como por ejemplo, 21-12-1997, 21/12/1997, 21-12-77, 21-dic-77, etc. Notar que las fechas son combinaciones especiales de números y símbolos como guiones y barras inclinadas, que no son ni números ni textos. Observar que la alineación por defecto de todos los datos tipo fecha es a la derecha.

- **Monedas**. Excel también nos permite escribir cantidades de dinero, en nuestro caso euros, pero también dólares, libras, yenes, etc. Para escribir 30€ en una celda, basta con escribir el 30, y a continuación, añadir el símbolo del euro localizado en la tecla de la letra E. El dato escrito no es un texto, ni un número, sino que se tratará como una cantidad de dinero en euros. La alineación por defecto de todos los datos de tipo moneda es a la derecha.

---

[18] Algunas de las fuentes más utilizadas en Excel son Calibri (la fuente por defecto), Comic Sans MS, Arial, Times New Roman, etc.

[19] Por color de relleno nos referimos al color de fondo de la celda, que por defecto es "Sin relleno". Siempre que cambiemos el color de relleno veremos que los bordes de las celdas afectadas desaparecen, no porque no sigan allí, sino porque los hemos pintado de ese color (incluso si elegimos el color blanco).

- **Porcentajes**. Los porcentajes son tipos de datos especiales que nos permiten obtener porcentajes de otros tipos de datos, fundamentalmente numéricos y de monedas. Un ejemplo de dato de tipo porcentaje es 20%. Este dato no es ni un texto ni un número; el símolo % indica que es un porcentaje. La alineación por defecto de todos los datos de tipo porcentaje es a la derecha.

Fórmulas. Los distintos datos que escribamos en distintas celdas pueden operarse mediante fórmulas. Toda fórmula debe precederse del signo igual (=) para que Excel identifique esa operación como una fórmula.

Una fórmula, además de números y operaciones, puede contener referencias a datos en otras celdas. A la hora de escribir una fórmula en Excel, podemos hacerlo de dos formas, pero solo una de ellas es la correcta. Imaginemos que queremos realizar la suma de los números 4 y 9. Una opción sería escribir directamente en una celda =4+9. Como vemos en la celda donde hemos escrito la fórmula, el resultado es 13. Otra opción seria escribir el 4 en una celda (por ejemplo, la A1) y el 9 en otra celda (por ejemplo, la A2), para a continuación escribir en una tercera celda la expresión =A1+A2. Como vemos en la celda donde hemos ecrito la fórmula, el resultado vuelve a ser 13.

¿Cuál de las dos opciones es la correcta? Como hemos comentado en la introducción, la mayor utilidad de Excel es que si cambiamos los datos de partida, los resultados que operan dichos datos también se actualizan. En efecto, si acudimos a las celdas A1 y A2 y cambiamos los dato que queremos sumar, veremos que la fórmula =A1+A2 cambia el resultadoque ofrece para adpatarse a esos datos, pero la fórmula =4+9 no (porque siempre suma el 4 y el 9).

Por consiguiente, siempre debemos tener presente la siguiente regla: En las fórmulas, nunca debemos poner directamente los datos que queremos operar, sino la referencia a la celda donde se localizan esos datos.

Operaciones con los datos. Es importante tener claro qué tipos de operaciones podemos realizar con los distintos tipos de datos. Esto es así porque hay operaciones que podemos hacer con unos datos, pero que no podemos hacer con otros.

Los datos numéricos puede sumarse, restarse, multiplicarse, dividirse, y exponenciarse. Algunos ejemplos de operaciones con números son:
= 5+9 (resultado: 14).
= 6-3 (resultado: 3).
= 2*8 (resultado: 16).
= 24/2 (resultado: 12).
= 2^4 (resultado: 16).

Los textos no pueden sumarse, ni restarse, ni multiplicarse, etc. No tiene sentido hacer esas operaciones con los textos. Lo que sí que podemos hacer con ellos en concatenarlos, contar sus caracteres, y otro tipo de operaciones que estudiaremos más adelante. Tampoco podemos sumar n multiplicar un texto por un número, tampoco tiene sentido.

Las fechas tampoco pueden sumarse, restarse, etc. entre sí ni con otros números; no son operaciones lógicas. Lo que sí que podremos hacer es calcular el número de días entre dos fechas, extraer el día de la semana crrespondiente a una cierta fecha, etc.

Las cantidades de dinero pueden sumarse y restarse entre sí, siempre que sean del mismo tipo de moneda. También podemos multiplicar una cantidad de dinero por un número, y dividirla por un número. Imaginemos que en laa celda A1 tenemos el dato 20€ y en la celda A2 el dato 30€; algunos ejempos de operaciones válidas son:
= A1+A2
= A2-A1

=A2*3
=A1/4

Por último, los porcentajes son un tipo especial de datos. Por ejemplo, si deseamos extaer el 20% de 70, escribimos 20% en la celda A1 y 70 en la celda A2, y usamos la fórmula =A1*A2. Como vemos, para sacar el procentaje de un número, basta con multiplicar ese porcentaje por ese número. A la inversa, si queremos obtener que porcentaje es 25 respecto de 80, escribimos cada número en una celda (A1 y A2, respectivamente), y dividimos el número menor entre el número mayor (con la fórmula =A1/A2). A continuación, seleccionamos la ceda donde hemos efectuado esa división, y en el bloque "Número" de la pestaña "Inicio", seleccionamos porcentaje en la lista desplegable. El resultado debería ser 31,25%, lo que significa que 25 es el 31,25% de 80. Tambiénn podemos sacer y extraer porcentaje de cantidades de dinero.

Referencias absolutas. Como ya hemos indicado, para referenciar los datos a operar en una fórmula no se suelen escribir directamente los datos, sino las celdas donde se encuentran dichos datos.

	A	B	C	D
1	4	8		=A1+B1
2				

Una vez escrita la fórmula, ésta se puede arrastrar para copiarla, y operar de la misma forma el resto de datos. Excel es un programa inteligente, y no copia la fórmula textualmente, sino que actualiza las celdas referenciadas para operar los datos adecuados.

	A	B	C	D	E
1	4	8		12	<--- =A1+B1
2	2	7		9	<--- =A2+B2
3	2	3		5	<--- =A3+B3
4	9	1		10	<--- =A4+B4
5	6	5		11	<--- =A5+B5

Cuando al copiar una fórmula no se desea actualizar automáticamente las celdas referenciadas, hay que hacer una referencia absoluta (referencia fija) a la celda que no se desea que se actualice con la copia. Esto se consigue añadiendo el símbolo $ (dólar) delante de la letra del número de la celda que se desea fijar.

	A	B	C	D	E	F
1	4	*	2	=	8	<--- =A1*C1
2	2	*		=	4	<--- =A2*C1
3	2	*		=	4	<--- =A3*C1
4	9	*		=	18	<--- =A4*C1
5	6	*		=	12	<--- =A5*C1

Cálculos combinados. Cuando en una misma fórmula hay que efectuar varias operaciones, Excel resolverá las operaciones dentro de la fórmula con un determinado orden de prioridad, siguiendo la jerarquía habitual de las operaciones matemáticas: Primero se evalúan las operaciones dentro de los paréntesis y corchetes, sean cuales sean; después exponentes y raíces; luego multiplicaciones y divisiones; y por último sumas y restas.

Por ejemplo, imaginar que queremos obtener la nota media de las notas 3; 7; 6; y 9. Una persona poco cuidadosa escribiría =3+7+6+9/4, pero el resultado de esta operación es 18,25, que es obviamente incorrecto. Notar que para obtener la media primero debemos sumar las notas parciales, y luego dividir entre el número de notas. Como la suma es una operación de menor jerarquía que la división, para hacerla

antes debemos ponerla entre paréntesis: =(3+7+6+9)/4. El resultado de esta nueva fórmula es 6,25, lo cual es correcto.

APLICAR FORMATO.

Aplicar formato significa "decorar" las hojas Excel para cambiar su aspecto, ya sea cambiando el tipo, color, o tamaño de la fuente, el color de relleno, la aplicación de bordes, la combinación de celdas, etc. Los pasos a seguir para dar un formato correcto a una tabla de Escel son los siguientes:

1) En primer lugar, escribimos los datos y las fórmulas que beben de ellos sin aplicar ningún tipo de formato.
2) A continuación, especificamos el tipo de datos contenido en cada una de las celdas, ya contengan datos o fórmulas. Para ello, acudimos al bloque "Número" de la pestaña "Inicio". Esto es especialmente útil en los datos numéricos, porcentajes, y monedas.
3) Finalmebte, utilizamos las herramientas de la pestaña inicio para aplicar a la tabla el formato deseado (tamaños, colores, bordes, etc.). Notar que el formato es lo último que se aplica sobre la tabla, incluso después de conseguir que la tabla funcione correctamente.

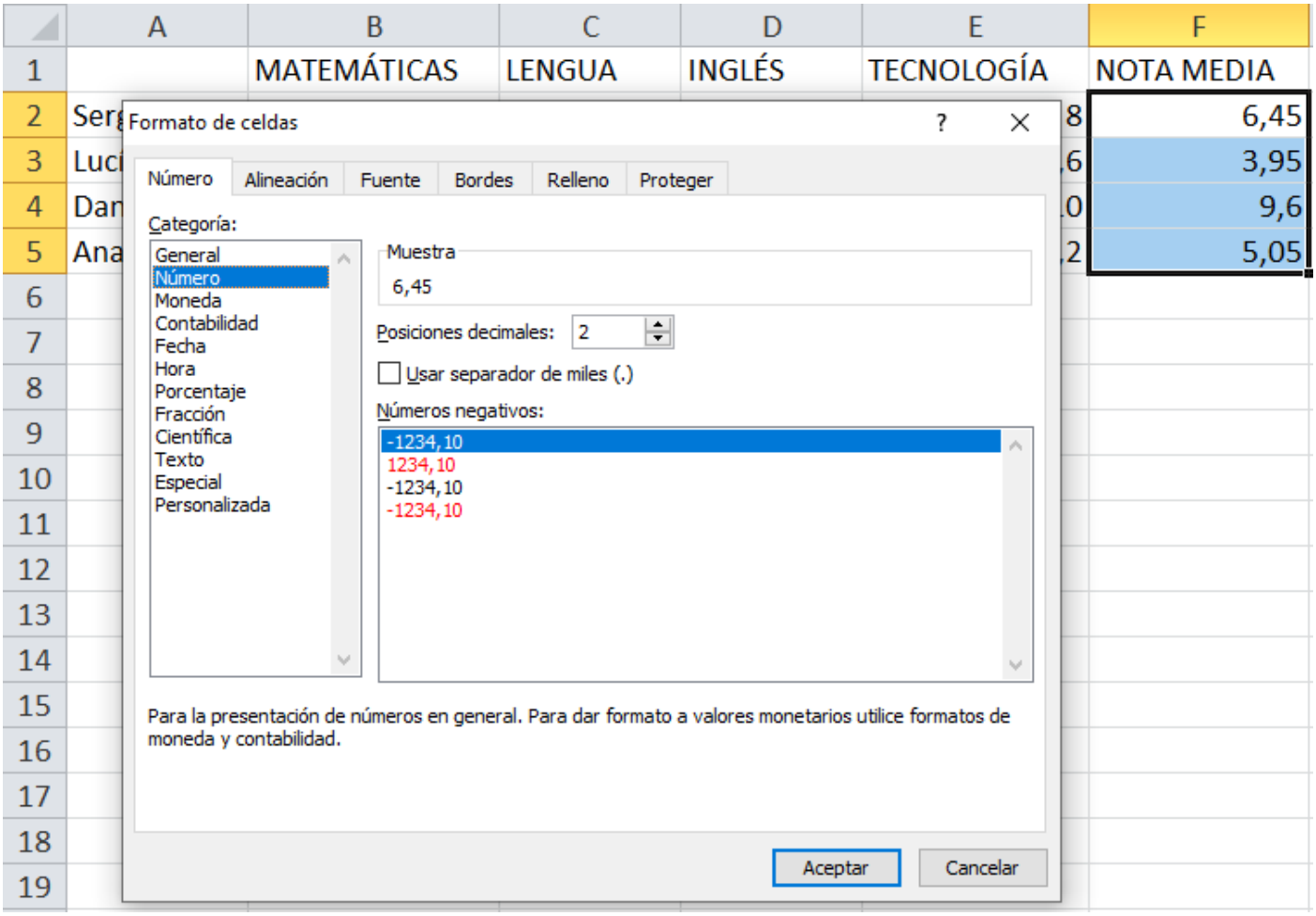
A modo de ejemplo, vamos a contruir partiendo de cero la siguiente tabla Excel:

	A	B	C	D	E	F
1		MATEMÁTICAS	LENGUA	INGLÉS	TECNOLOGÍA	NOTA MEDIA
2	Sergio	5,3	9	3,5	8	
3	Lucía	4,5	3,4	2,3	5,6	
4	Daniel	10	9	9,4	10	
5	Ana	5,1	4,8	2,1	8,2	
6						

A continuación, escribimos las fórmulas:

	A	B	C	D	E	F	G
1		MATEMÁTICAS	LENGUA	INGLÉS	TECNOLOGÍA	NOTA MEDIA	
2	Sergio	5,3	9	3,5	8	6,45	
3	Lucía	4,5	3,4	2,3	5,6	3,95	
4	Daniel	10	9	9,4	10	9,6	
5	Ana	5,1	4,8	2,1	8,2	=(B5+C5+D5+E5)/4	
6							

Después especificamos el tipo de datos contenido en las distintas celdas: Los datos de la columna A son textos; los datos de la fila 1 también son textos, las notas parciales son números con un decimal, y las notas medias son números con dos decimales. Este último comentario constituye una regla general: El resultado de operar datos numéricos con un cierto número de decimales siempre debe añadir un decimal más. En este caso, como la nota media es el resultado de operar notas parciales con un decimal, la nota media debe añadir dos decimales.



Ahora que la tabla ya funciona correctamente, empezamos a aplicar formato. Cambiamos el formato y el color de la fuente, la alineación, el color de relleno, y aplicamos bordes como indica la figura.

	MATEMÁTICAS	LENGUA	INGLÉS	TECNOLOGÍA	NOTA MEDIA
Sergio	5,3	9,0	3,5	8,0	**6,45**
Lucía	4,5	3,4	2,3	5,6	**3,95**
Daniel	10,0	9,0	9,4	10,0	**9,60**
Ana	5,1	4,8	2,1	8,2	**5,05**

Usamos la herramienta de "combiar y centrar" para añadir el encabezado.

NOTA MEDIA POR ALUMNO					
	MATEMÁTICAS	LENGUA	INGLÉS	TECNOLOGÍA	NOTA MEDIA
Sergio	5,3	9,0	3,5	8,0	**6,45**
Lucía	4,5	3,4	2,3	5,6	**3,95**
Daniel	10,0	9,0	9,4	10,0	**9,60**
Ana	5,1	4,8	2,1	8,2	**5,05**

Finalmente, si una de las asignaturas tuviese un nombre mucho más largo que las demás, utilizamos las herramientas de "ajustar texto" y de alinear en vertical para formatear los encabezados:

	NOTA MEDIA POR ALUMNO				
	MATEMÁTICAS	LENGUA	INGLÉS	TECNOLOGÍA ROBÓTICA	NOTA MEDIA
Sergio	5,3	9,0	3,5	8,0	**6,45**
Lucía	4,5	3,4	2,3	5,6	**3,95**
Daniel	10,0	9,0	9,4	10,0	**9,60**
Ana	5,1	4,8	2,1	8,2	**5,05**

EXCEL ACTIVIDAD 1.

Realiza las siguientes actividades sobre operaciones básicas y formato. Haz cada ejercicio en una hoja de un único archivo Excel, que debes guardar como Ejer1.xlsx.

En la Hoja 1:

CÁLCULOS COMBINADOS

1) NOTAS DE UN ALUMNO (1).

NOTA 1	NOTA 2	NOTA 3	NOTA 4	NOTA 5	NOTA MEDIA
7	4,5	6	5	4	

2) NOTAS DE UN ALUMNO (2).

NOTA 1	NOTA 2	NOTA 3	NOTA 4	NOTA 5	NEGATIVOS	NOTA FINAL
5,9	4,1	8,3	7,7	5,2	4	

Observación: la nota final se calcula como la media de las notas parciales, restándole 0,1 puntos por cada negativo.

3) NOTAS DE UN ALUMNO (3).

NOTA 1	NOTA 2	NOTA 3	NOTA 4	NOTA 5	NEGATIVOS	ACTITUD	NOTA FINAL
4,7	3,7	6,2	6,9	8,1	3	0,5	

Observación: El 90% de la nota final se calcula como la media de las notas parciales. La calificación restante viene dada por la actitud, que toma valores entre 0 y 1 puntos. Por último, cada negativo resta 0,1 puntos a la nota final.

En la Hoja 2:

LIBRERÍA "EL ESTUDIANTE"

Código	Artículo	Cantidad vendida	Precio unitario	Subtotal	IVA	TOTAL
	Goma	20	0,50 €			
	Lápiz	35	0,70 €			
	Bolígrafo	30	0,99 €			
	Cuaderno	24	1,75 €			
	Regla	30	1,15 €			
	Compás	15	2,50 €			

IVA	18%

1) Completar los códigos del artículo como una serie automática, introduciendo AR1 en la primera celda, y arrastrando el contenido a las siguientes
2) Calcular el SUBTOTAL multiplicado la cantidad vendida por el precio unitario
3) Calcular el IVA multiplicando el subtotal por el valor indicado en la celda de IVA (18%).
4) Calcular el TOTAL sumando el subtotal + el IVA.

En la Hoja 3:

TABLA DINÁMICA DE MULTIPLICAR

Tabla del	3

1	por	3	es igual a	3
2	por	3	es igual a	6
3	por	3	es igual a	9
4	por	3	es igual a	12
5	por	3	es igual a	15
6	por	3	es igual a	18
7	por	3	es igual a	21
8	por	3	es igual a	24
9	por	3	es igual a	27
10	por	3	es igual a	30

=C4

(Referencia absoluta)

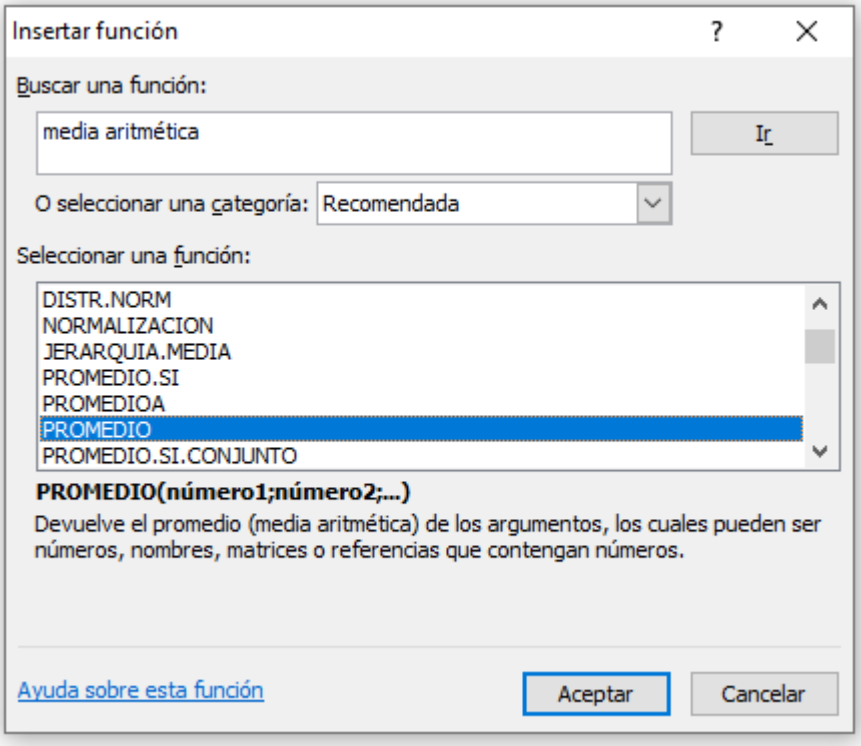
En la Hoja 4:

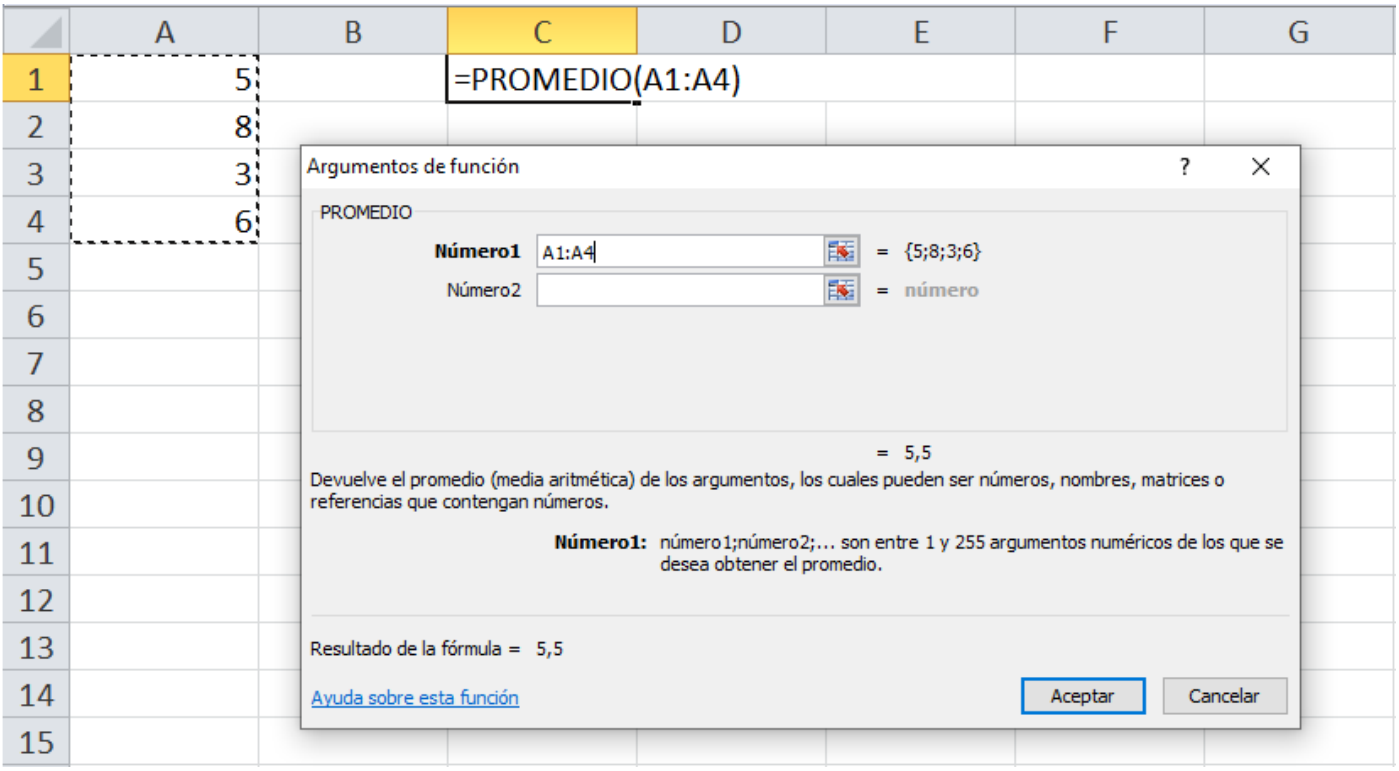
	A	B	C	D	E	F	G	H	I
1									
2		**TRANSPORTES TRANSIBÉRICA S.A.**							
3									
4		**TRANSIBÉRICA S.A.**					**DATOS DEL CLIENTE**		
5		AVDA. CONSTITUCIÓN s/n					Remedios Buendía		
6		Albacete					C/ Panamá nº 3		
7		TLF: 967 11 22 33					Madrid (CP: 46010)		
8		CIF: A - 12345678					NIF: 49015061 - K		
9									
10		**Código**	**Vehículo**	**Destino**	**Número de km**	**Precio del km**	**Subtotal**	**IVA**	
11		CD1	Camión grúa	Madrid	125	2,40 €			
12		CD2	Furgoneta	Valencia	235	1,80 €			
13		CD3	Autocar	La Coruña	200	2,70 €			
14		CD4	Camión frigorífico	Bilbao	120	3,10 €			
15		CD5	Coche sin conductor	Valencia	500	1,60 €			
16		CD6	Minibus	Madrid	100	2,00 €			
17		CD7	Furgoneta	Sevilla	80	1,80 €			
18									
19		**IVA**	**18%**			**SUMA**			
20						**TOTAL A PAGAR**			
21									
22									

1.3. FUNCIONES.

USAR FUNCIONES EN EXCEL.

Hasta ahora solo hemos escrito fórmulas para realizar operaciones que únicamente implicaban la utilización de operaciones básicas como sumas, restas, multiplicaciones, y divisiones (como por ejemplo, el cálculo de un subtotal, un IVA, y un total; o la obtención de una nota media). Sin embargo, Excel permite la realización automática de multitud de operaciones mucho más complejas (matemáticas, estadísticas, lógicas, de fechas y hora, de búsqueda, de manipulación de textos, etc.), que ofrece ya preconstruidas en forma de **funciones**. Sea cual sea la oeración que deseemos realizar, seguro que Excel incluye una función capaz de realizarla.





La utilización de funciones requiere de un cierto conocimiento de las mismas (su nombre, para qué sirven, resultado que ofrecen, sintaxis, datos de entrada aceptados, etc.). Sin embargo, y para los principiantes en el uso de funciones, Excel proporciona un asistente de funciones. Para acceder al asistente basta con pulsar en el botón de insertar función (f_x) junto al cuadro de navegación de celdas, o acudir a la biblioteca de funciones en la pestaña de Fórmulas.

A través de ambas vías se abrirá una ventana de diálogo donde podemos buscar la función (escribiendo una descripción, buscando por categorías, etc.). Al seleccionar la función que deseamos utilizar aparece el asistente de uso de funciones, que nos informará sobre cómo opera la función, operadores que requiere, resultado que devuelve, etc.

FUNCIONES COMUNES EN EXCEL.

La siguiente tabla incluye algunas de las funciones de uso más habitual en Excel.

FUNCIONES ESTADÍSTICAS.	
=PROMEDIO()	Esta función nos devuelve el promedio (la media aritmética) de los números o del rango entre paréntesis. Ejemplo: =PROMEDIO(4,5,6) devuelve el valor 5
=MAX()	Esta función nos devuelve el valor máximo de una lista de números o de celdas. Ejemplo: =MAX(12,7,125) devuelve el valor 125
=MIN()	Esta función nos devuelve el valor mínimo de una lista de números o de celdas. Ejemplo: =MIN(12,7,125) devuelve el valor 7
=MODA()	Esta función nos devuelve el valor más repetido de una lista de números o de celdas. Ejemplo: =MODA(6,2,3,2,9) devuelve el valor 2 (el más repetido).
=CONTAR()	Devuelve el número de celdas de un rango que contienen datos numéricos.
=CONTAR.BLANCO()	Cuenta el número de celdas en blanco de un rango.

=CONTARA()	Cuenta el número de veces que aparece contenido en forma de texto o en forma de números en un rango de celdas. Ejemplo: =CONTARA(A1:A10). Si en esas diez celdas hay seis que contienen algún tipo de texto, otras dos que contienen números, y el resto están vacías, el valor devuelto es 8 (porque hay 6 celdas que contienen texto más otras 2 que contienen números).
=SI()	Comprueba si se cumple una condición. Si el resultado es VERDADERO devuelve un valor, mientras que si es FALSO devuelve otro valor. Ejemplo: =SI(A1>=5;"aprobado";"suspenso"). Si el valor que está en la celda A1 es, por ejemplo, 7, el valor devuelto es "aprobado" y si es menor que 5, el valor será "suspenso"
=CONTAR.SI()	Cuenta las celdas en un rango que cumplen una determinada condición especificada. Ejemplo: =CONTAR.SI(A1:A10;">=10"). Si en el rango especificado hay dos números mayores o iguales que 10, el valor devuelto es 2.
FUNCIONES MATEMÁTICAS	
= SUMA() =PRODUCTO()	Realiza la suma (+) o producto (∗) de la cadena de números especificada.
=SUMAR.SI	Suma las celdas indicadas si se cumple determinada condición.
=M.C.D.() =M.C.M.()	Funciones para obtener el máximo común divisor y el mínimo común múltiplo
=POTENCIA()	Devuelve el resultado de elevar un número a una potencia. Ejemplo: =POTENCIA(2;3) da 8, porque $2^3 = 2 \times 2 \times 2 = 8$
=RADIANES()	Convierte de grados a radianes.
=TRUNCAR()	Convierte un decimal en entero, eliminando la parte decimal.
=PI()	Devuelve el valor del número pi (π) con 15 dígitos.
=FACT()	Devuelve el factorial de un número ($n!$)
=ALEATORIO() =ALEATORIO.ENTRE()	Devuelve un número aleatorio entre 0 y 1. Devuelve un número aleatorio entre los números especificados.
= NUMERO.ROMANO()	Convierte un número a número romano, en formato texto.
=SENO() =COS() =TAN()	Calcula el seno, coseno, o tangente de un ángulo.
FUNCIONES DE FECHA Y HORA	
=AHORA() =HOY()	Devuelve la fecha y hora actuales.
=DIASEM()	Devuelve el día de la semana codificado del 1 al 7
=DIAS360()	Devuelve el número de días entre dos fechas (año de 360 días)
=AÑO() =MES() =DIA() =HORA()	Devuelve el año actual (1997, 2015), mes actual (1-12), día actual del mes (1-31), y hora actual (0-23), respectivamente.
FUNCIONES CON TEXTOS	
=CONCATENAR()	Une varias cadenas de texto en una sola.
=DERECHA()	Devuelve el número de caracteres especificado a la derecha de un texto.
=IZQUIERDA()	Idem.
=LARGO()	Devuelve el número total de caracteres de una cadena de texto (letras, dígitos numéricos, símbolos, espacios en blanco, etc.).
=IGUAL()	Comprueba si dos cadenas de texto son iguales.
=REEMPLAZAR()	Reemplaza parte de una cadena de texto por otra.
FUNCIONES DE BÚSQUEDA Y REFERENCIA	
=BUSCARV()	Busca un valor en la primera columna de la izquierda de una tabla y luego devuelve un valor en la misma fila y en la columna especificada.

Para más información sobre estas funciones, utiliza el asistente de funciones de Excel. Para conocer otras funciones, emplea la herramienta de búsqueda de funciones. Excel dispone de un gran repertorio de funciones, y para cualquier cosa que se te ocurra hacer, seguro que existe una función para hacerla. ¡Busca la función adecuada! Para terminar, la tabla a continuación resume los operadores de comparación más útiles, que necesitaremos usar dentro de algunas de las funciones previas.

Símbolo	Operación efectuada
=	Comparación IGUAL QUE
>	Comparación MAYOR QUE
>=	Comparación MAYOR o IGUAL QUE
<	Comparación MENOR QUE
<=	Comparación MENOR o IGUAL QUE
<>	Comparación DISTINTO QUE

EXCEL ACTIVIDAD 2.

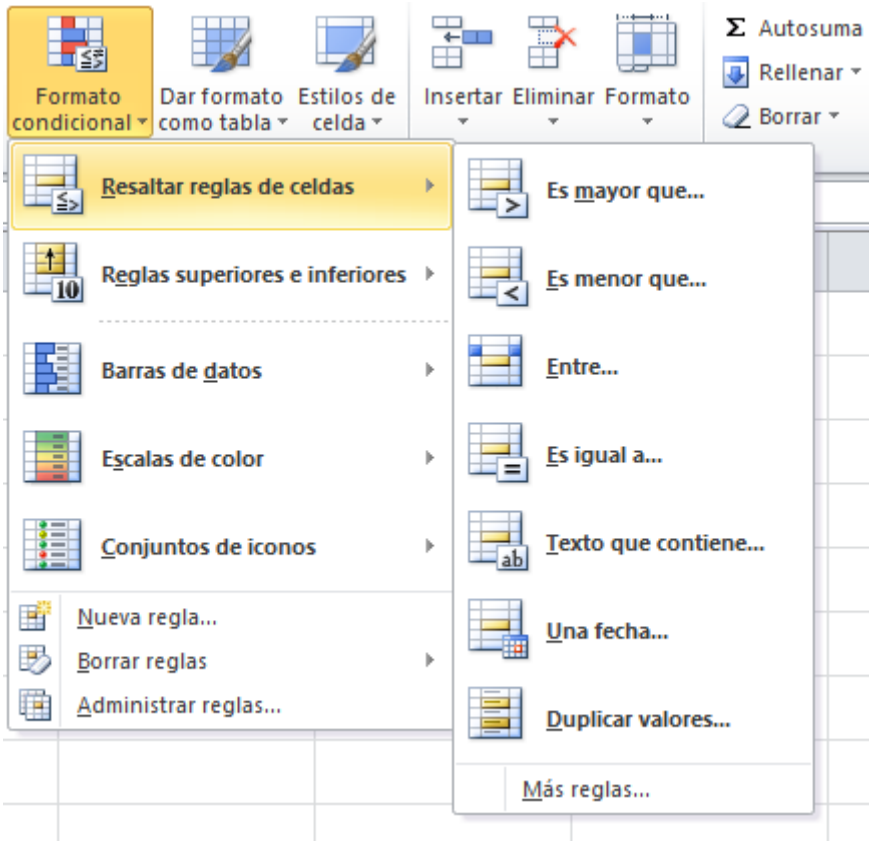
Abre el archivo "Ejercicios de funciones (BACH).xlsx", y realiza las actividades planteadas en cada una de sus hojas. Usa el asistente de funciones para encontrar la función adecuada y conocer cómo se usa.

EXCEL ACTIVIDAD 3.

En esta actividad vamos a utilizar algunas funciones más avanzadas. Abre el archivo "Ejercicios de funciones avanzadas (BACH).xlsx", y realiza las actividades planteados en cada una de sus hojas.

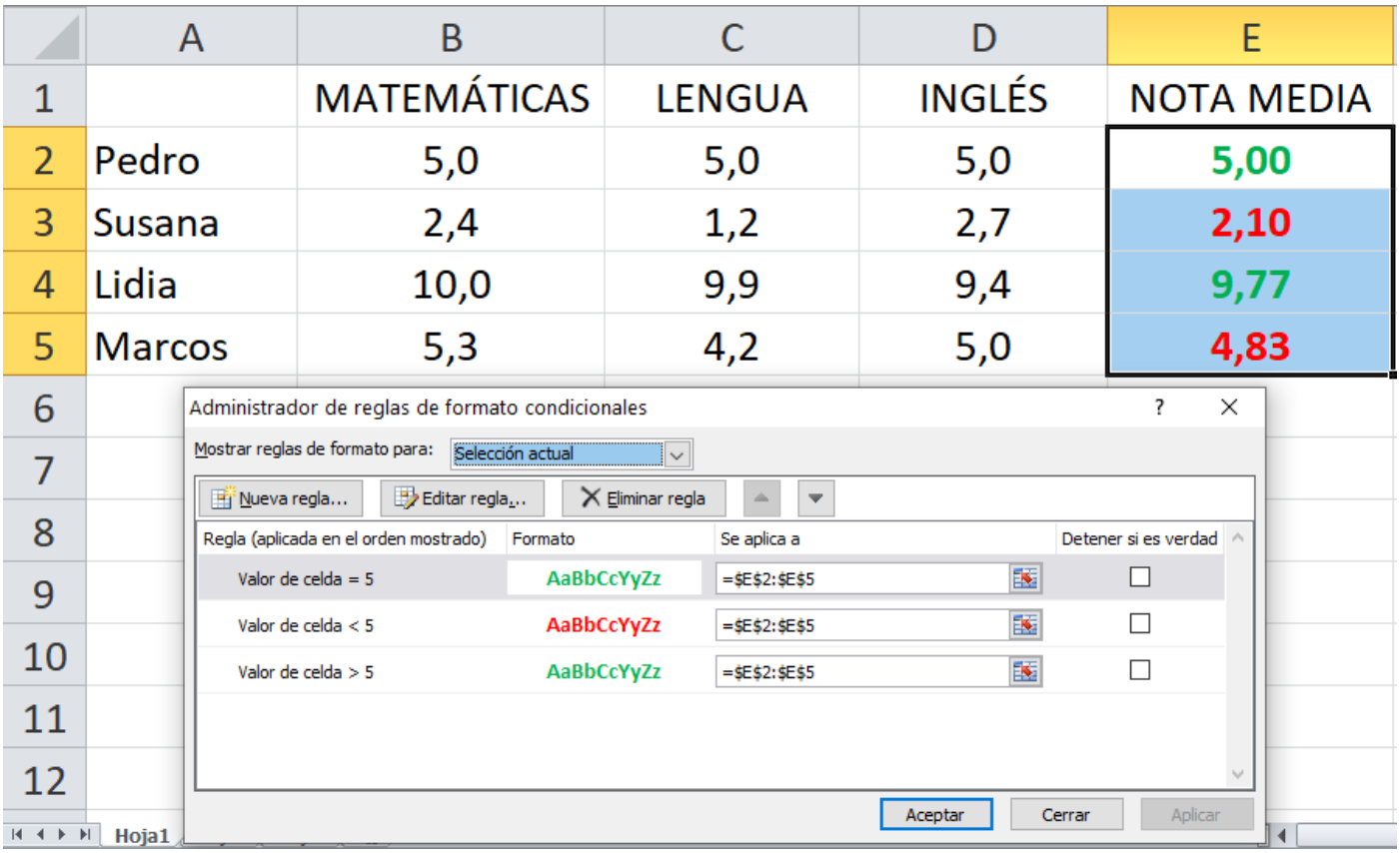
1.4. FORMATO CONDICIONAL.

El formato condicional nos permite aplicar un formato especial sobre una celda, dependiendo del valor contenido en esa celda. El formato condicional suele utilizarse para resaltar errores, para destacar valores que verifiquen una determinada condición, para enfatizar las celdas según el valor contenido en ellas, etc.



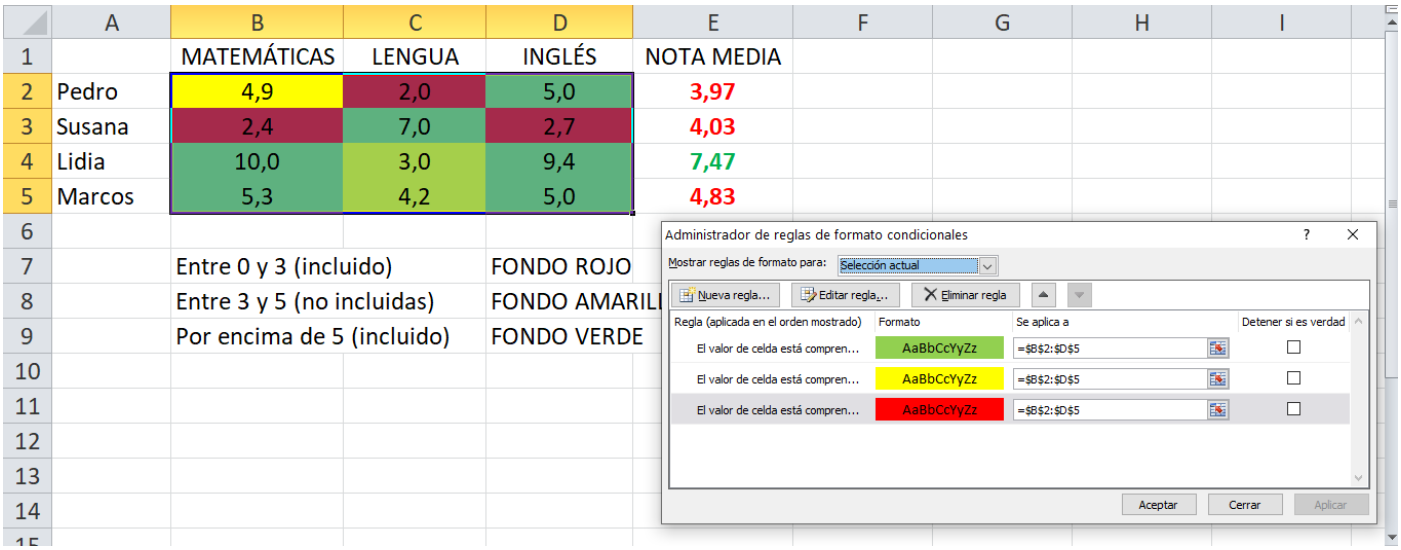
Para aplicar formato condicional basta con seleccionar la celda o rango de celdas de interés, y seleccionar la opción "Formato Condicional" de la pestaña "Inicio". Existen diversas opciones para establecer el formato condicional, pero la forma más sencilla es utilizar las reglas predefinidas que ofrece Excel, como vemos en la figura. Una vez definidas las reglas del formato condicional, la herramienta de "Administrar reglas" nos permite modificar, borrar, o añadir nuevas reglas de formato condicional.

A modo de ejemplo, vamos a aplicar formato condicional a una sencilla tabla de notas. En la primera figura, hemos aplicado formato condicional para poner en rojo y negrita todas las notas medias suspensas, y en verde y negrita todas las notas medias aprobadas. Notar que hemos necesitado definir dos reglas para las notas medias aprobadas (notas mayores que 5 y notas iguales a 5).

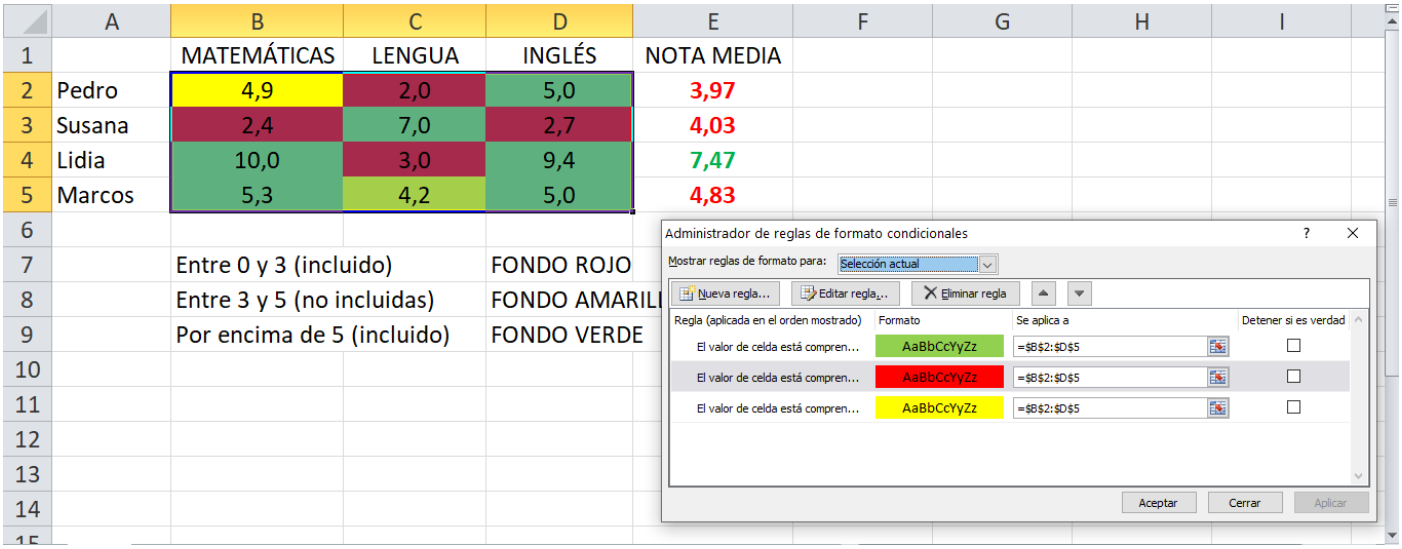


	A	B	C	D	E
1		MATEMÁTICAS	LENGUA	INGLÉS	NOTA MEDIA
2	Pedro	5,0	5,0	5,0	5,00
3	Susana	2,4	1,2	2,7	2,10
4	Lidia	10,0	9,9	9,4	9,77
5	Marcos	5,3	4,2	5,0	4,83

A continuación, hemos aplicado formato condicional para fijar el color de fondo de las notas parciales. Si la nota está entre 0 y 3 (3 incluído), el fondo debe aparecer rojo; si la nota está entre 3 y 5 (sin incluir ni al 3 ni al 5), el fondo debe ser amarillo; y si la nota está por encima de 5 (5 incluído), el fondo debería ponerse de color verde.



	A	B	C	D	E
1		MATEMÁTICAS	LENGUA	INGLÉS	NOTA MEDIA
2	Pedro	4,9	2,0	5,0	3,97
3	Susana	2,4	7,0	2,7	4,03
4	Lidia	10,0	3,0	9,4	7,47
5	Marcos	5,3	4,2	5,0	4,83
7		Entre 0 y 3 (incluido)		FONDO ROJO	
8		Entre 3 y 5 (no incluidas)		FONDO AMARILL	
9		Por encima de 5 (incluido)		FONDO VERDE	



La primera figura muestra nuestro primer intento de aplicar este formato condicional, donde observamos que las notas iguales a 3 no toman el formato deseado. Esto se debe a que tenemos dos reglas que definen dos colores distintos para el 3 (rojo y amarillo). La regla que se impone es aquella que está por encima en la pila de reglas definidas, de forma que para que el 3 se ponga en rojo, debemos mover la regla del rojo y situarla por encima de la regla del amarillo en la pila de jerarquías.

Para terminar, eliminamos las reglas que hemos definido para las notas parciales, y en lugar de cambiar su color de fondo, vamos a asignarles un conjunto de iconos. Así, si la nota está entre 0 y 3 (3 incluído), aparecerá un semáforo en rojo junto a ella; si la nota está entre 3 y 5 (sin incluir ni al 3 ni al 5), el semáforo se pondrá amarillo; y si la nota está por encima de 5 (5 incluído), el semáforo aparecerá en coor verde. La figura muestra cómo hemos definido la regla de formato condicionale que gobierna el comportemineeto de los semáforos junto a las notas.

EXCEL ACTIVIDAD 4.

Abre el archivo llamado "Formato condicional (BACH).xlsx" y realiza los ejercicios propuestos, aplicando los formatos condicionales solicitados.

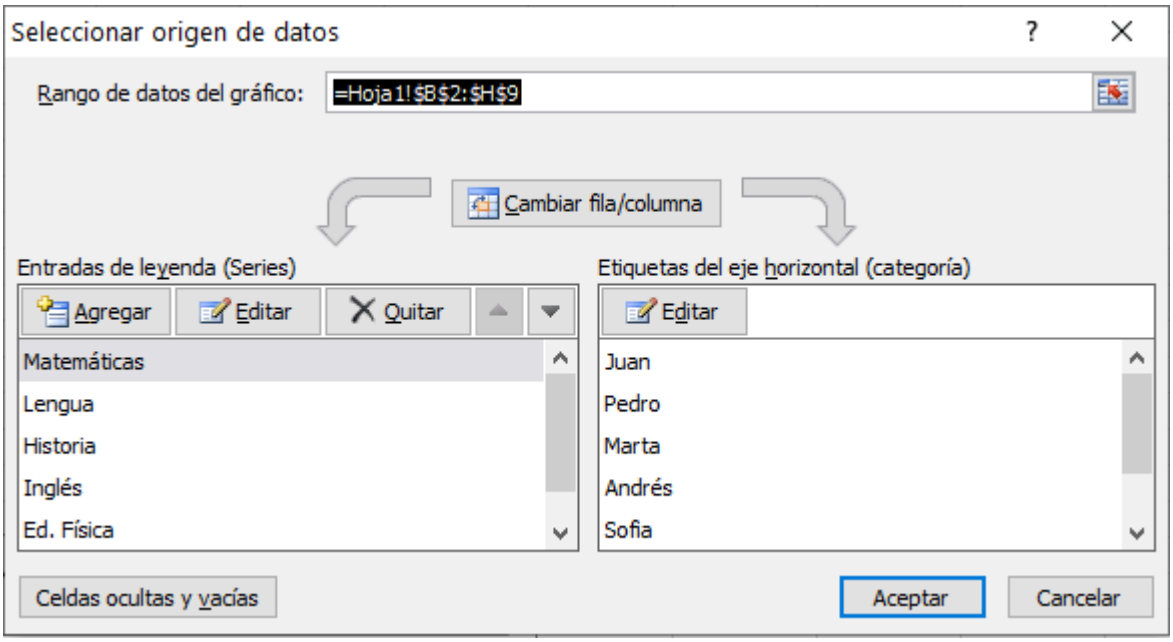
1.5. GRÁFICOS.

Aunque Excel precisa que los datos estén inicialmente en forma de tabla, una vez procesados podemos representarlos visualmente en forma de gráfica para facilitar su análisis. Excel ofrece muchos tipos de gráficas distintas, por lo que deberemos elegir el tipo de gráfica que muestre de forma más clara los datos que deseamos plasmar.

Para construir una gráfica, debemos seguir los siguientes pasos:

1) Seleccionamos el rango de los datos que deseamos representar.
2) Acudimos a la pestaña "Insertar", y en el bloque de "Gráficos" seleccionaos el tipo de gráfica más adecuada para los datos a mostrar. Excel construirá una primera versión de la gráfica, que deberemos editar.
3) Para editar el gráfico preconstruido por Excel basta con seleccionarlo y acudir al menú emergente de "Herramientas de gráficos", que abarca tres pestañas:

- Diseño: En esta pestaña podemos cambiar el tipo de gráfico, así como aplicar algunos estilos de diseño predefinidos, que nosotros no vamos a usar (los haremos todos personalizados). La herramienta más importante es la de "Seleccionar datos", que nos permite añadir series de datos a la gráfica, definir los valores del eje horizontal común, cambiar el nombre de las series, modificar los datos contenidos en esas series, etc.

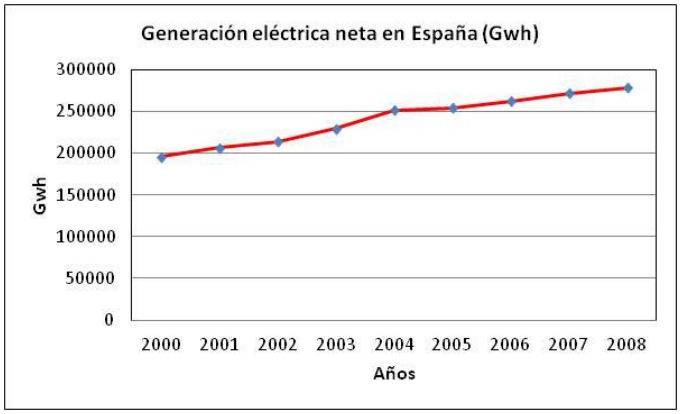


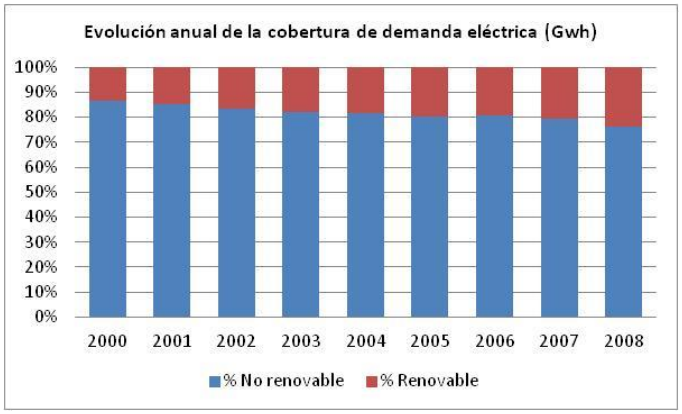
- Presentación: Aquí podemos editar la leyenda del gráfico, añadir rótulos a los ejes, añadir etiquetas a los datos, editar el título del gráfico, modificar la escala de los ejes, etc.
- Formato: En formato podemos cambiar el formato de las series de datos, el color de relleno de las barras o líneas, la separación y el grosor de las barras, y otros efectos.

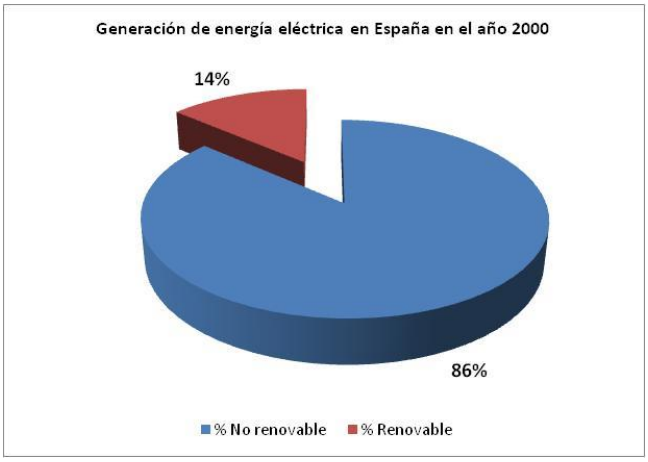
EXCEL ACTIVIDAD 5.

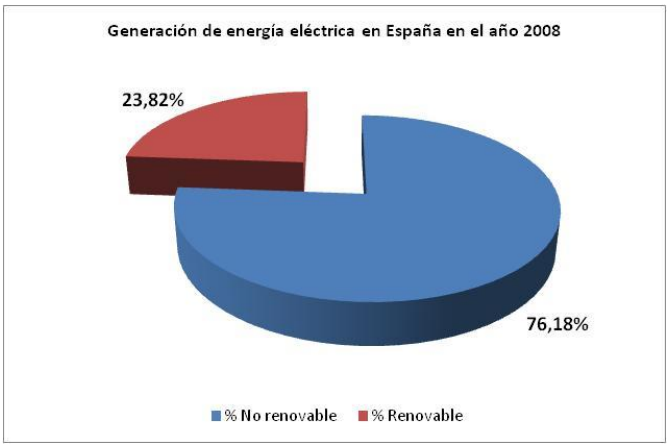
Abre el libro "Ejercicio de gráficas (1) (BACH).xlsx, y completa los ejercicios propuestos.

Hoja 1. Consumo eléctrico. Debes obtener las siguientes gráficas:



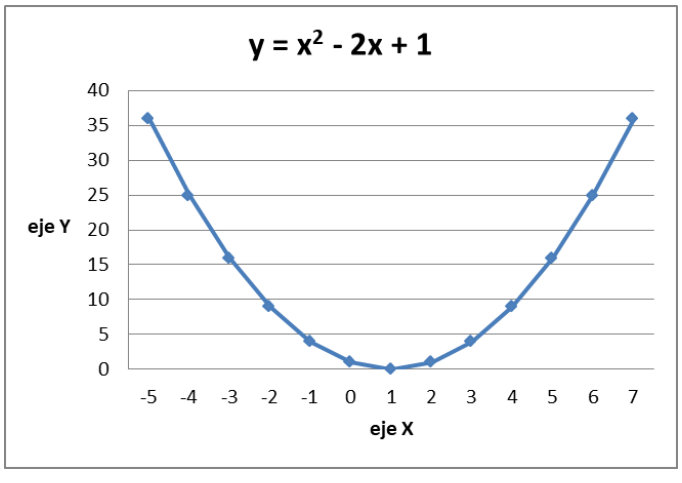


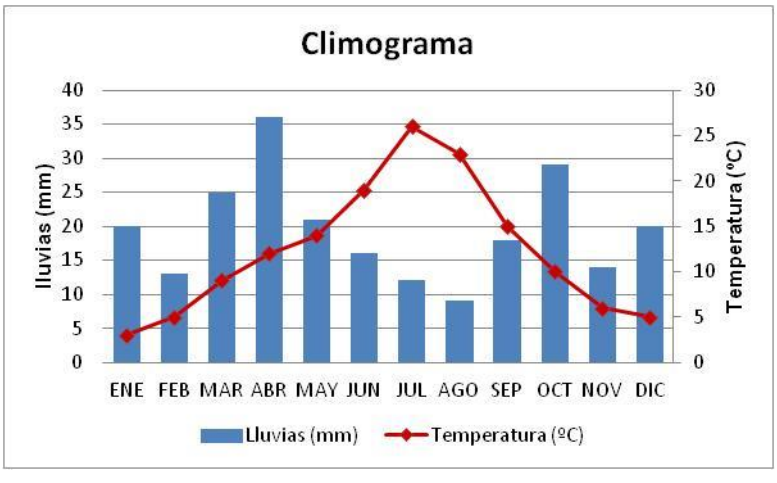




Hoja 2. Funciones.
Representa gráficamente las funciones indicadas, como en el ejemplo de la figura.

Hoja 3. Climograma.
Debes obtener un climograma que represente en la misma gráfica la temperatura y las precipitaciones mensuales, tal como muestra la figura.

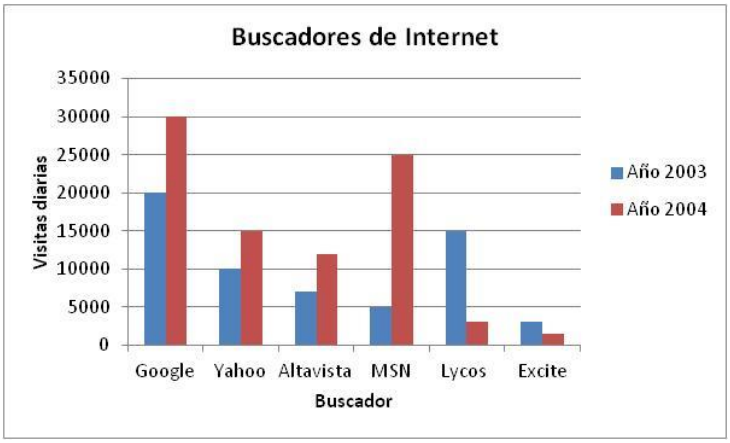


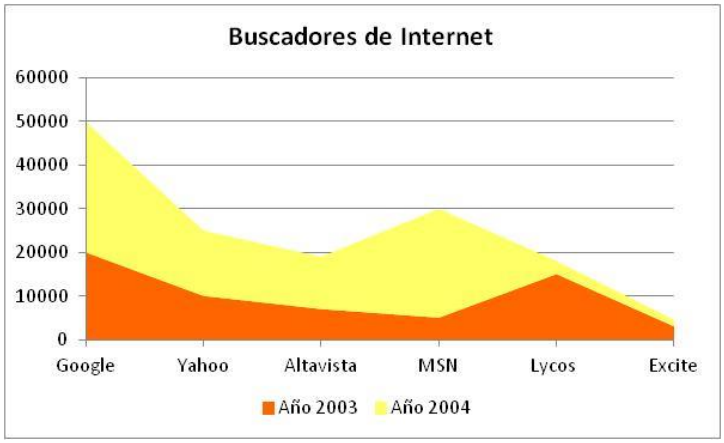


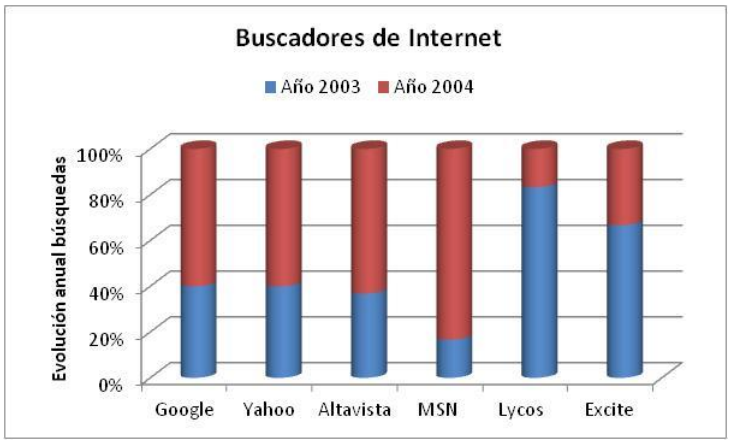
EXCEL ACTIVIDAD 6.

Abre el libro "Ejercicio de gráficas (2) (BACH).xlsx", donde encontrarás una serie de tablas de datos con las que debes generar las siguientes gráficas:

Ejercicio "Buscadores de Internet".

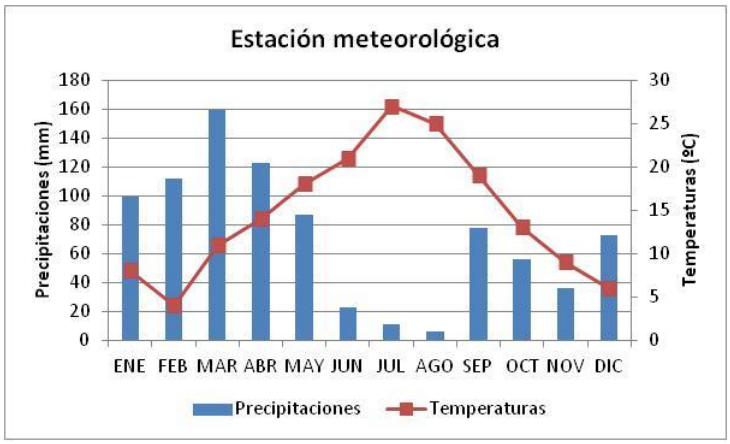




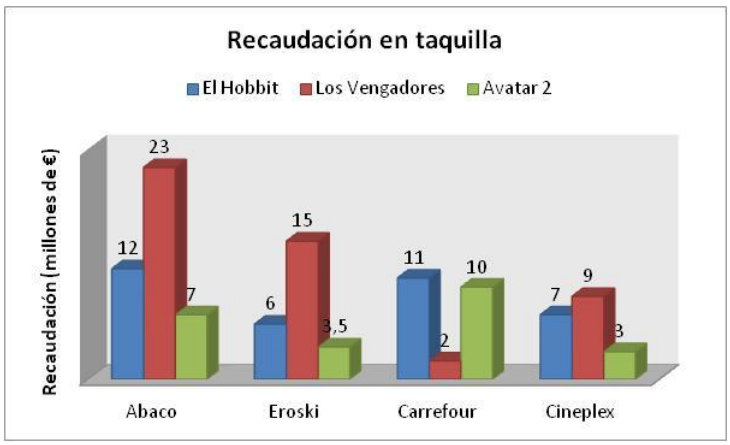


Ejercicio "Elecciones CLM".

Ejercicio "Estación meteorológica"



Ejercicio "Recaudación cines".



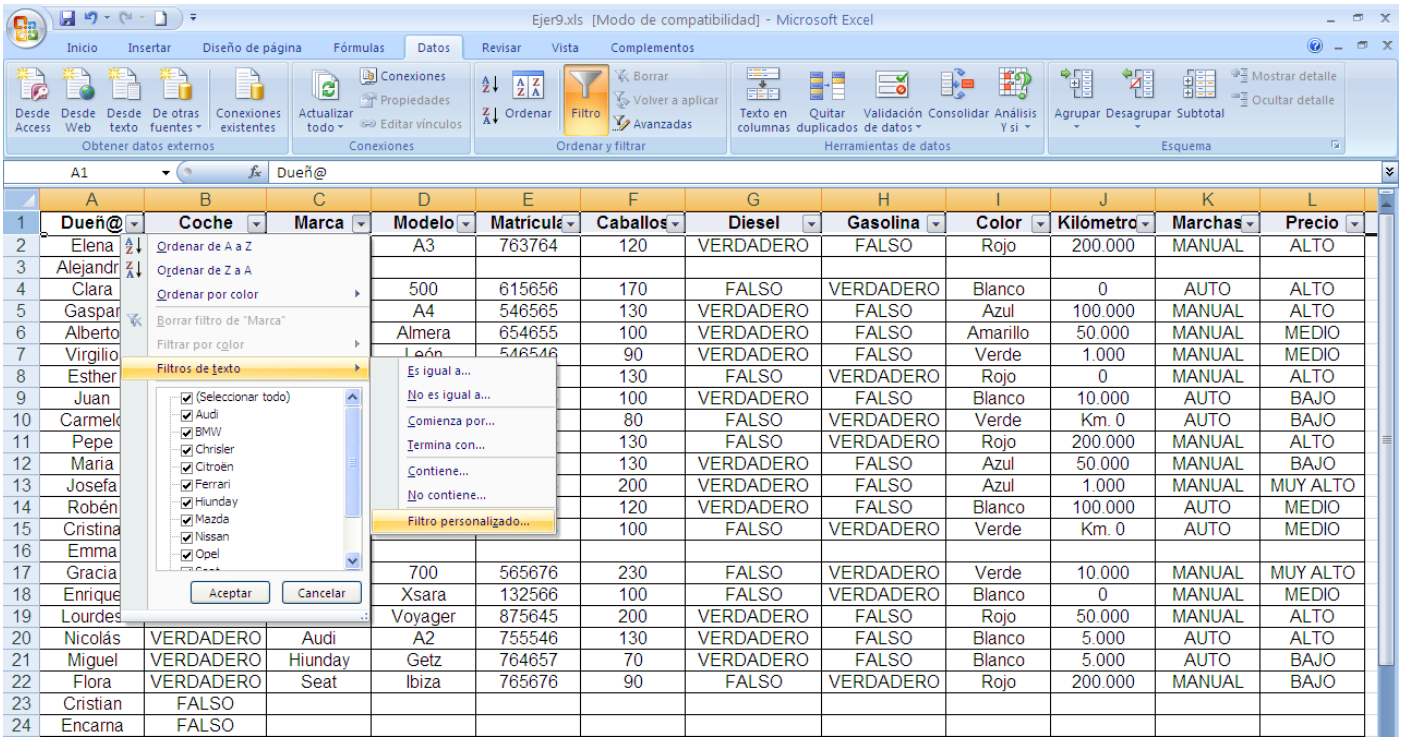
1.6. ORDENAR Y FILTRAR.

Las hojas de cálculo pueden llegar a incluir grandes cantidades de datos. En ocasiones, buscar un dato en una hoja de gran tamaño puede llegar a ser muy complicado. Para buscar datos que cumplan con determinadas condiciones, se utilizan los **filtros**. Los filtros permiten cribar un conjunto de datos según un criterio dado, para obtener los datos buscados. En Excel podemos definir dos tipos de filtros: Los autofiltros y los filtros avanzados.

AUTOFILTROS.

Los **autofiltros** son filtros sencillos que podemos activar de forma rápida y directa, y que ofrecen una potencia de filtrado estándar. Para activar los autofiltros, debemos seguir los siguientes pasos:

a) En primer lugar, seleccionamos la fila de los encabezados de la tabla.
b) A continuación cudimos a la pestaña "Datos", y en el bloque de opciones "Ordenar y filtrar", activamos la herramienta de "Filtros". En ese momento, los encabezados se convierten en menús desplegables donde podemos ordenar los datos y aplicar los filtros deseados, según el tipo de los datos: Es igual a, No es igual a, Empieza por, Termina por, Contiene, No contiene, Superior al promedio, Inferior al promedio, Filtros personalizados, etc.



	A	B	C	D	E	F	G	H	I	J	K	L
1	Dueñ@	Coche	Marca	Modelo	Matricula	Caballos	Diesel	Gasolina	Color	Kilómetro	Marchas	Precio
2	Elena			A3	763764	120	VERDADERO	FALSO	Rojo	200.000	MANUAL	ALTO
3	Alejandr											
4	Clara			500	615656	170	FALSO	VERDADERO	Blanco	0	AUTO	ALTO
5	Gaspar			A4	546565	130	VERDADERO	FALSO	Azul	100.000	MANUAL	ALTO
6	Alberto			Almera	654655	100	VERDADERO	FALSO	Amarillo	50.000	MANUAL	MEDIO
7	Virgilio			León	546546	90	VERDADERO	FALSO	Verde	1.000	MANUAL	MEDIO
8	Esther					130	FALSO	VERDADERO	Rojo	0	MANUAL	ALTO
9	Juan					100	VERDADERO	FALSO	Blanco	10.000	AUTO	BAJO
10	Carmelo					80	FALSO	VERDADERO	Verde	Km. 0	AUTO	BAJO
11	Pepe					130	FALSO	VERDADERO	Rojo	200.000	MANUAL	ALTO
12	Maria					130	VERDADERO	FALSO	Azul	50.000	MANUAL	BAJO
13	Josefa					200	VERDADERO	FALSO	Azul	1.000	MANUAL	MUY ALTO
14	Robén					120	VERDADERO	FALSO	Blanco	100.000	AUTO	MEDIO
15	Cristina					100	FALSO	VERDADERO	Verde	Km. 0	AUTO	MEDIO
16	Emma											
17	Gracia			700	565676	230	FALSO	VERDADERO	Verde	10.000	MANUAL	MUY ALTO
18	Enrique			Xsara	132566	100	FALSO	VERDADERO	Blanco	0	MANUAL	MEDIO
19	Lourdes			Voyager	875645	200	VERDADERO	FALSO	Rojo	50.000	MANUAL	ALTO
20	Nicolás	VERDADERO	Audi	A2	755546	130	VERDADERO	FALSO	Blanco	5.000	AUTO	ALTO
21	Miguel	VERDADERO	Hiunday	Getz	764657	70	VERDADERO	FALSO	Blanco	5.000	AUTO	BAJO
22	Flora	VERDADERO	Seat	Ibiza	765676	90	FALSO	VERDADERO	Rojo	200.000	MANUAL	BAJO
23	Cristian	FALSO										
24	Encarna	FALSO										

Cuando un filtro está activo, la flecha del menú desplegable en el encabezado muestra el icono de un embudo. Para deshacer el filtro, seleccionamos la opción "Borrar filtro" del menú desplegable.

	A	B	C	D	E	F	G
1	Dueñ@	Coche	Marca	Modelo	Matricula	Caballos	Diesel
4	Clara	VERDADERO	BMW	500	615656	170	FALSO
11	Pepe	VERDADERO	BMW		978655	130	FALSO
17	Gracia	VERDADERO	BMW	700	565676	230	FALSO
34	Sandra	VERDADERO	BMW	500	214555	250	VERDADERO

Marca:
Igual a "BMW"

EXCEL ACTIVIDAD 7.

Abre el archivo Excel "Filtros (1) (BACH).xlsx" y realiza las actividades propuestas en la Hoja de EJERCICIOS.

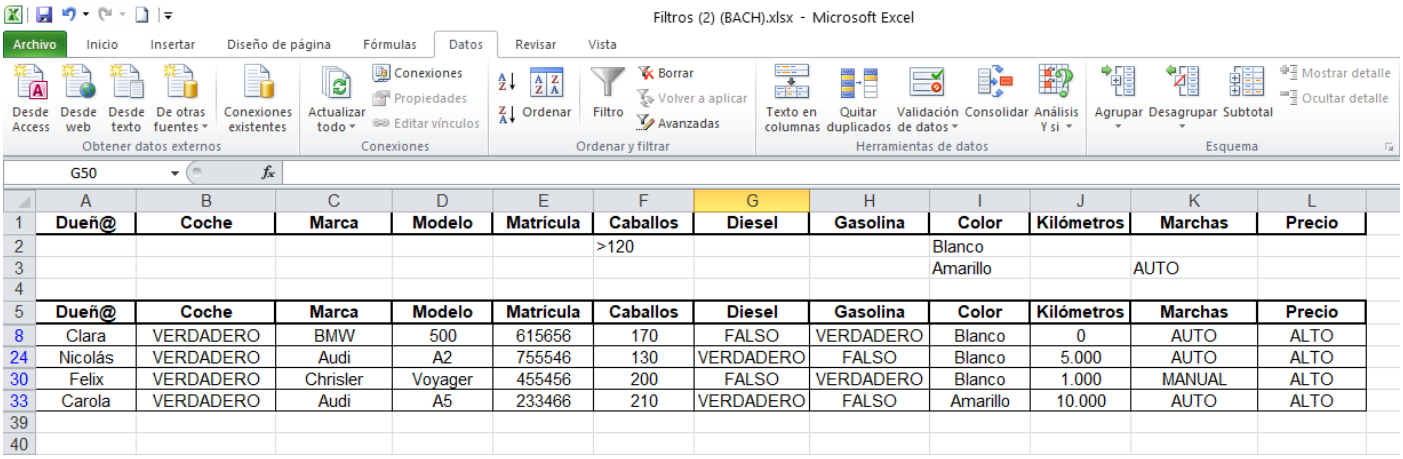
FILTROS AVANZADOS.

Cuando necesitemos filtrar la información de forma personalizada, sin tener que acudir los filtros predefinidos por Excel, podemos usar los **filtros avanzados**. Para poder usar filtros avanzados debemos definir en nuestra hoja Excel dos zonas bien diferenciadas:

1) **Rango de criterios**: Zona de la hoja que ocupa las primeras líneas, donde se establecerán los criterios de filtrado definidos por el usuario.
2) **Lista de datos**: Es la zona de la hoja donde residirán los datos propiamente dichos.

Para definir un filtro avanzado, debemos seguir los siguientes pasos:

a) Dejamos 4 filas libres por encima de la fila de encabezados.
b) Copiamos la fila de encabezados en la primera fila de la hoja. De esta forma creamos el rango de criterios.
c) Establecemos los criterios de búsqueda en la segunda fila, debajo del campo que nos interese. Los criterios pueden ser muchos y muy variados, pero hay una regla clara: Si los criterios están en la misma fila es un "Y", y si los criterios están en distinta fila es un "O". En la figura hemos elegido mostrar los coches de color blanco Y de más de 120 caballos O los coches amarillos Y con cambio automático.
d) Acudimos a la pestaña "Datos", y en el bloque de opciones "Ordenar y filtrar", seleccionamos la herramienta "Avanzadas". A continuación seleccionamos el rango de la lista de datos y el rango de los criterios (y establecemos también el rango donde copiar el resultado en el caso de haber seleccionado la opción "Copiar a otro lugar").



	A	B	C	D	E	F	G	H	I	J	K	L
1	Dueñ@	Coche	Marca	Modelo	Matrícula	Caballos	Diesel	Gasolina	Color	Kilómetros	Marchas	Precio
2						>120			Blanco			
3									Amarillo		AUTO	
4												
5	Dueñ@	Coche	Marca	Modelo	Matrícula	Caballos	Diesel	Gasolina	Color	Kilómetros	Marchas	Precio
8	Clara	VERDADERO	BMW	500	615656	170	FALSO	VERDADERO	Blanco	0	AUTO	ALTO
24	Nicolás	VERDADERO	Audi	A2	755546	130	VERDADERO	FALSO	Blanco	5.000	AUTO	ALTO
30	Felix	VERDADERO	Chrisler	Voyager	455456	200	FALSO	VERDADERO	Blanco	1.000	MANUAL	ALTO
33	Carola	VERDADERO	Audi	A5	233466	210	VERDADERO	FALSO	Amarillo	10.000	AUTO	ALTO
39												
40												

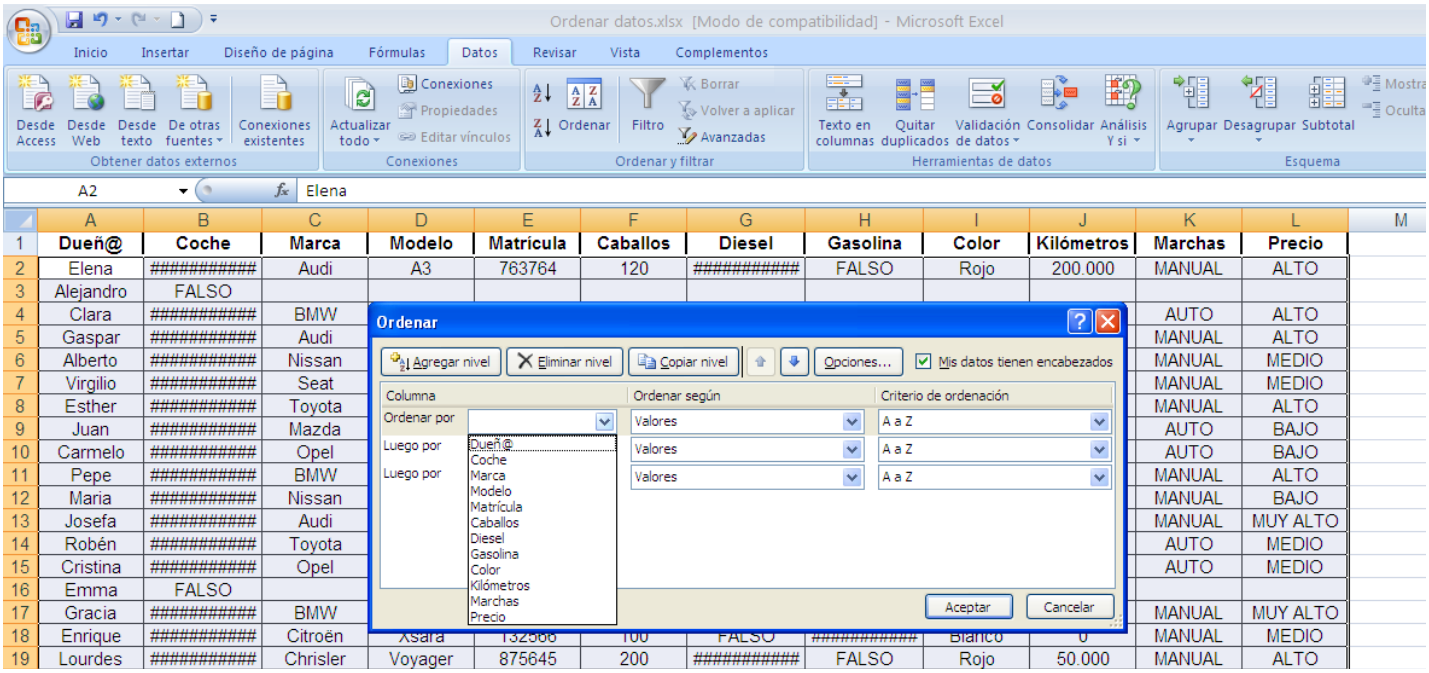
EXCEL ACTIVIDAD 8.

Abre el archivo Excel "Filtros (2) (BACH).xlsx" y realiza las actividades propuestas en la Hoja de EJERCICIOS.

ORDENAR DATOS.

En las tablas que construímos en Excel, las filas de datos se muestran en el mismo orden en el que las hemos introducido. La mayoría de las veces ese orden no guarda ninguna relación con ninguna de las columnas de la tabla, lo que dificulta la localización de un dato en concreto. Por el contrario, si los datos están ordenados en base a alguna columna, encontrar un dato resulta mucho más fácil. (Por ejemplo, las

guías telefónicas están ordenadas por apellidos). Para ordenar una colección de datos acudimos a la pestaña de "Datos", y en el bloque "Ordenar y filtrar", seleccionamos la herramienta "Ordenar".



	A	B	C	D	E	F	G	H	I	J	K	L
1	Dueñ@	Coche	Marca	Modelo	Matrícula	Caballos	Diesel	Gasolina	Color	Kilómetros	Marchas	Precio
2	Elena	##########	Audi	A3	763764	120	##########	FALSO	Rojo	200.000	MANUAL	ALTO
3	Alejandro	FALSO										
4	Clara	##########	BMW								AUTO	ALTO
5	Gaspar	##########	Audi								MANUAL	ALTO
6	Alberto	##########	Nissan								MANUAL	MEDIO
7	Virgilio	##########	Seat								MANUAL	MEDIO
8	Esther	##########	Toyota								MANUAL	ALTO
9	Juan	##########	Mazda								AUTO	BAJO
10	Carmelo	##########	Opel								AUTO	BAJO
11	Pepe	##########	BMW								MANUAL	ALTO
12	Maria	##########	Nissan								MANUAL	BAJO
13	Josefa	##########	Audi								MANUAL	MUY ALTO
14	Robén	##########	Toyota								AUTO	MEDIO
15	Cristina	##########	Opel								AUTO	MEDIO
16	Emma	FALSO										
17	Gracia	##########	BMW								MANUAL	MUY ALTO
18	Enrique	##########	Citroën	Xsara	132566	100	FALSO	##########	Blanco	0	MANUAL	MEDIO
19	Lourdes	##########	Chrisler	Voyager	875645	200	##########	FALSO	Rojo	50.000	MANUAL	ALTO

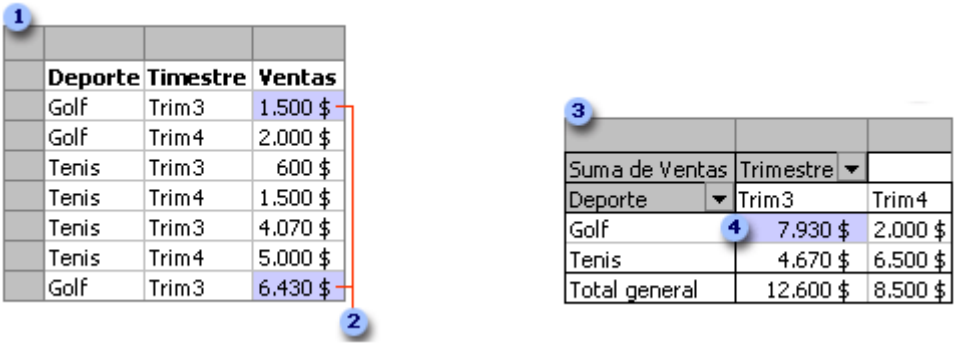
EXCEL ACTIVIDAD 9.

Abre el archivo "Ordenar datos (BACH).xlsx" y usa la herramienta de ordenar datos para hacer los ejercicios planteados.

1.7. TABLAS Y GRÁFICOS DINÁMICOS.

TABLAS DINÁMICAS.

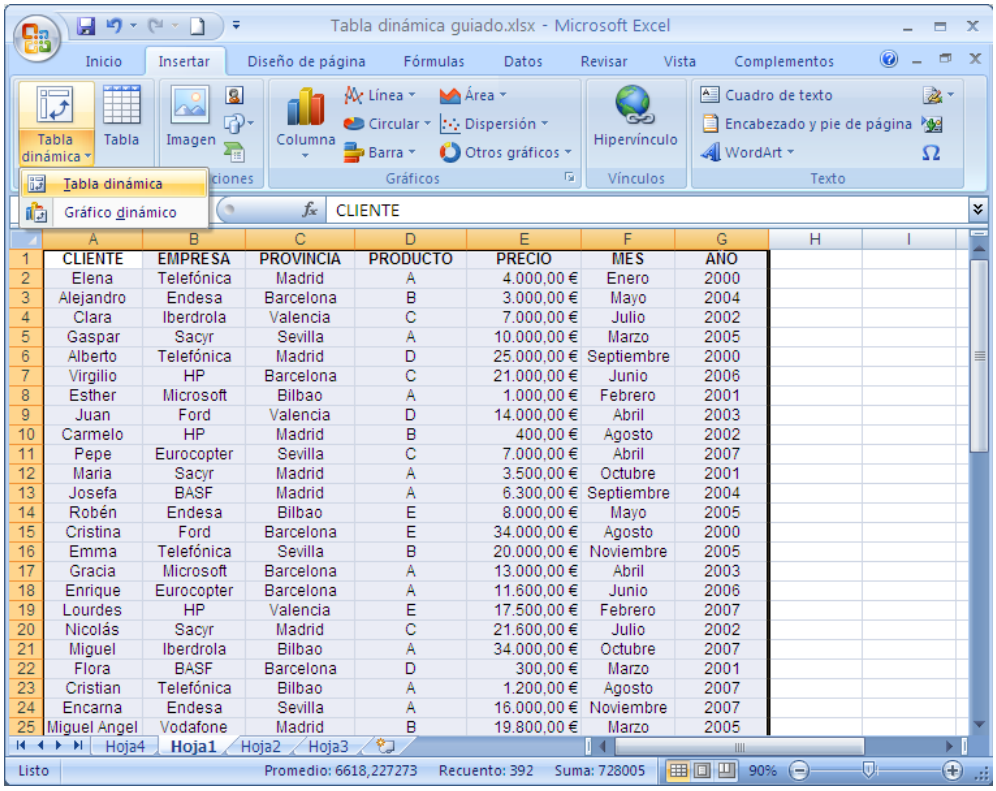
Una **tabla dinámica** es un resumen de un conjunto de datos, atendiendo a varios criterios de agrupación, y representado como una tabla de doble entrada que nos facilita la interpretación de dichos datos. Se denomina "dinámica" porque nos permite obtener diferentes totales, filtrar datos, cambiar la presentación de los datos, visualizar o no los datos de origen, etc. La figura muestra una tabla normal a la izquierda, y una tabla dinámica a la derecha.



Para crear una tabla dinámica, debemos seguir los siguientes pasos:

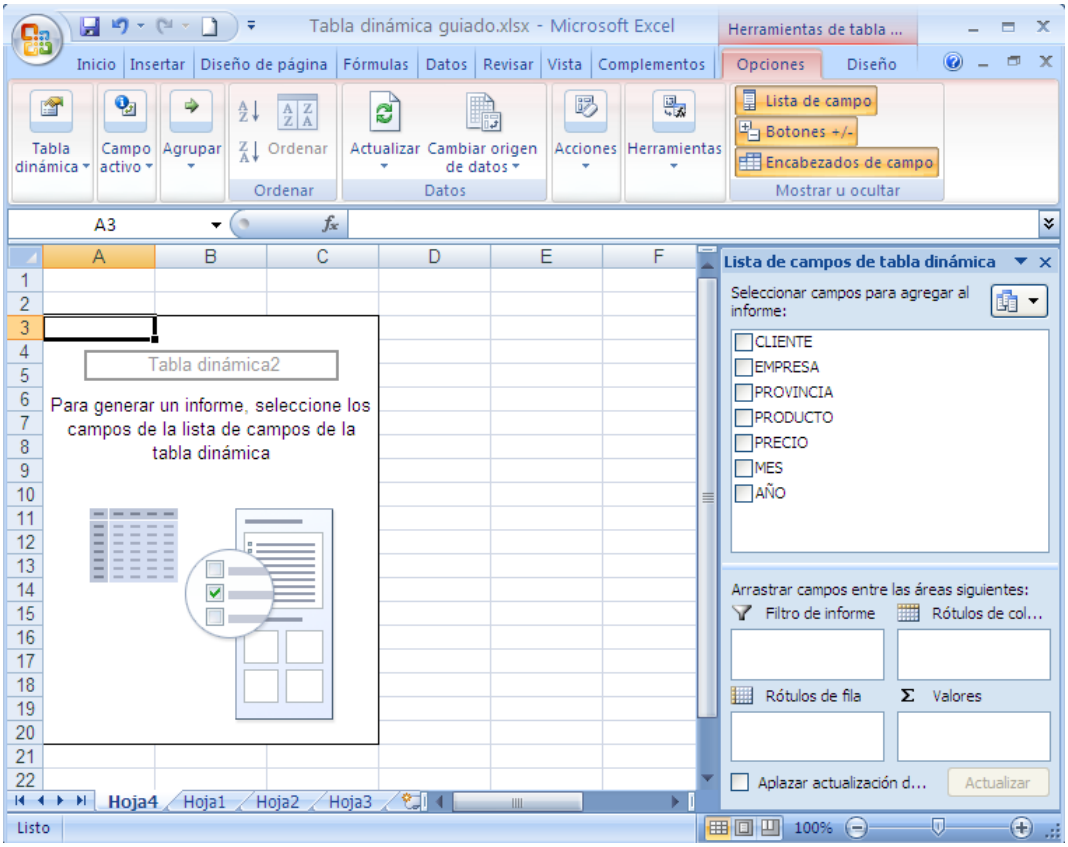
a) En pestaña "Insertar", en el bloque de opciones "Tablas", seleccionamos la herramienta "Tabla dinámica".
b) Seleccionamos los datos para crear la tabla dinámica y el lugar donde ubicarla (mejor en una nueva hoja).
c) A continuación, se abre el asistente para construir la tabla dinámica.

d) En el asistente repartimos los campos de la tabla original en filas, columnas, valores, y filtros, para configurar la estructura de la tabla dinámica.



	A	B	C	D	E	F	G
1	CLIENTE	EMPRESA	PROVINCIA	PRODUCTO	PRECIO	MES	AÑO
2	Elena	Telefónica	Madrid	A	4.000,00 €	Enero	2000
3	Alejandro	Endesa	Barcelona	B	3.000,00 €	Mayo	2004
4	Clara	Iberdrola	Valencia	C	7.000,00 €	Julio	2002
5	Gaspar	Sacyr	Sevilla	A	10.000,00 €	Marzo	2005
6	Alberto	Telefónica	Madrid	D	25.000,00 €	Septiembre	2000
7	Virgilio	HP	Barcelona	C	21.000,00 €	Junio	2006
8	Esther	Microsoft	Bilbao	A	1.000,00 €	Febrero	2001
9	Juan	Ford	Valencia	D	14.000,00 €	Abril	2003
10	Carmelo	HP	Madrid	B	400,00 €	Agosto	2002
11	Pepe	Eurocopter	Sevilla	C	7.000,00 €	Abril	2007
12	Maria	Sacyr	Madrid	A	3.500,00 €	Octubre	2001
13	Josefa	BASF	Madrid	A	6.300,00 €	Septiembre	2004
14	Robén	Endesa	Bilbao	E	8.000,00 €	Mayo	2005
15	Cristina	Ford	Barcelona	E	34.000,00 €	Agosto	2000
16	Emma	Telefónica	Sevilla	B	20.000,00 €	Noviembre	2005
17	Gracia	Microsoft	Barcelona	A	13.000,00 €	Abril	2003
18	Enrique	Eurocopter	Barcelona	A	11.600,00 €	Junio	2006
19	Lourdes	HP	Valencia	E	17.500,00 €	Febrero	2007
20	Nicolás	Sacyr	Madrid	C	21.600,00 €	Julio	2002
21	Miguel	Iberdrola	Bilbao	A	34.000,00 €	Octubre	2007
22	Flora	BASF	Barcelona	D	300,00 €	Marzo	2001
23	Cristian	Telefónica	Bilbao	A	1.200,00 €	Agosto	2007
24	Encarna	Endesa	Sevilla	A	16.000,00 €	Noviembre	2007
25	Miguel Angel	Vodafone	Madrid	B	19.800,00 €	Marzo	2005

Tabla origen.



Asistente de tablas dinámicas.

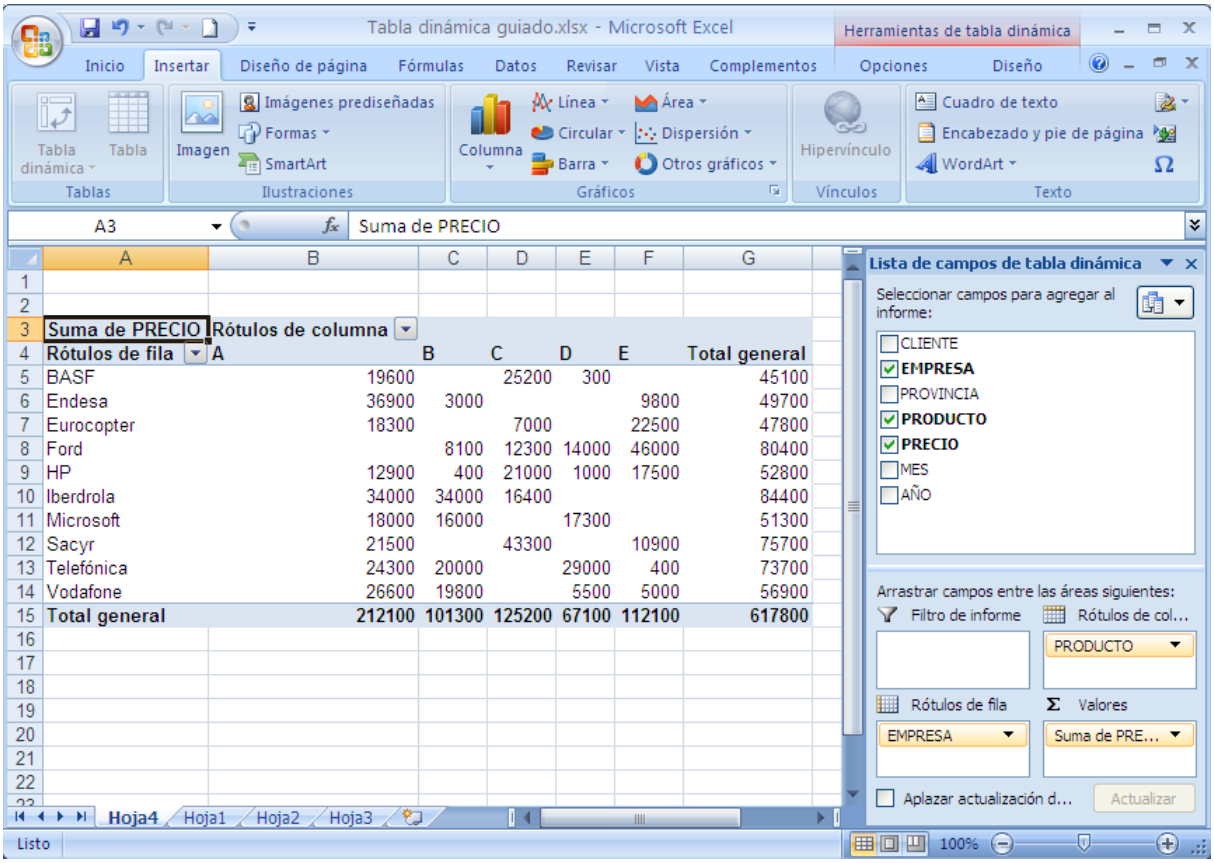
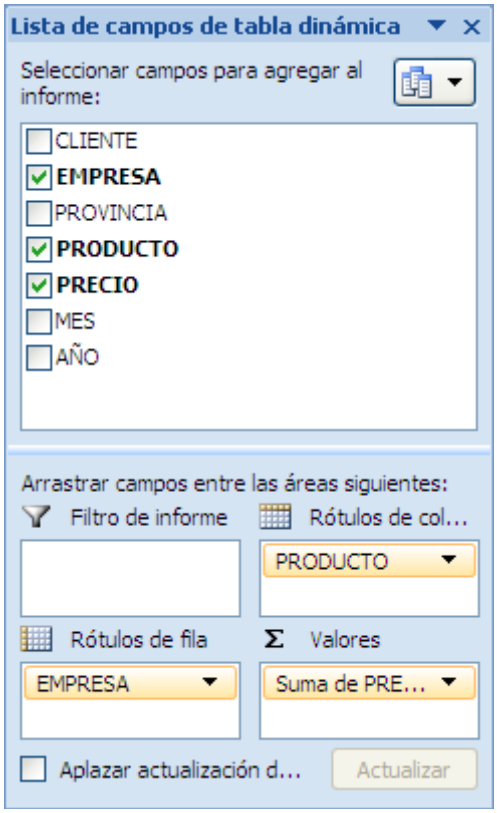


Tabla dinámica ya construida.

EXCEL ACTIVIDAD 10.

Abre el archivo "Tabla dinámica guiado (BACH).xlsx, y sigue los siguientes pasos para aprender a crear una tabla dinámica:

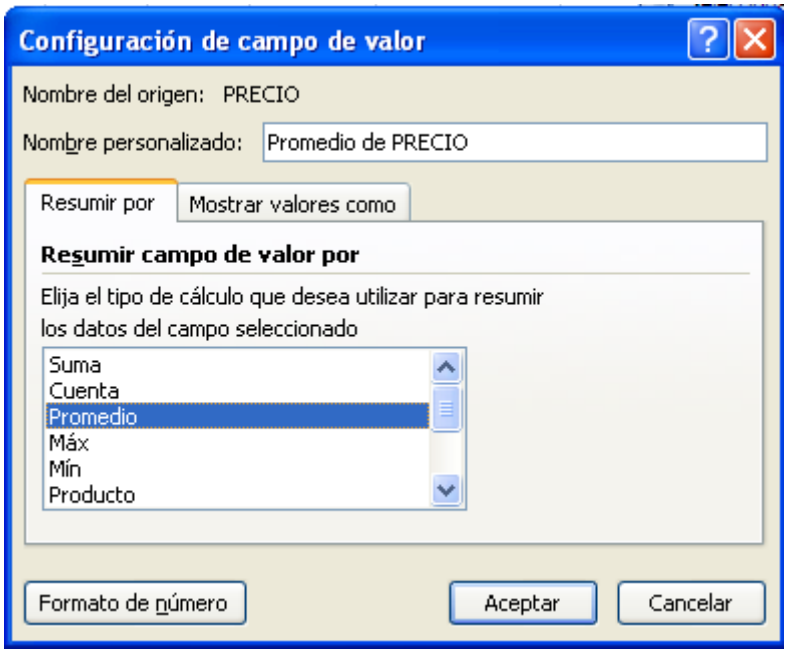
1) Selecciona todos los datos de la tabla original.
2) Crea una tabla dinámica en una hoja nueva, con la siguiente estructura:



3) Si lo has hecho correctamente, la tabla dinámica presentará la siguiente apariencia:

Suma de PRECIO	Rótulos de columna					
Rótulos de fila	A	B	C	D	E	Total general
BASF	19600		25200	300		45100
Endesa	36900	3000			9800	49700
Eurocopter	18300		7000		22500	47800
Ford		8100	12300	14000	46000	80400
HP	12900	400	21000	1000	17500	52800
Iberdrola	34000	34000	16400			84400
Microsoft	18000	16000		17300		51300
Sacyr	21500		43300		10900	75700
Telefónica	24300	20000		29000	400	73700
Vodafone	26600	19800		5500	5000	56900
Total general	**212100**	**101300**	**125200**	**67100**	**112100**	**617800**

4) Modifica el diseño de la tabla dinámica para que en las columnas aparezca la provincia destino del producto.
5) Modifica la tabla dinámica para mostrar precios de venta por provincia y por mes de venta.
6) Modifica la tabla para obtener el promedio de precio de ventas, por provincia y por mes.



7) Si lo has hecho correctamente, la tabla dinámica debe quedar así:

Promedio de PRECIO	Rótulos de columna					
Rótulos de fila	Barcelona	Bilbao	Madrid	Sevilla	Valencia	Total general
Enero			9375			9375
Febrero		4666,666667	8000		17500	7900
Marzo	1050		19800	11150		8840
Abril	13150		12000	7000	14000	11860
Mayo	8350	8000	21700			11600
Junio	12700				20900	14750
Julio	12000	20350	21600		7000	16260
Agosto	34000	1200	1766,666667	14600		9183,333333
Septiembre	21000	15900	15650		4000	14440
Octubre	5850	34000	3250			10440
Noviembre	17300		900	18000	2000	11240
Diciembre				10900	4000	7450
Total general	**11946,66667**	**12644,44444**	**9682,352941**	**12971,42857**	**9914,285714**	**11232,72727**

8) Modifica de nuevo la tabla dinámica para mostrar la suma de precios de venta por compañía y producto, para la provincia de Madrid.

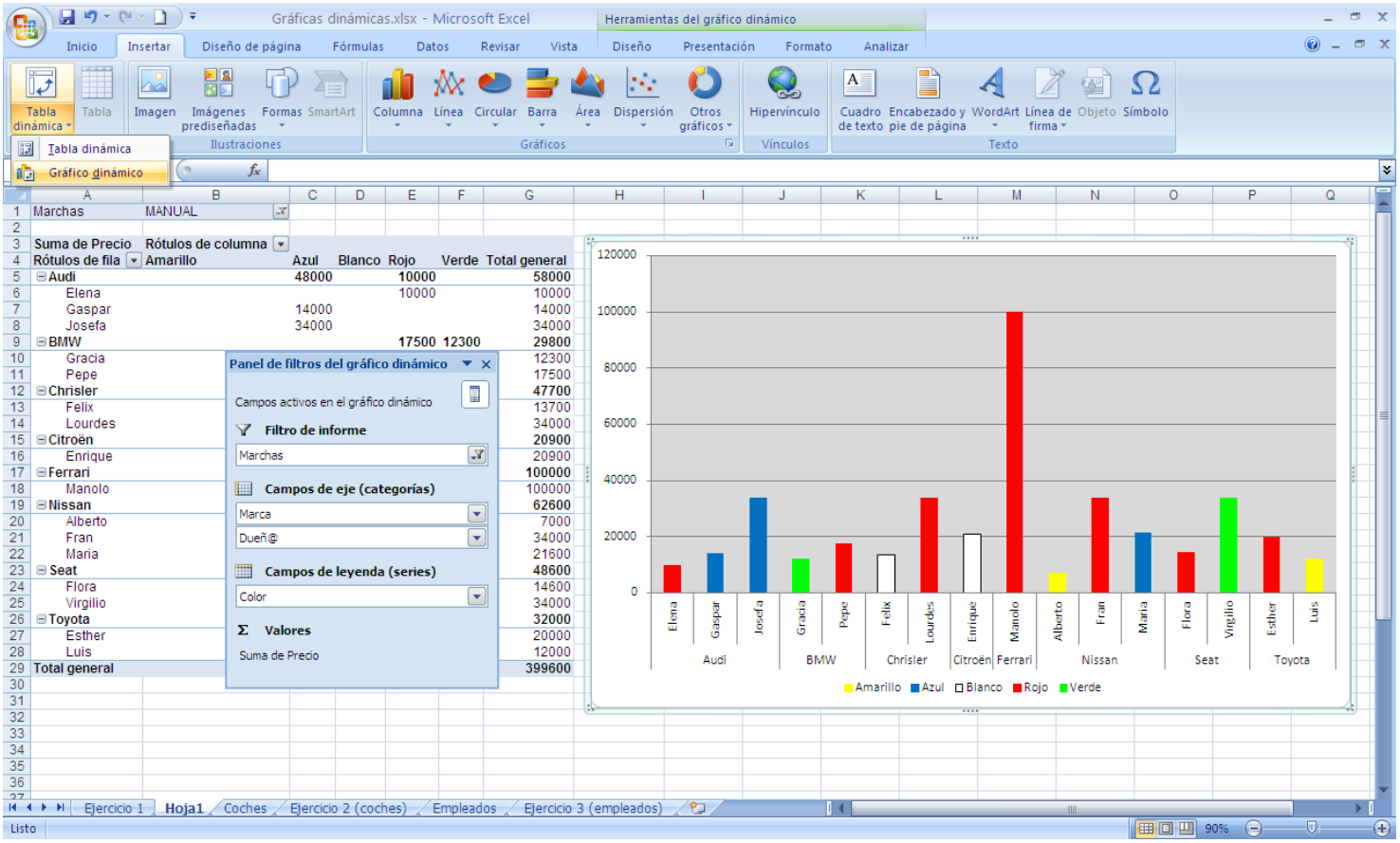
PROVINCIA	Madrid					
Suma de PRECIO	Rótulos de columna					
Rótulos de fila	A	B	C	D	E	Total general
BASF	6300					6300
Ford		8100			12000	20100
HP	900	400				1300
Iberdrola			9400			9400
Microsoft		16000				16000
Sacyr	11500		43300			54800
Telefónica	8500			25000	400	33900
Vodafone		19800			3000	22800
Total general	**27200**	**44300**	**52700**	**25000**	**15400**	**164600**

EXCEL ACTIVIDAD 11.

Abre el archivo "Tabla dinámica (BACH).xlsx", y realiza los ejercicios propuestos utilizando tablas dinámicas. Acuérdate de guardar el archivo al terminar.

GRÁFICOS DINÁMICOS.

Los datos dinámicos pueden presentarse en forma de tablas (tablas dinámicas), o de manera gráfica (gráficos dinámicos). Para crear un **gráfico dinámico**, procedemos de forma similar a como lo hacemos con las tablas dinámicas: En la pestaña "Insertar" acudimos al bloque de opciones "Tablas", y seleccionamos la herramienta "Gráfico dinámico". Siempre que creemos un gráfico dinámico, también se creará su tabla dinámica asociada. De hecho, el diseño de los gráficos dinámicos es idéntico al de las tablas dinámicas, sólo que al mismo tiempo que se configura la tabla dinámica asociada, se va generando el gráfico dinámico correspondiente.

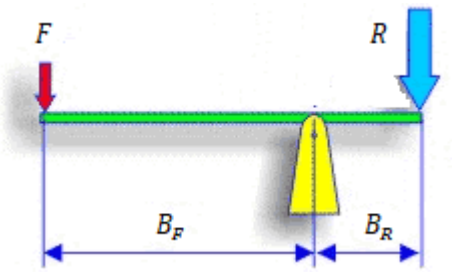


EXCEL ACTIVIDAD 12.

Obtén los gráficos dinámicos derivados de las tablas dinámicas obtenidas en la actividad 11. Responde a las cuestiones propuestas, esta vez utilizando los gráficos dinámicos. Guarda el archivo como "Gráficos dinámicos (BACH).xlsx".

1.8. EXCEL APLICADO A LAS ASIGNATURAS DE TECNOLOGÍAS, MATEMÁTICAS, Y FÍSICA.

LEY DE LA PALANCA.



La **ley de la palanca** es una ecuación que resume el comportamiento de una palanca. En esta ecuación F es la **fuerza aplicada** sobre la palanca; B_F es el **brazo de fuerza** (distancia del punto donde se aplica la fuerza al punto de apoyo); R es la **resistencia**, esto es, la fuerza que queremos vencer; y B_R es el **brazo de resistencia** (distancia del punto donde se ubica la resistencia al punto de apoyo). En palabras, esta ecuación afirma que, cuando una palanca está equilibrada, los productos $F \times B_F$ y $R \times B_R$ deben ser iguales:

$$F \times B_F = R \times B_R$$

Como toda ecuación, si conocemos tres de las cantidades involucradas, podemos despejarla para hallar la cuarta. Pero el verdadero contenido teórico de la ecuación aparece cuando resolvemos para la fuerza aplicada:

$$F = \frac{R \times B_R}{B_F}$$

Como en toda máquina, el objetivo de una palanca es minimizar la fuerza que necesitamos aplicar para vencer la resistencia. Como vemos en la ecuación, para minimizar la fuerza aplicada debemos hacer el brazo de fuerza grande y el brazo de resistencia pequeño. En otras palabras, cuanto más grande sea el brazo de fuerza y más pequeño el brazo de resistencia, menor será la fuerza que debemos aplicar para vencer una cierta resistencia.

La **ventaja mecánica** es la relación entre la resistencia a vencer y la fuerza aplicada. La ventaja mecánica permite medir el grado de efectividad de una palanca, y nos sirve para comparar distintas palancas. Si la ventaja en mayor que uno, la palanca permite reducir el esfuerzo necesario. Si la ventaja es igual a uno, la palanca ni reduce ni aumenta el esfuerzo. Por último, si la ventaja es menor que uno, la palanca no solo no disminuye esfuerzo, sino que aumenta la fuerza que necesitaríamos aplicar para vencer la resistencia. Por supuesto, la mejor palanca es aquella que tenga un valor más alto de ventaja mecánica.

$$VM = \frac{R}{F} = \frac{B_F}{B_R}$$

EXCEL ACTIVIDAD 13.

Crea una hoja Excel que permita calcular la fuerza que debemos aplicar para vencer una determinada resistencia dada, introduciendo como datos el tamaño de la palanca (brazo de fuerza y brazo de resistencia) así como la resistencia a vencer. Recuerda que la fuerza y la resistencia se miden en newtons, y los brazos en metros. La hoja debe calcular también la ventaja mecánica (el rendimiento de la palanca), y decidir si esa palanca permite reducir, mantener, o aumentar la fuerza aplicada. Añade un formato atractivo a la tabla, que incluya imágenes y figuras para ilustrarla.

ANÁLISIS DE CIRCUITOS.

El objetivo de la siguiente práctica será crear una tabla Excel que permita resolver **circuitos serie** de hasta 5 resistores. Como datos de partida proporcionaremos el voltaje total de la pila (V_T) y el valor de resistencia de cada resistor (R_1, R_2, R_3, R_4, y R_5). A partir de estos datos deberemos calcular:

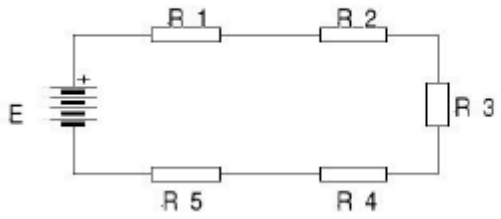
- Resistencia total del circuito serie ($R_T = R_1 + R_2 + R_3 + R_4 + R_5$).
- Corriente total circulante, en mA ($I_T = V_T / R_T$). Recordar que $1\ mA = 10^{-3}\ A = 0{,}001\ A$.
- Potencia total consumida, en mW ($P_T = V_T \times I_T$). Recordar que $1\ mW = 10^{-3}\ W = 0{,}001\ W$.
- Voltaje en cada resistor en V ($V_1 = I_T \times R_1$, $V_2 = I_T \times R_2$, etc.).
- Intensidad en cada resistor, en mA (recordar que en los circuitos serie, la intensidad es la misma para todos los resistores, e igual a la total).
- Potencia en cada resistor, en mW ($P_1 = V_1 \times I_1$, $P_2 = V_2 \times I_2$, etc.).

EXCEL ACTIVIDAD 14.

Diseña una hoja Excel que te permita analizar un circuito serie de hasta 5 resistores, a partir de los valores de resistencia de esos resistores y del voltaje total de la pila. Redondea todos los resultados a 2 decimales, y aplica un formato similar al mostrado en la figura. La figura del circuito la puedes dibujar tú mismo en Crocodile, Paint, etc., o buscarlo en Internet. Notar que esta tabla también debería funciona para analizar, por ejemplo, un circuito de solo tres resistores serie. Para ello, bastaría con fijar a $0\ \Omega$ el valor de resistencia de los dos últimos resistores.

CALCULADOR AUTOMATICO DE CIRCUITOS EN SERIE

DATOS	
E (voltios)	24
R1(KΩ)	4,7
R2(KΩ)	4,7
R3(KΩ)	12
R4(KΩ)	0,68
R5(KΩ)	24



VALORES GLOBALES	
Resistencia	46,08
Corriente(mA)	0,52
Potencia(mW)	12,5

VALORES PARCIALES

Componente	Resistencia	Tensión(V)	Corriente(mA)	Potencia(mW)
R1	4,7	2,45	0,52	1,27
R2	4,7	2,45	0,52	1,27
R3	12	6,25	0,52	3,26
R4	0,68	0,35	0,52	0,18
R5	24	12,5	0,52	6,51
TOTALES:	46,08	24	0,52	12,5

EXCEL ACTIVIDAD 15.

Diseña una hoja Excel similar a la anterior, pero que te permita resolver **circuitos paralelos** de 3 resistores. Para ello necesitas las siguientes fórmulas:

$$V_1 = V_2 = V_3 = V_4 = V_5 = V_T$$

$$I_1 = \frac{V_1}{R_1} = \frac{V_T}{R_1}$$

$$I_2 = \frac{V_2}{R_2} = \frac{V_T}{R_2}$$

$$I_3 = \frac{V_3}{R_3} = \frac{V_T}{R_3}$$

$$\frac{1}{R_T} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3}$$

Las potencias se calculan de forma idéntica al caso del circuito serie. En este caso, la tabla no sirve para analizar circuitos paralelo con menos resistores (fijar a $0\ \Omega$ un resistor equivale a crear un cortocircuito).

RESOLUCIÓN DE ECUACIONES.

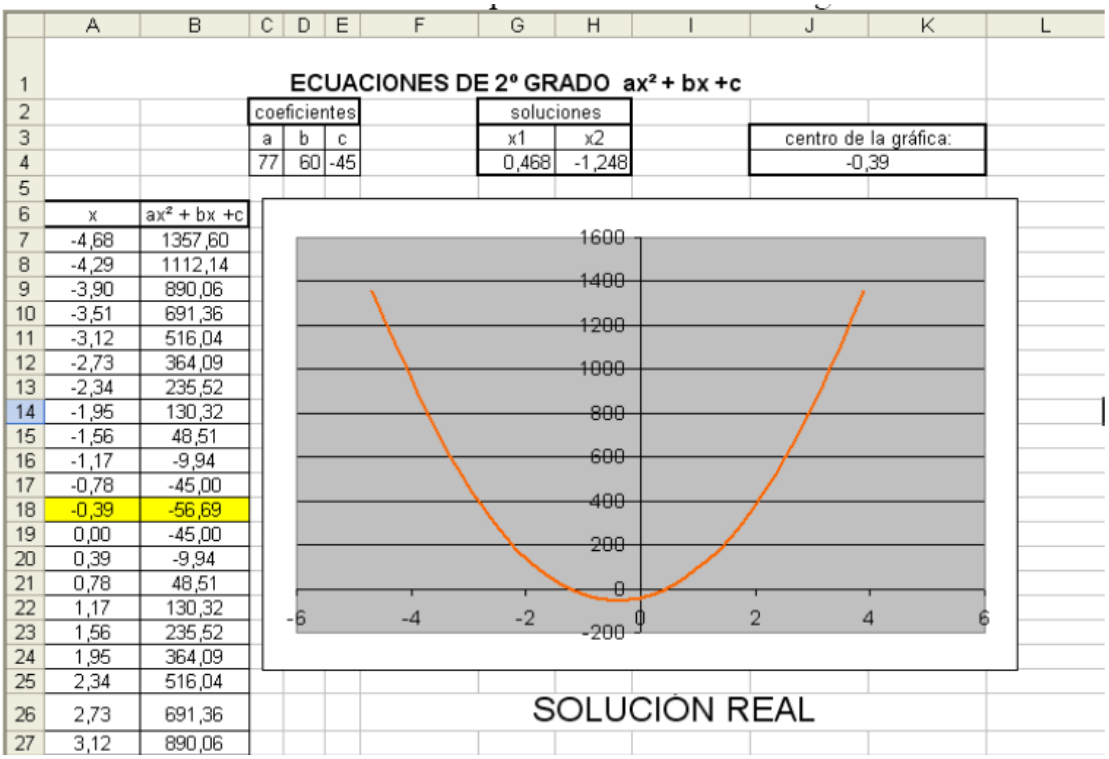
Vamos a construir unas tablas para resolver ecuaciones cuadráticas y sistemas de ecuaciones lineales.

EXCEL ACTIVIDAD 16.

En Esta actividad debes realziar una hoja de cálculo que obtenga las raíces reales de una **ecuación cuadrática** de la forma $ax^2 + bx + c = 0$ (donde a, b, y c son los coeficientes que acompañan al término cuadrático, al término lineal, y el término independiente, y serán números reales conocidos). Como sabemos, las soluciones vienen dadas por la **fórmula cuadrática**:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Recordar que la cantidad $\nabla = b^2 - 4ac$ dentro de la raíz se denomina **discriminante**, y nos permite saber si la ecuación cuadrática dispone o no de soluciones reales. Si $\nabla > 0$ la ecuación tendrá dos raíces reales, si $\nabla = 0$ la ecuación tendrá una sola raíz real (que se dice que es doble), y si $\nabla < 0$ la ecuación no tiene ninguna solución real. Tu hoja de cálculo también debe ser capaz de discernir cuándo una ecuación cuadrática carece de raíces reales. Por último, la hoja de cálculo debe representar el gráfico de la parábola que se corresponde con la ecuación indicada. Las soluciones equivalen a los puntos de corte de la parábola con el eje x (eje horizontal). Para representar correctamente la parábola, la gráfica debe estar centrada respecto a su eje de simetría (centro de la gráfica), el cual se calcula como $x = -b/2a$. Las figuras a continuación muestran un ejemplo de diseño de la hoja de cálculo solicitada, aunque dispones de total libertad para elegir el diseño que más te guste.



EXCEL ACTIVIDAD 17.

Diseña una hoja de cálculo capaz de resolver sistemas de dos ecuaciones lineales con coeficientes enteros:

$$\left.\begin{aligned} ax + by = c \\ dx + ey = f \end{aligned}\right\}$$

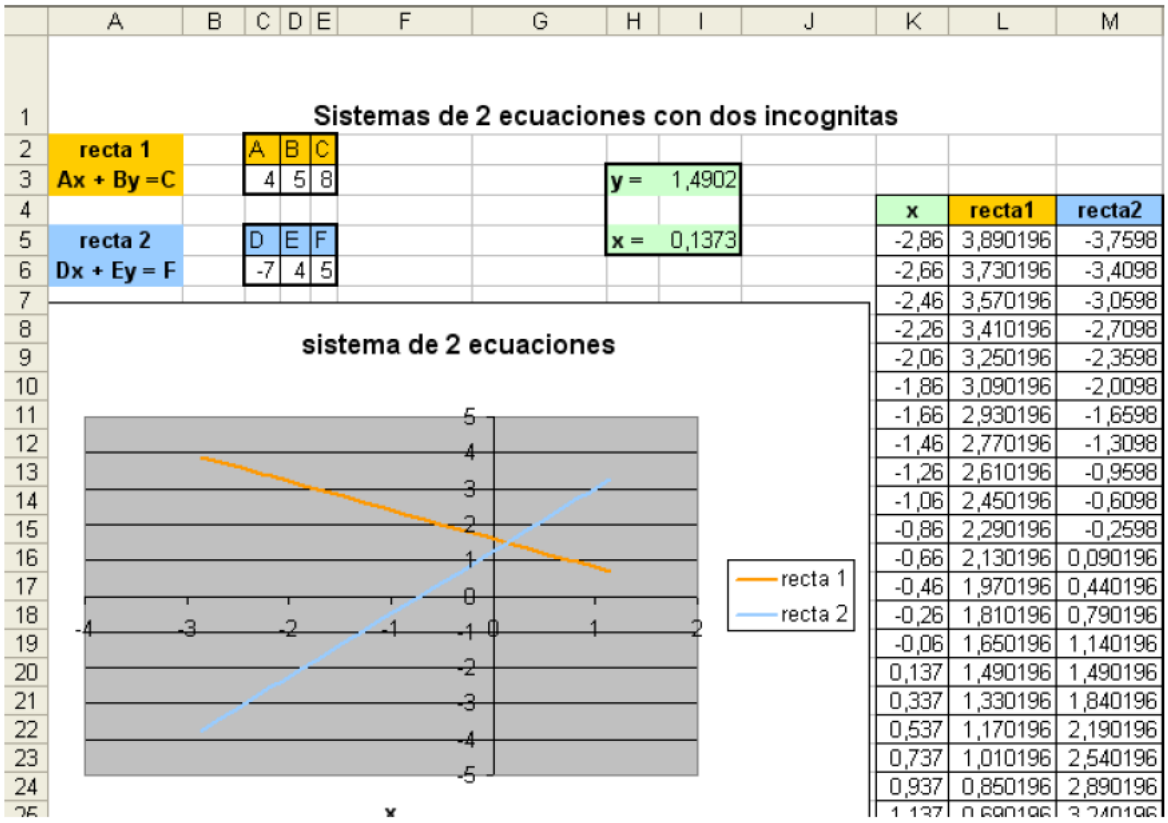
Las ecuaciones del sistema son rectas. Los datos de partida serán los coeficientes a, b, c, d, e, y f de ambas rectas.

Para resolver el sistema, debes aplicar las siguientes fórmulas:

$$x = \frac{\begin{vmatrix} c & b \\ f & e \end{vmatrix}}{\begin{vmatrix} a & b \\ d & e \end{vmatrix}} = \frac{ce - bf}{ae - bd} \qquad y = \frac{\begin{vmatrix} a & c \\ d & f \end{vmatrix}}{\begin{vmatrix} a & b \\ d & e \end{vmatrix}} = \frac{af - cd}{ae - bd}$$

Además de resolver el sistema, la hoja debe representar las dos rectas. Recordar que la intersección de ambas rectas representa la solución al sistema.

La figura a continuación muestra un ejemplo de diseño, pero tienes total libertad para elegir un diseño perosnalizado a tu gusto.



TIRO PARABÓLICO.

El objetivo de la última práctica es el estudio del lanzamiento parabólico de proyectiles. Dicho estudio se realiza descomponiendo el movimiento bidimensional en dos movimientos unidimensionales independientes, a saber, un movimiento rectilíneo uniforme en la dirección del eje x, y un movimiento rectilíneo uniformemente acelerado en la dirección del eje y, en el que la aceleración es la de la gravedad ($g = -9{,}8\ m/s^2$). Aquí vamos a despreciar la resistencia del aire.

Los datos de partida serán la velocidad inicial v_0, el ángulo α que forma la dirección del movimiento proyectil con el eje horizontal (que debe ser menor o igul que $90°$), y la aceleración gravitatoria $g = -9{,}8\ m/s^2$ (negativa porque apunta hacia abajo). Las ecuaciones de la trayectoria seguida por el proyectil son las siguientes:

Ecuaión de la trayectoria parabólica $y(x)$ que sigue el proyectil:

$$\left.\begin{array}{ll} v_{0x} = v_0\cos\alpha \quad \rightarrow & x = v_{0x}t \\ v_{0y} = v_0\sin\alpha \quad \rightarrow & y = v_{0y}t - \dfrac{gt^2}{2} \end{array}\right\} \quad \rightarrow \quad y(x) = x\tan\alpha - \frac{gx^2}{2v_0^2\cos^2\alpha}$$

Velocidad v:

$$\left.\begin{array}{l} v_x = v_{0x} \\ v_y = v_{0y} - gt \end{array}\right\} \quad \rightarrow \quad v = \sqrt{v_x^2 + v_y^2}$$

Alcance horizontal máximo x_{max} del proyectil:

$$x_{max} = \frac{v_0^2\sin(2\alpha)}{g}$$

Altura máxima y_{max} que alcanza el proyectil:

$$y_{max} = \frac{v_0^2\sin^2\alpha}{2g}$$

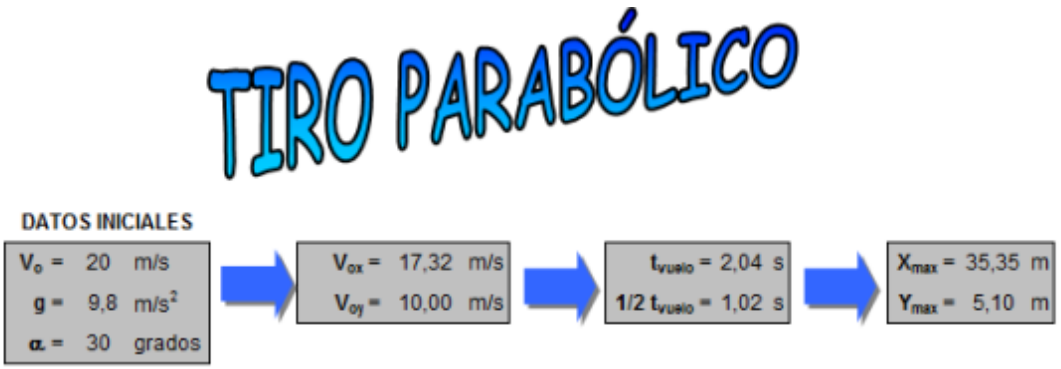
Tiempo de vuelo t_{vuelo} del proyectil:

$$t_{vuelo} = \frac{2v_0 \sin \alpha}{g}$$

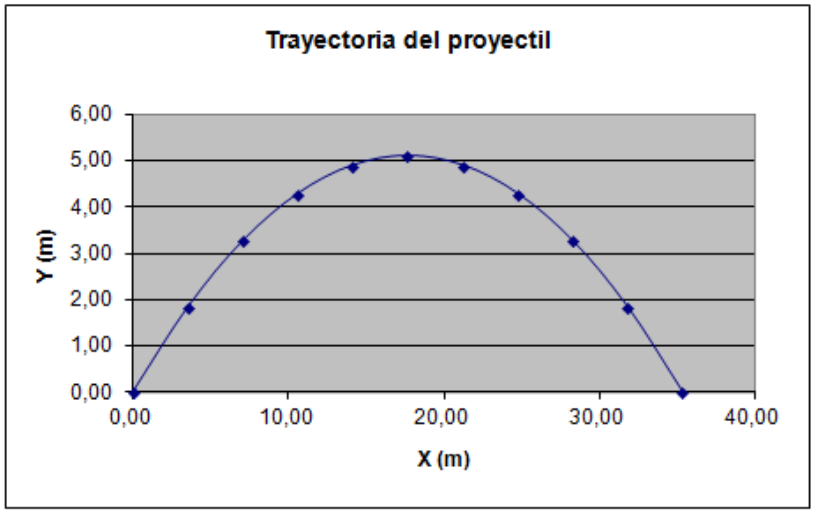
EXCEL ACTIVIDAD 18.

Construye una hoja Excel que calcule, a partir de los datos de partida (v_0, α, y g), los parámetros característicos del tiro parabólico: Velocidad inicial v_{0x} en eje x, velocidad inicial v_{0y} en eje y, tiempo de vuelo t_{vuelo}, alcance máximo x_{max}, y altura máxima y_{max}. Además, partiendo del tiempo $t = 0$ hasta $t = t_{vuelo}$, y en incrementos de tiempo de $t_{vuelo} / 10$, calcula la distancia x recorrida en horizontal, la distancia y recorrida en vertical, las velocidades horizontal v_x y vertical v_y, y la velocidad global v. Por último, dibuja una gráfica que muestre la trayectoria parabólica $y(x)$ del proyectil.

De nuevo, la figura muestra un ejemplo de diseño de la hoja de cálculo solicitada, pero cmo siempre, puedes elegir tu propio diseño personalizado.



X (m)	Y (m)	t (s)	V_x (m/s)	V_y (m/s)	V (m/s)
0,00	0,00	0,00	17,32	10,00	20,00
3,53	1,84	0,20	17,32	8,00	19,08
7,07	3,27	0,41	17,32	6,00	18,33
10,60	4,29	0,61	17,32	4,00	17,78
14,14	4,90	0,82	17,32	2,00	17,44
17,67	5,10	1,02	17,32	0,00	17,32
21,21	4,90	1,22	17,32	-2,00	17,44
24,74	4,29	1,43	17,32	-4,00	17,78
28,28	3,27	1,63	17,32	-6,00	18,33
31,81	1,84	1,84	17,32	-8,00	19,08
35,35	0,00	2,04	17,32	-10,00	20,00

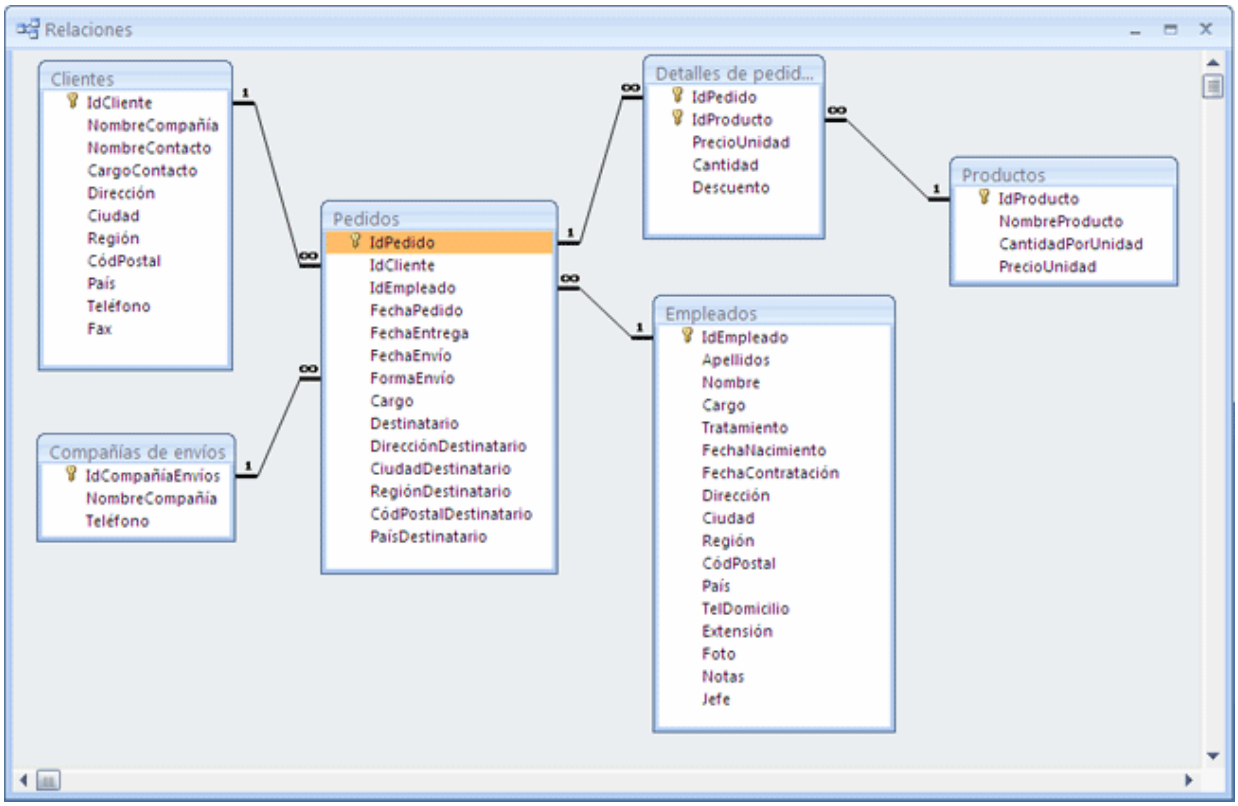


2. BASES DE DATOS: ACCESS.

2.1. ¿QUÉ ES UNA BASE DE DATOS?

En la actualidad hay multitud de situaciones en las que es necesario gestionar grandes volúmenes de datos de forma rápida y eficaz. Para dar respuesta a este problema se utilizan las bases de datos. Una **base de datos** (BBDD) es un "almacén" de información que permite guardar grandes cantidades de datos de forma organizada, para permitir su posterior acceso y consulta. Normalmente la información hace referencia a un tema concreto (Libros, Clientes, Películas, Música, Alumnos, etc.). Las **aplicaciones** más usuales de las bases de datos son la gestión de empresas y negocios (control de ventas, de clientes, de pedidos, etc.), instituciones y organismos públicos (policía, justicia, educación), y otros organismos y negocios (bibliotecas, videoclub, librerías), etc.

Las BBDD más habituales almacenan y organizan la información en tablas. Una BBDD está formada por **un conjunto de tablas**, donde los datos de las diferentes tablas están **interrelacionados** entre sí.



Los programas que permiten gestionar estas bases de datos se denominan **Sistemas Gestores de Bases de Datos** (SGBD). Estos SGBD permiten almacenar y posteriormente acceder a los datos de forma rápida y estructurada, mediante herramientas tales como consultas, formularios, informes, etc.

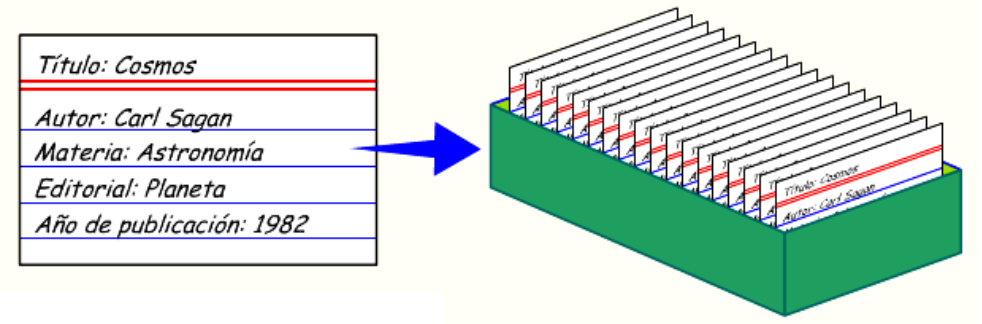
Algunos ejemplos de SGBD muy utilizados son Oracle, dBase, Sybase, MySQL, PostgreSQL (estos dos últimos gratuitos), etc. En este tema nos centraremos en uno de los sistemas gestores más extendidos y comunes, como es el **Microsoft Access**.

2.2. CONCEPTOS BÁSICOS DE BASES DE DATOS.

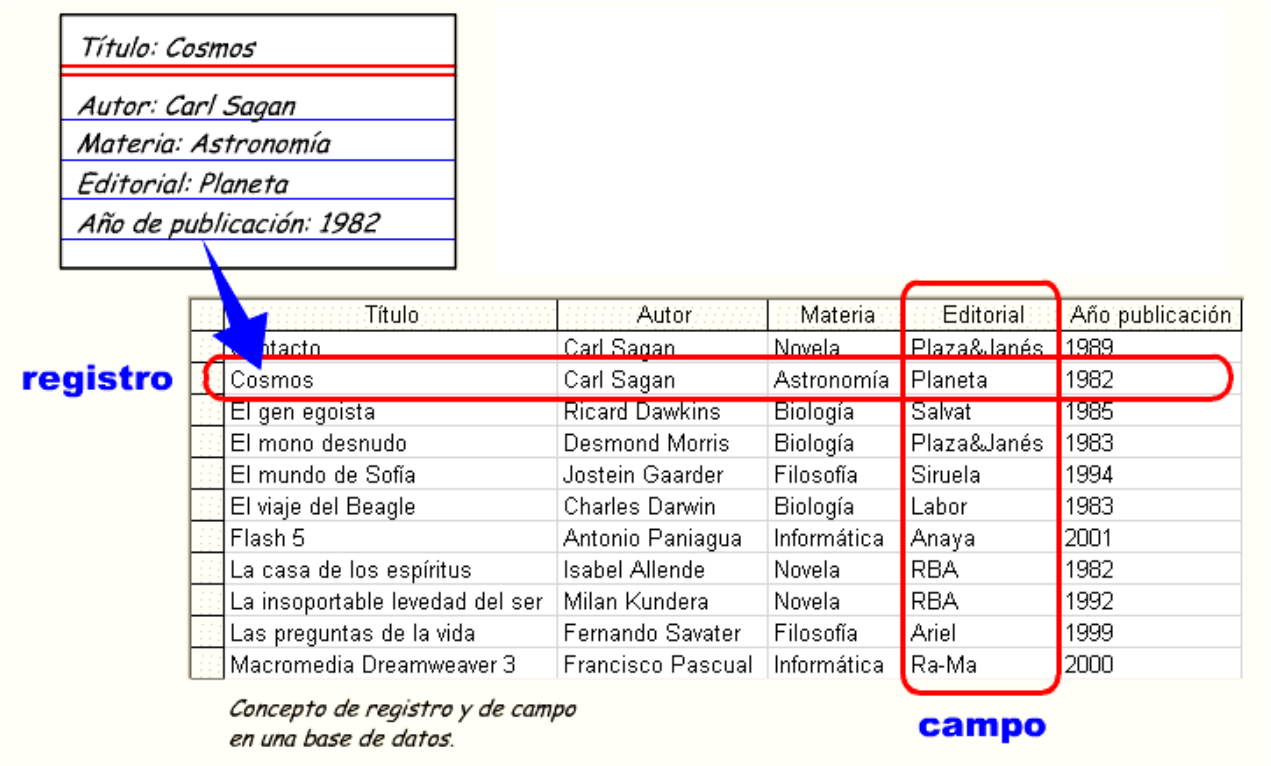
Los elementos básicos de una BBDD son los registros, los campos y las tablas.

Un **registro** es una "ficha" donde se guarda información sobre un objeto concreto del tema de estudio (por ejemplo: un cliente, un libro, un alumno, una empresa, etc.). Los registros contienen información del objeto

de estudio en forma de **campos**. Los campos son datos parciales del objeto de estudio (por ejemplo: en la ficha o registro un libro, los campos serían el título, el autor, la editorial, la fecha de publicación, etc.). Una **tabla** consiste en un conjunto organizado de datos formado por varios registros. En las Tablas los Registros son las filas y los Campos son cada uno de los datos que componen un registro (columnas). Por último, se dice que una **base de datos** es una colección de datos relacionados y organizados en forma de Tablas.

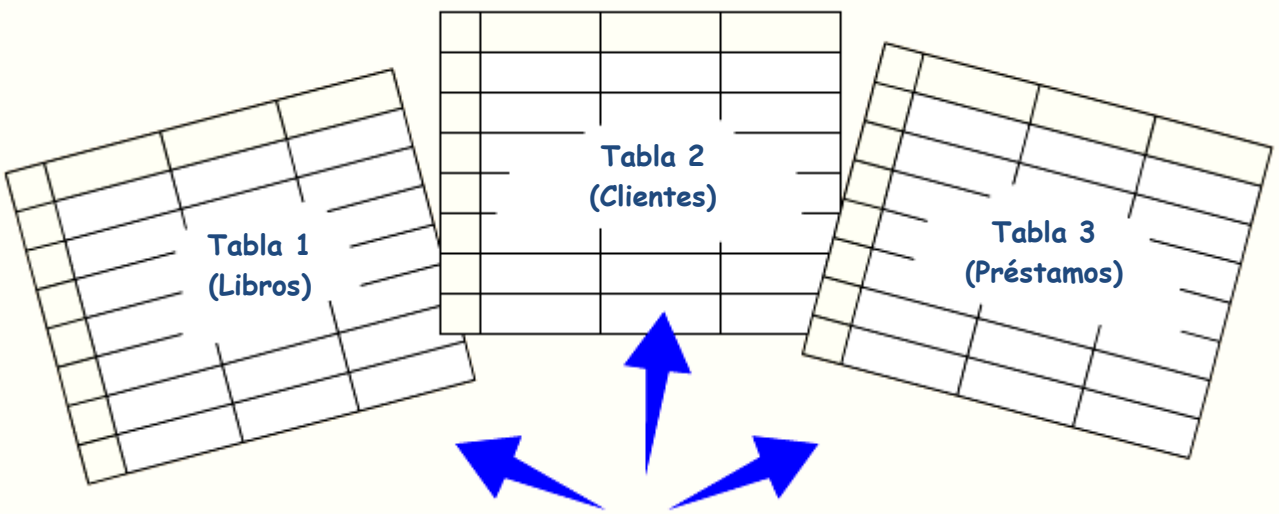


Registro (con 5 campos) Tabla (conjunto de registros)



Título	Autor	Materia	Editorial	Año publicación
ntacto	Carl Sagan	Novela	Plaza&Janés	1989
Cosmos	Carl Sagan	Astronomía	Planeta	1982
El gen egoista	Ricard Dawkins	Biología	Salvat	1985
El mono desnudo	Desmond Morris	Biología	Plaza&Janés	1983
El mundo de Sofía	Jostein Gaarder	Filosofía	Siruela	1994
El viaje del Beagle	Charles Darwin	Biología	Labor	1983
Flash 5	Antonio Paniagua	Informática	Anaya	2001
La casa de los espíritus	Isabel Allende	Novela	RBA	1982
La insoportable levedad del ser	Milan Kundera	Novela	RBA	1992
Las preguntas de la vida	Fernando Savater	Filosofía	Ariel	1999
Macromedia Dreamweaver 3	Francisco Pascual	Informática	Ra-Ma	2000

Concepto de registro y de campo en una base de datos.



Base de datos (grupo de Tablas relacionadas)

Para que una BBDD funcione correctamente es muy importante la **fase de creación y definición de las tablas** y el posterior **proceso de introducción de datos en las tablas**. Si estas fases se hacen correctamente, después se podrán realizar multitud de operaciones sobre los datos: Ordenarlos, filtrarlos, hacer consultas, hacer consultas relacionando datos de varias tablas, modificar y editar los datos de las tablas, mostrar los datos a través de la pantalla de forma atractiva, sacar listados o informes impresos, etc.

2.3. OBJETOS DE UNA BBDD EN ACCESS.

El gestor de BBDD que utilizaremos en la asignatura será Microsoft Access. Se trata de un gestor por muy popular y empleado en la actualidad, de gran potencia, flexibilidad, y de fácil uso. Los diferentes objetos que se pueden encontrar en una base de datos son **tablas**, **consultas**, **formularios**, e **informes**. Vamos a hacer un breve repaso de todo lo que ya sabemos antes de explicar estos objetos:

Una **Base de Datos** contiene todos los datos que se refieren a un determinado tema (como por ejemplo, la relación de jugadores de la liga de fútbol en primera división). En las Bases de Datos la información se guarda en forma de **Registros** formados por varios **Campos**. Tales registros son las entradas (filas) las **Tablas** que constituyen la BBDD. Una serie de tablas interrelacionadas entre sí conforman la **Base de Datos**.

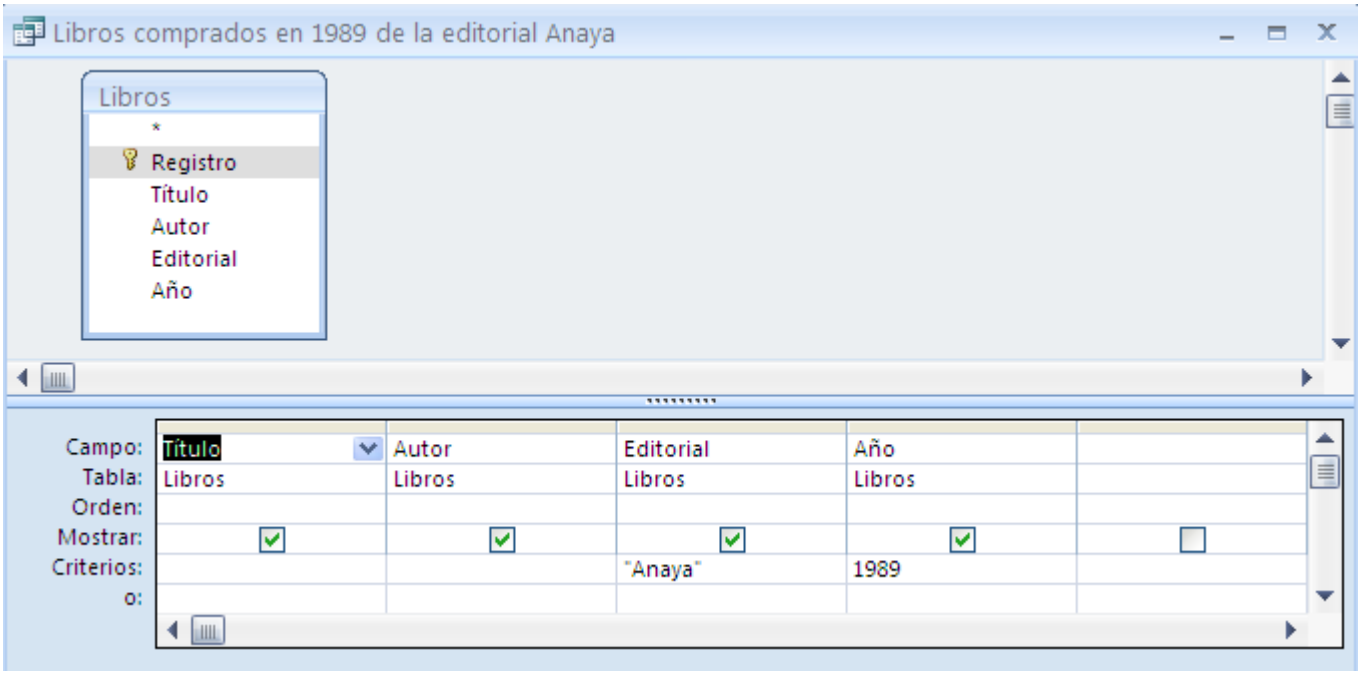
En las Bases de Datos Access existen diferentes objetos que permiten interaccionar con ellas, a saber, **Tablas**, **Consultas**, **Formularios** e **Informes**. Las **Tablas** se usan para almacenar los datos de forma estructurada. Las tablas permiten la introducción, edición, y visualización de los datos almacenados en la Base de Datos. Las Tablas son el elemento básico de una Base de Datos y origen de los otros tres objetos. Sin tablas no hay Base de Datos. Por su parte, las **Consultas** son objetos que permiten preguntar a la Base de Datos, obteniendo como respuesta datos que cumplan ciertos criterios. Un ejemplo de consulta sería "Extraer únicamente el nombre de aquellos clientes de Madrid que compraron más de 3 productos en Diciembre". Los **Formularios** permiten visualizar los datos de modo diferente al de una tabla, resultando más atractivos, vistosos, y eficaces (aspecto gráfico de una ficha). Al igual que las tablas, los formularios permiten introducir, mostrar, y editar la información de una base de datos de forma mucho más visual, con la ayuda de listas desplegables, botones, cuadros de opción, etc. Por último, los **Informes** se usan para realizar la impresión en forma de documento de los datos almacenados en una base de datos. Estos informes se pueden diseñar al gusto del usuario (datos mostrados, cabeceras, columnas, colores, líneas, bordes, etc.).

Libros

Registro	Título	Autor	Editorial	Año	Agregar nuevo campo
1	La vida del Lazarillo de Tormes	Anónimo	Castalia	1989	
2	Le Petit Prince	Saint Exupery, Antoine De	Gallimard	1987	
3	El Buscón	Quevedo, Francisco De	Castalia	1989	
4	Cartas marruecas	Cadalso, José	Castalia	1989	
5	El Buscón	Quevedo, Francisco De	Castalia	1989	
6	Pepita Jiménez	Valera, Juan	Castalia	1988	
7	Coplas a la muerte de su padre	Manrique, Jorge	Castalia	1989	
8	Romancero viejo. Antología	V.V.A.A.	Castalia	1989	
9	Rimas	Bécquer, Gustavo Adolfo	Castalia	1989	
10	Poesías escogidas	Machado, Antonio	Castalia	1989	
11	Relatos breves	Alas, Leopoldo (1852-1901) (Clarín)	Castalia	1986	
12	Cantar del Mío Cid	Anónimo	Espasa-Calpe	1988	
13	Cantar del Mío Cid	Anónimo	Espasa-Calpe	1988	
14	Cantar del Mío Cid	Anónimo	Espasa-Calpe	1988	
15	Cantar del Mío Cid	Anónimo	Espasa-Calpe	1988	
16	Cantar del Mío Cid	Anónimo	Espasa-Calpe	1988	
17	Cantar del Mío Cid	Anónimo	Espasa-Calpe	1988	
18	Cantar del Mío Cid	Anónimo	Espasa-Calpe	1988	

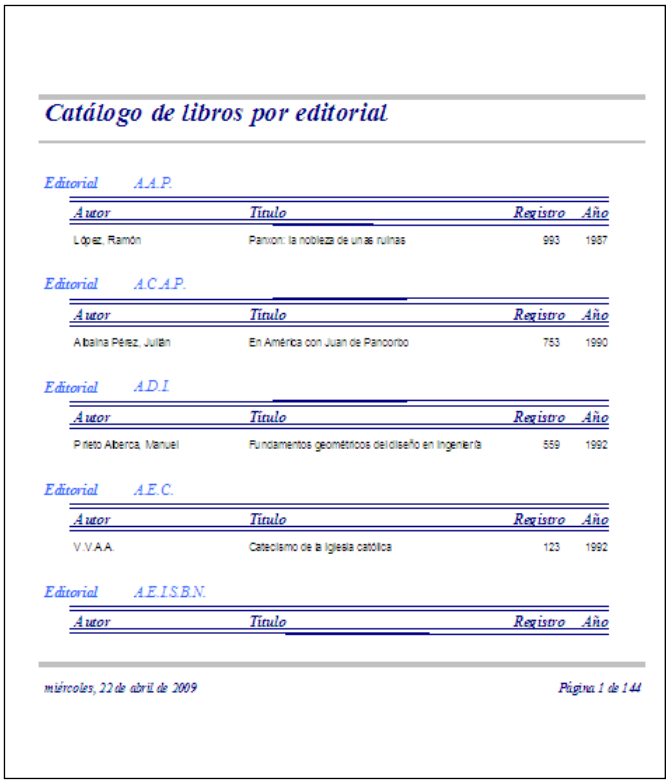
Registro: 1 de 1840 Sin filtro Buscar

Tabla.



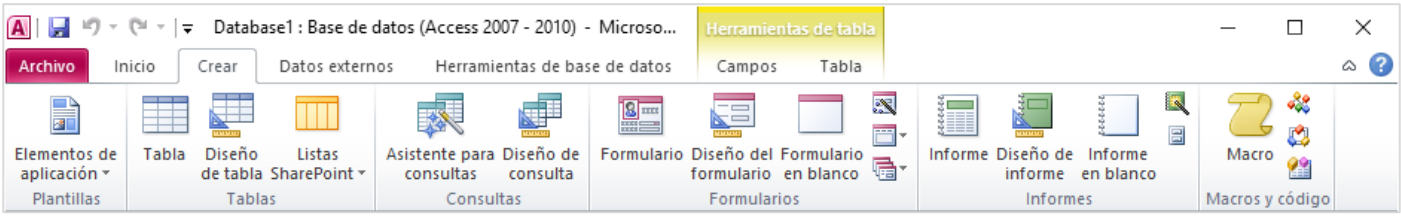
Consulta.

Formulario.



Informe.

La gestión de los diferentes objetos que componen una BBDD se realiza de forma centralizada desde el menú "Crear".



A modo de ejemplo, abre la BBDD llamada "Biblioteca.accdb" y observa los objetos que tiene creados. Responde a las siguientes preguntas en un documento Word (Ejer1.docx), y haz capturas de pantalla para justificar tus respuestas.

a) Abre la tabla "Libros" pulsando sobre ella dos veces con el ratón. ¿Cuántos campos y cuántos registros tiene? En el menú "Inicio" > "Ver", observa las vistas disponibles para las tablas (Vista de Hoja de Datos, Vista de Diseño).

b) Abre la consulta que hay definida. Observa el nombre de la consulta y sus resultados. Observa las 3 vistas disponibles (Vista de Diseño, Vista Hoja de Datos y Vista SQL)

c) Abre el formulario libros y observa cómo se muestran los datos de la tabla.

d) Abre el informe y observa el diseño del mismo. Comprueba que imprime los datos almacenados en la base de datos.

2.4. OPERACIONES BÁSICAS CON TABLAS EN ACCESS.

Para introducir, editar, y visualizar la información de una BBDD desde una tabla hay que trabajar con la tabla en la vista de Hoja de Datos.

INTRODUCIR NUEVOS REGISTROS EN UNA TABLA.

Hay varias formas de introducir nuevos registros:

1) Pulsando el botón de "último registro" en la barra de navegación, y añadiendo en la última posición los datos del nuevo registro.

2) Sobre la tabla, en la columna a la izquierda del primer campo, pulsando con el botón derecho del ratón y seleccionando la opción "Nuevo registro".

3) A través del menú "Inicio" > "Registros" > "Nuevo".

ELIMINAR REGISTROS.

Hay 2 formas de eliminar registros de una tabla:

1) Seleccionando el registro (fila), pulsando botón derecho del ratón, y seleccionando la opción "Eliminar registro".

2) Seleccionando el registro, y en el menú "Inicio" > "Registros" > "Eliminar".

ORDENAR LOS REGISTROS.

a) Seleccionamos el campo o campos en base a los cuales deseamos ordenar los registros.
b) Pulsamos con el botón derecho del ratón, y elegir uno de los botones de ordenar, según el tipo de ordenación deseada (creciente, decreciente, o personalizada).

EDITAR LOS DATOS DE UNA TABLA.

Basta con situarse en el campo a modificar y reescribir el dato.

ACTIVIDAD 1.

Abre la base de datos "Biblioteca.accdb" y accede a la tabla "Libros".

a) Introduce estos 5 registros nuevos por el método que prefieras:

Registro	Título	Autor	Editorial	Año
1852	El rey y la reina	Sender, Ramón J.	Destino	1992
1853	Ulises	Joyce, James	Lumen	1995
1854	La sonrisa etrusca	Sampedro, José Luis	R.B.A.	1988
1855	El General en su laberinto	García Márquez, Gabriel	Oveja Negra	1993
1856	Jardín de flores raras	Caro Baroja, Julio	Círculo Lectores.	1995

NOTA: El campo registro es de tipo Autonumérico, lo cual significa que se rellena automáticamente con cada nuevo registro introducido.

b) Elimina el registro 22 ("Artículos" de Mariano José de Larra).

c) Ordena la tabla por Año de publicación, del más actual a más antiguo. Ordena ahora la tabla por Título y por Autor, alfabéticamente. Observa cómo, en caso de igualdad en el título, se ordenan por el autor.

d) Modifica el registro 9 de la Tabla ("Rimas"). Cambia el nombre por "Rimas y Leyendas", cambia la editorial a Planeta y el año a 1987.

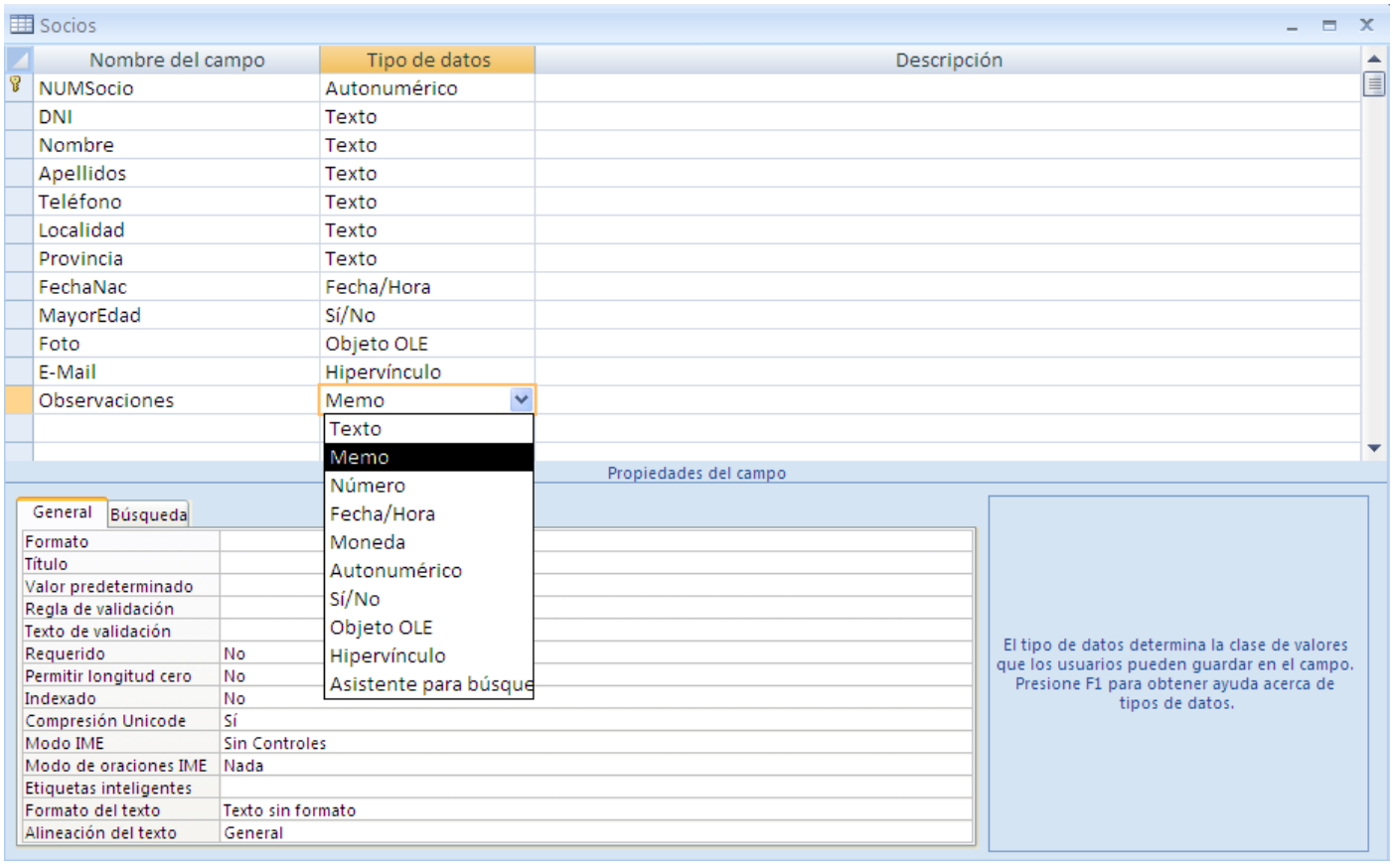
2.5. OPERACIONES AVANZADAS CON TABLAS.

DEFINICIÓN DE LA ESTRUCTURA DE UNA NUEVA TABLA.

Hasta ahora hemos trabajado con BBDDs que tenían sus tablas previamente creadas. Aquí veremos cómo crear y definir las tablas de una BBDD partiendo de cero.

Para crear una tabla, acudimos al menú "Crear" > "Tablas" > "Diseño de Tabla". Entonces se abrirá la **vista de diseño** de la tabla para la definición de la **estructura** de la nueva tabla. En la estructura de una tabla se especifica:

- El nombre de cada uno de los campos que incluirá la tabla.
- El tipo de datos almacenados en cada campo (textos, números, fechas, etc.).
- Una descripción del campo.
- Las propiedades del campo (tamaño, campo opcional u obligatorio, formato esperado en los datos, etc.).



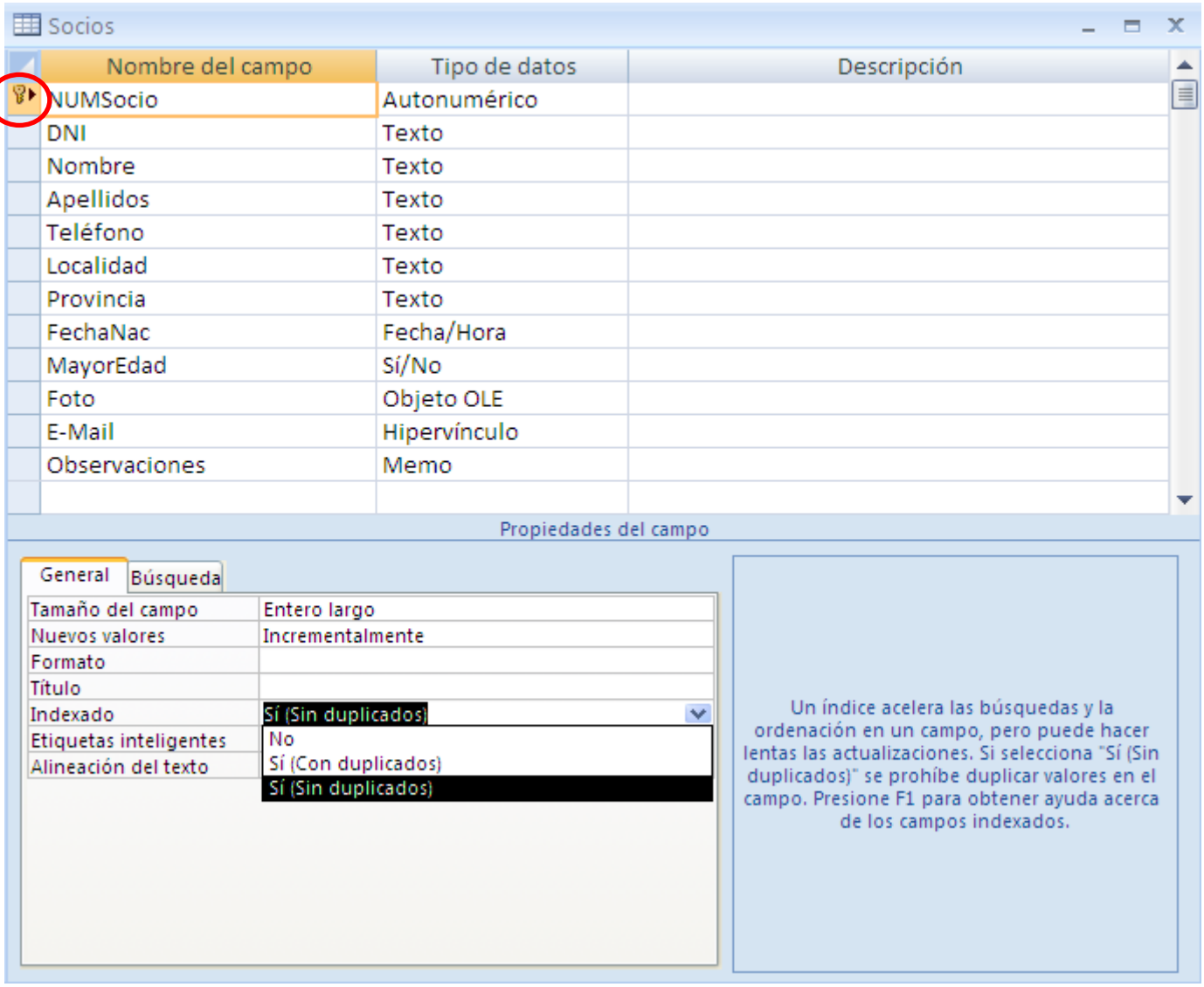
Los tipos de datos que pueden tener los campos de una tabla son los siguientes:

TIPOS DE DATOS	DESCRIPCIÓN
Texto	Admite textos o combinación de texto y caracteres numéricos, pero no es capaz de operar con los caracteres numéricos.

Numérico	Datos numéricos para los campos que requieran cálculos matemáticos
Fecha/Hora	Para la inclusión de fechas y horas
Si/No	Campo que sólo puede contener dos valores posibles: Si/No, Verdadero/Falso, Activado/Desactivado, etc.
Autonumérico	Secuencia de números que se incrementan en una unidad cada vez que se introduce un nuevo registro.
Moneda	Para valores numéricos de tipo moneda. Se diferencian de los numéricos en que evitan el redondeo.
Memo	Admite textos y números de gran longitud.
Objeto OLE	Para la inclusión de objetos: Imágenes, sonidos, documentos (PDF, Word, Excel), etc.
Hipervínculo	Campo para almacenar hipervínculos.

En toda tabla hay un campo especial denominado **clave principal** (o **indexado**), cuya misión es identificar de forma unívoca cada registro de una tabla. El campo que actúa de clave principal se designa con el icono de una llave en la vista de diseño de la tabla. El campo de clave principal suele ser de tipo **autonumérico**, para asegurar que un valor no se repita en dos registros distintos. La clave principal facilita trabajar con bases de datos, ya que:

- Acelera las búsquedas de datos.
- Facilita establecer relaciones entre tablas.



Ahora, las propiedades que caracterizan a cada uno de los campos de un atabla dependen del tipo de datos almacenado en dicho campo. La siguiente tabla resume las propiedades que caracterizan a los campos según su tipo de datos:

PROPIEDADES	FUNCIÓN	TIPO DE DATOS
Tamaño del campo	Para determinar el número de caracteres	Texto, Numérico
Formato	Personaliza la forma del campo	Texto, Numérico, Moneda, Fecha/Hora, Autonumérico, Si/No.
Máscara de entrada	Para dar idéntico formato de entrada en todos los contenidos del campo.	Texto, Numérico, Moneda, Fecha/Hora, Autonumérico, Si/No.
Título	Nombre para formularios e informes	En todos los campos.
Valor predeterminado	Valor que tomará el campo por defecto.	Texto, Numérico, Moneda, Fecha/Hora, Si/No.
Regla de validación	Determina las condiciones que debe cumplir el dato que se introduce.	Texto, Numérico, Moneda, Fecha/Hora, Si/No, Objeto OLE.
Texto de validación	Mensaje de error que aparece si el dato no cumple la regla de validación.	Texto, Numérico, Moneda, Fecha/Hora, Si/No, Objeto OLE.
Requerido	Obliga a introducir datos en este campo. El campo no puede quedar vacío.	Texto, Numérico, Moneda, Fecha/Hora, Si/No, Objeto OLE.
Indexado	Determina un índice para el campo	Texto, Numérico, Moneda, Fecha/Hora, Autonumérico.
Lugares decimales	Indica el número de decimales por defecto.	Numérico y moneda

Para terminar, los datos que podemos introducir en cada uno de los campos de un atabla pueden venir controlados por una máscara de entrada, que restringe el formato del dato que podemos insertar en ese campo. Para definir una máscara de entrada se utilizan los símbolos que indicamos a continuación:

0 → Número (0-9). Entrada obligatoria
9 → Número (0-9). Entrada NO obligatoria
L → Letra (A-Z). Entrada obligatoria
? → Letra (A-Z). Entrada NO obligatoria

A → Número o Letra. Entrada obligatoria
a → Número o Letra. Entrada no obligatoria
C → Cualquier carácter (número, letra, u otro). No obligatorio
\ → Hace que el carácter que viene después sea literal.

Algunos ejemplos de uso de máscaras de entrada son los siguientes:

MÁSCARA	ADMITIRÍA	NO ADMITIRÍA
(956) 00-00-00	(956) 25-60-98 ; (956) 66-10-20	(956) AA-¿?-() ; (956)256098
M-0000-L?	M-4589-X ; M-4135-BB; M-7895-KL	M-45WQ-55 ; M-4Q6F-AMF
REF-90AA-\C\A	REF-7855-CA ; REF-56BK-CA REF-9BJ-CA ; REF-48G2-CA	REF-7855-JK ; REF-MM-CA REF-33A$-CA ; REF-M1J1-CA

ACTIVIDAD 2.

Crea una BBDD llamada "Club_Deportivo.accdb", para la que necesitaremos dar de alta las siguientes tablas:

TABLA 1 → "SOCIOS"

- Nº SOCIO → Se numerará automáticamente, será la clave de la tabla.
- DNI → La estructura habitual incluyendo la letra.
- NOMBRE → Es obligatorio, con 25 caracteres es suficiente.
- APELLIDOS → Es obligatorio, con 50 caracteres es suficiente.
- TELEFONO → Todos los socios son y serán de Albacete (prefijo 967). El prefijo en paréntesis.
- LOCALIDAD → Es obligatorio, con 60 caracteres es suficiente.
- PROVINCIA → Todos los socios son y serán de la provincia de Albacete.
- AÑO NACIMIENTO → Debe aparecer el año con 4 dígitos.
- MAYOR EDAD → Aquí se debe indicar si el socio es mayor de edad o no.
- FOTO → Incluirá una foto digitalizada del socio.
- E-MAIL → Indicaremos la dirección de correo electrónico del socio.
- OBSERVACIONES → Aquí se podrá escribir todo lo que queramos sobre el socio (opcional).

TABLA 2 → "PISTAS"

- COD_PISTA → Se numerarán automáticamente y será la clave de la tabla.
- DESCRIPCIÓN → Aquí escribiremos el tipo de pista (Tenis, Pádel, Futbol, Baloncesto).
- OBSERVACIONES → Para poder escribir cualquier comentario (desperfecto, etc.).

TABLA 3 → "ACTIVIDADES"

- COD_SOCIO → Numérico, aquí se indica qué socio de la tabla socios está alquilando.
- COD_PISTA → Numérico, aquí se indica qué pista de la tabla pistas se alquilará.
- FECHA → Aquí se indica en qué fecha se va a hacer la reserva.
- HORA_C → Hora de comienzo de la reserva de la pista.
- HORA_F → Hora de Finalización de la reserva de la pista.
- PRECIO → Aquí se indica el precio de la reserva (€).

TABLA AUX → "LOCALIDADES"

- NUM → Autonumérico.
- Localidad → Texto, 50 caracteres.

ACTIVIDAD 3.

Vamos a generar una BBDD llamada ***Video_club.accdb*** para poder llevar a cabo las actividades propias de un video club: registro de clientes y películas, control de devolución de películas, impresión de listados, etc.

La base de datos deberá poseer tres tablas:

1) Un registro de **clientes**, que incluya los datos relevantes de cada uno de los clientes del video club. Campos necesarios: Código de Cliente, Nombre, Apellidos, Dirección, Población, Teléfono y D.N.I.
2) Un catálogo de las **películas** disponibles. Campos necesarios: Código de Película, Título, y Sección (Acción, Drama, Comedia, Terror, Ciencia Ficción, Western, etc.)

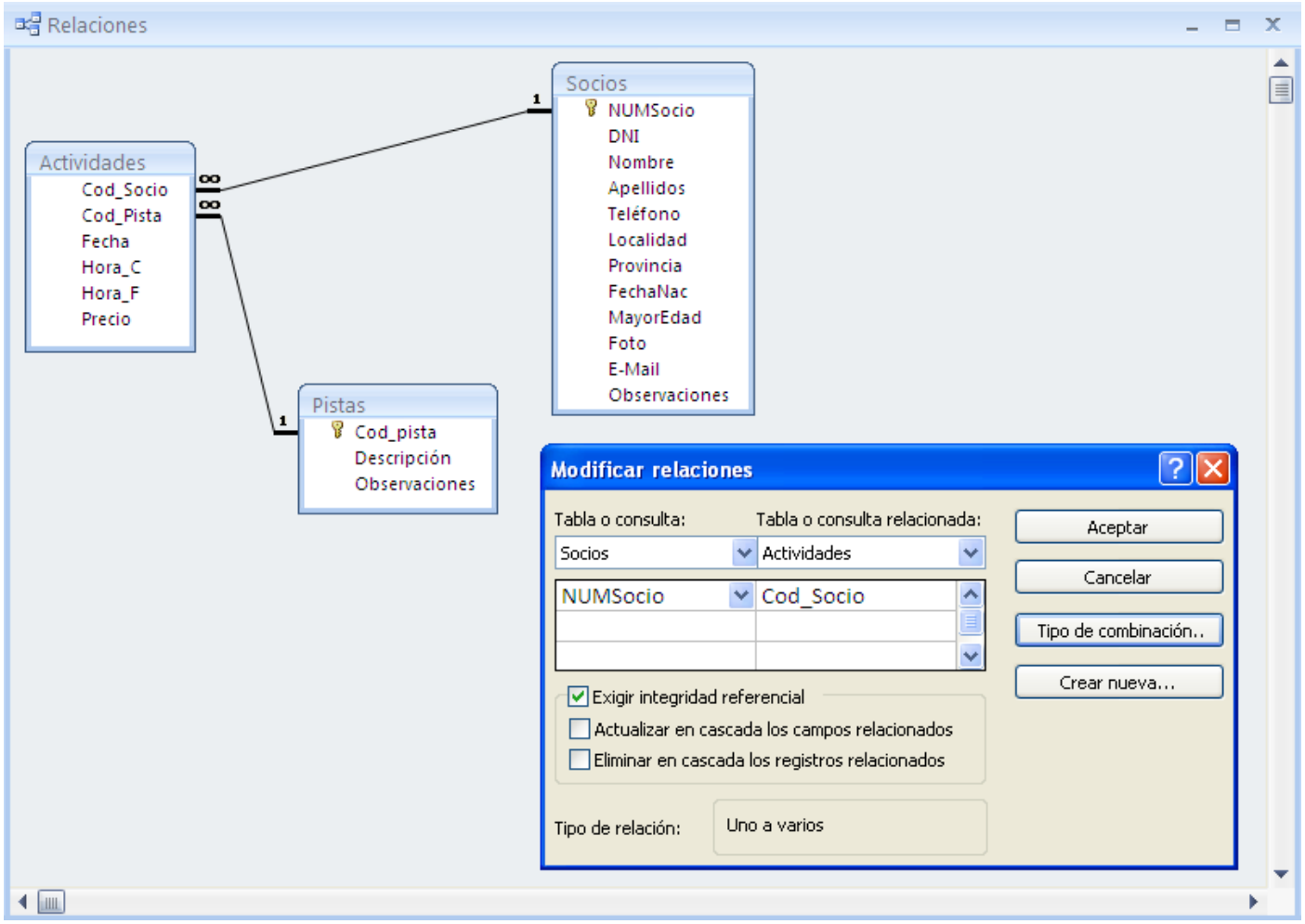
3) Un listado de **préstamos** que contenga aquellas películas que han sido prestadas, a qué cliente y si la película ha sido devuelta o no. Campos necesarios: Código de Cliente, Código de Película, Fecha de Préstamo, Devolución.

DEFINICIÓN DE LAS RELACIONES ENTRE LAS TABLAS.

Como ya hemos indicado con anterioridad, las BBDD están formadas por varias tablas cuyos datos están interrelacionados entre sí. Al construir una nueva BBDD, debemos indicar explícitamente las relaciones entre los datos de las diferentes tablas de esa BBDD. No todas las tablas pueden interrelacionarse entre sí. Para que dos tablas puedan estar relacionadas deben tener campos en común, y los campos comunes deben tener exactamente el mismo tipo de datos.

Para relacionar las tablas entre sí, debemos seguir estos pasos:

1) Seleccionamos el menú "Herramientas de bases de datos" > "Relaciones".
2) Pulsamos el botón "Mostrar Tabla", y agregamos las distintas tablas que queramos relacionar.
3) Las tablas se relacionan pinchando y arrastrando sobre el campo que tienen en común.
4) Activamos la opción "Exigir integridad referencial".



ACTIVIDAD 4.

1) Atendiendo a la figura previa, crea las relaciones entre las tablas de la BBDD "Club_Deportivo.accdb". Establece siempre la exigencia de integridad referencial.

2) Establece las relaciones necesarias entre las tablas de la BBDD "Video_club.accdb".

INTRODUCIR REGISTROS EN UNA NUEVA TABLA.

Una vez definidas las estrucuturas de las tablas de una nueva BBDD, y tras establecer las relaciones entre las mismas, podemos comenzar a introducir los datos en las diferentes tablas tal y como explicamos en el apartado 2.4. Recordemos que la introducción de datos en las tablas se realiza desde la "Vista Hoja de Datos". A la hora de introducir datos en una tabla, es vital mantener la coherencia tratando de seguir siempre los mismos criterios:

- Si los datos se introducen en mayúsculas, debemos hacerlo así en todas las entradas de datos.
- Si los datos se introducen en minúsculas y con tildes, debemos mantener este criterio en todas las entradas de datos.
- Si se usa una abreviatura (Sr./Sra., Avda., C/, etc.), debemos utilizar siempre la misma abreviatura en todos los registros

BUSCAR DATOS EN UNA TABLA.

La búsqueda de datos en una tabla de Access es muy similar a la de otras aplicaciones Office:

1) Abrimos la tabla en la "Vista Hoja de Datos".
2) Seleccionar el campo donde deseamos realizar la búsqueda.
3) Seleccionamos la herramienta de búsqueda adecuada en el menú "Inicio" > "Buscar", o "Inicio" > "Buscar y Reemplazar".

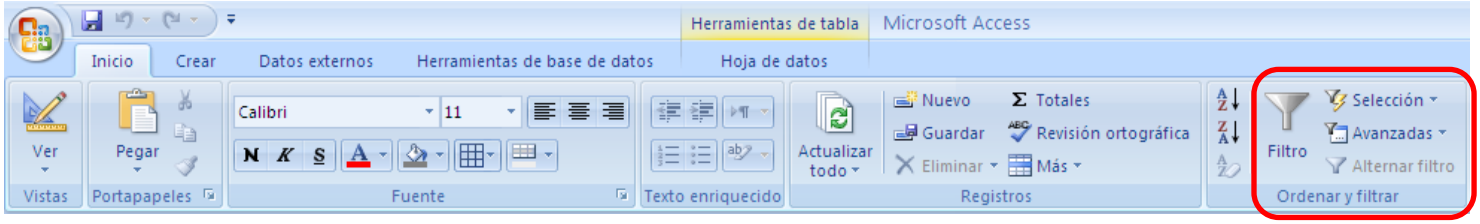
ACTIVIDAD 5

1) En la BBDD "Biblioteca.accdb":
   a) Busca todas las copias del libro "El Buscón". En un archivo de Word (que guardarás con el nombre de Ejer5_1.docx), apunta el número de registro de cada una de las copias.
   b) Busca los diferentes libros en cuyo título aparezca la palabra "hombre". En el mismo archivo, apunta el nombre de todos esos libros y su número de registro.
   c) Localiza los libros que traten sobre "astronomía". Apuntate su nombre completo y su autor.
   d) Localiza los libros cuyo título comience por "Enciclopedia". Apúntate su nombre completo, la editorial y el año de edición.

2) En la BBDD "Animales.accdb":
   a) Localiza la información sobre la Víbora. En un archivo de Word (que guardarás con el nombre de Ejer5_2.docx), apunta toda la información encontrada.
   b) Localiza aquellas especies cuyo hábitat sean los bosques, y anota su nombre, su clase, y su hábitat.

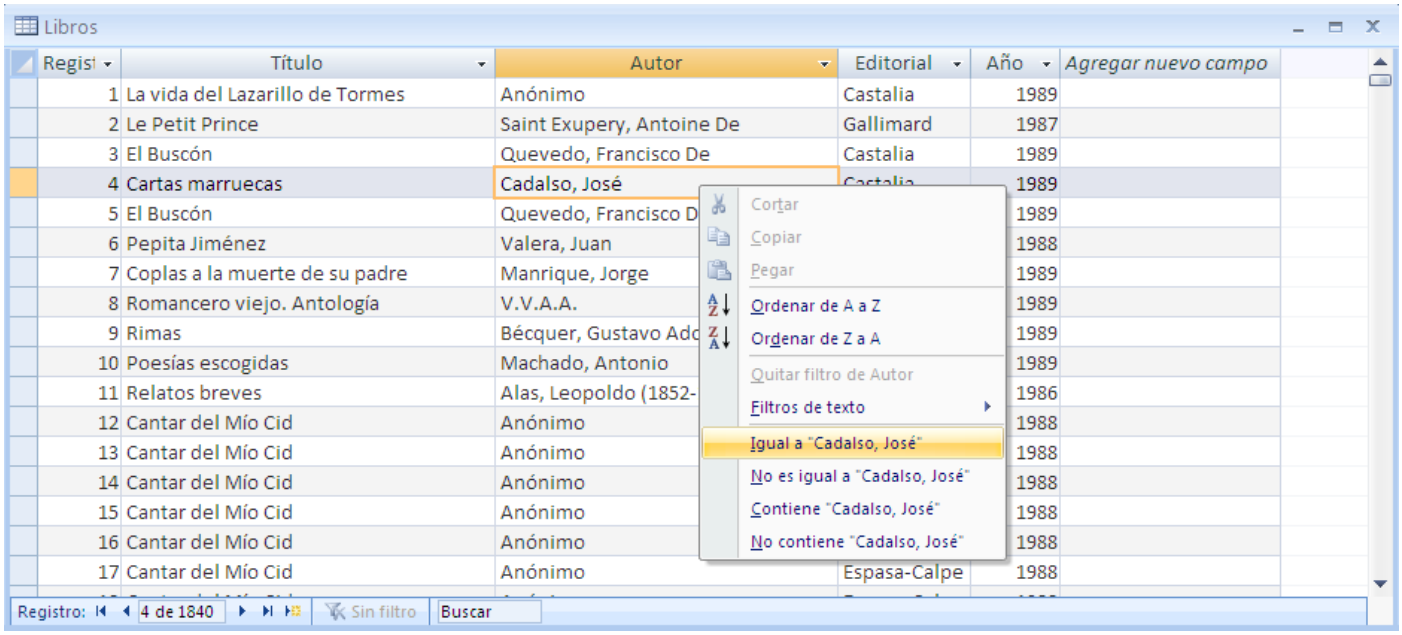
FILTRAR DATOS EN UNA TABLA.

La operación de filtrado de datos es muy similar a la herramienta correspondiente de Excel, y permite visualizar únicamente aquellos registros que presentan un dato común concreto en un campo o campos determinados. Existen varios tipos de filtrado:



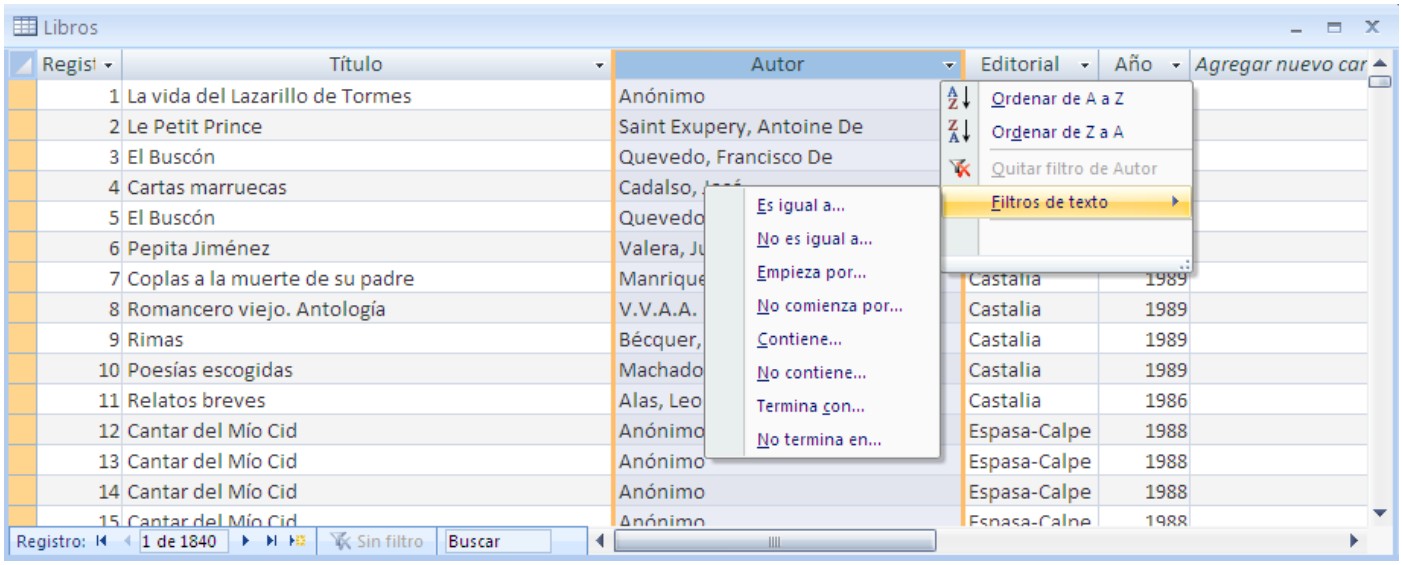
Filtros de selección.

Se usan para filtrar por el dato que está seleccionado en ese preciso momento. Para ello basta con seleccionar la casilla donde está el dato, y seleccionar uno de los filtros que propone el menú contextual. También está accesible en el menú "Inicio" > bloque "Ordenar y filtrar" > "Selección".

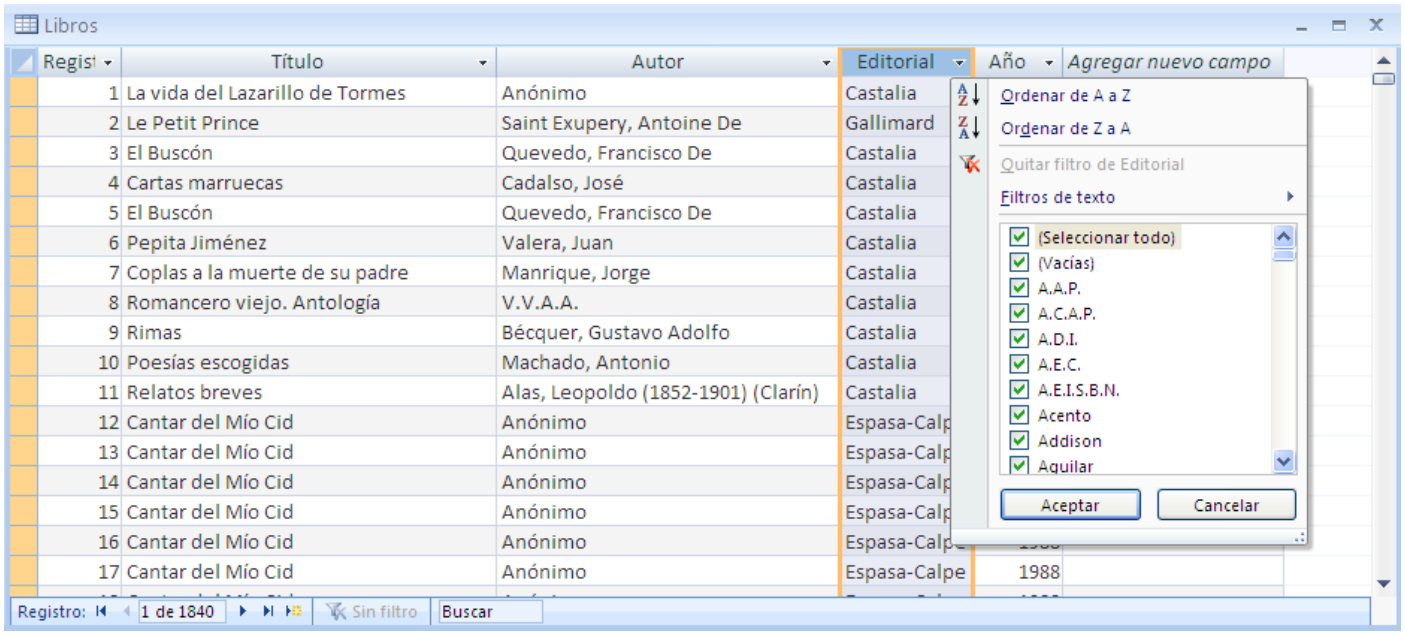


Filtros personalizados.

Son filtros que nos permiten realizar filtrados personalizados, atendiendo a diversos criterios. Para activarlos basta con seleccionar el campo por el que deseamos filtrar, pulsar en la flechita junto al nombre del campo, y seleccionar la condición de filtrado que más nos interese:



Según el tipo de campo y los datos almacenados en él, podrán aparecer filtros contextuales para determinar mediante botones de selección los datos a mostrar:



Filtros por formulario.

Esta opción nos permite operaciones de filtrado más complejas, como por ejemplo, filtros por varios campos simultáneos, mezclas de filtros múltiples, etc. Para activar el filtrado por formulario cudimos al menú "Inicio" > bloque "Ordenar y filtrar" > "Avanzadas" > "Filtro por formulario". Para aplicar el filtro por formulario que hayamos diseñado, seleccionamos la opción "Alternar filtro". Un ejemplo de filtro por formulario sería el filtro que nos permitiese buscar libros de la Editorial Planeta en 1982 O (Or) libros de la editorial Castalia O (Or) libros de un año distinto a 1988.

Editorial	Año
"Planeta"	1982

Editorial	Año
"Castalia"	

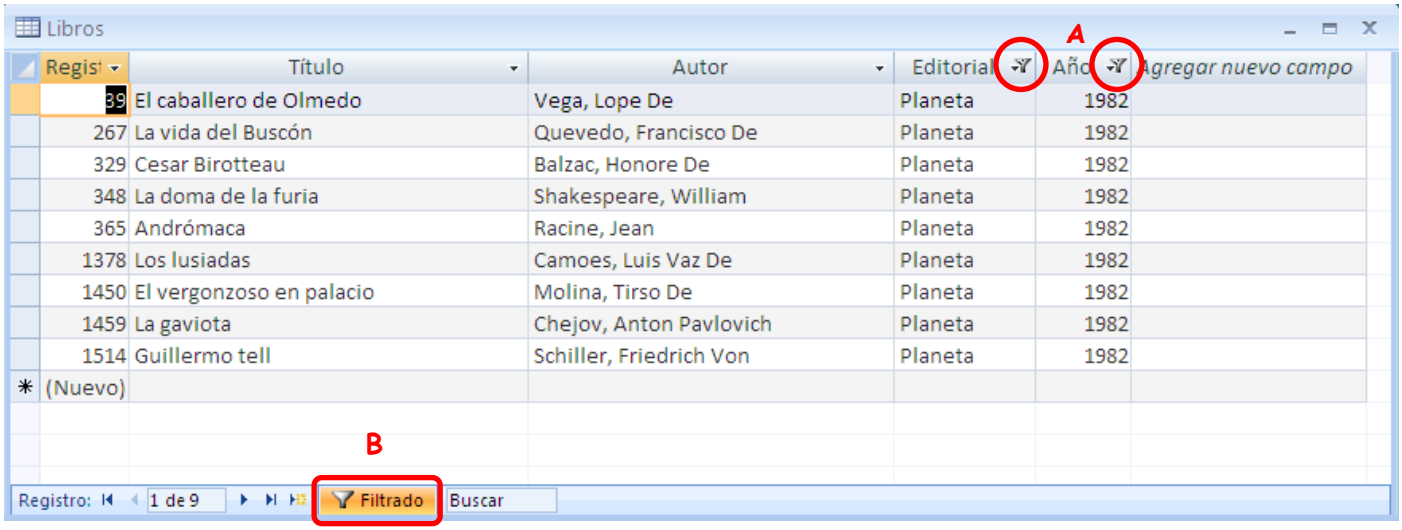
Editorial	Año
	<>1988

Eliminar filtros.

Los campos que incluyen filtros aparecen con el icono del embudo junto a su título. Para deshacer el filtrado efectuado sobre una tabla, hay varias posibilidades:

a) Pulsar en el icono del embudo del campo filtrado, y seleccionar la opción "Quitar filtro". Con ello eliminamos el filtro de ese campo, pero no los del resto de campos que también puedan tener algún filtro. Para deshacer todos los filtros tendríamos que repetir esta operación en todos los campos con filtro.

b) Deseleccionar la opción "Filtrado" al final de la tabla. Con esta opción se deshacen todos los filtros en todos los campos.



ACTIVIDAD 6

A continuación se proponen una serie de actividades para trabajar con filtros. Guarda los resultados del filtrado en un documento aparte (por ejemplo, con endos libros Excel llamados "Ejer6(biblioteca).xlsx" y "Ejer6(animales).xlsx," que incluyan una pestaña por cada ejercicio). Antes de pasar al siguiente filtro, no olvides quitar el filtro del ejercicio anterior (a menos que se te indique lo contrario)

En la BBDD "Biblioteca.accdb":

a) Crea un filtro para seleccionar los libros que se han comprado en 1985. ¿Cuántos son?
b) ¿Cuáles son los libros escritos sobre Goya?
c) Ordena los libros por fecha de devolución. ¿Dónde quedan los libros no devueltos aun?
d) Visualiza sólo los libros de la editorial Castalia y que fueron comprados en 1989.
e) Sobre lo anterior, filtra los que además de esas 2 condiciones, son de Francisco de Quevedo.
f) Quédate sólo con los libros cuyo autor es conocido.
g) Ahora filtra aquellos libros de la editorial Anaya publicados en 1985 más los que fueran publicados en 1994 sin importar su editorial.
h) Intenta ordenar los libros de menor a mayor año y, en caso de igualdad, alfabéticamente por el título de la publicación.

En la BBDD "Animales.accdb":

a) Filtra la tabla para saber cuántas especies de mamíferos hay.
b) Visualiza ahora sólo las especies que habiten en los bosques.
c) Ordena los registros de mayor a menor peso de cada especie.
d) Visualiza las especies cuyo nombre empiece por "C" (Pista: "C*").
e) ¿Cuáles de las especies son "reales"?
f) ¿Qué especies miden más de 1 metro?
g) ¿Qué especies viven o pueden vivir en las montañas?
h) Muestra los peces que viven en los ríos junto a los mamíferos que viven en los mares.

2.6. CONSULTAS.

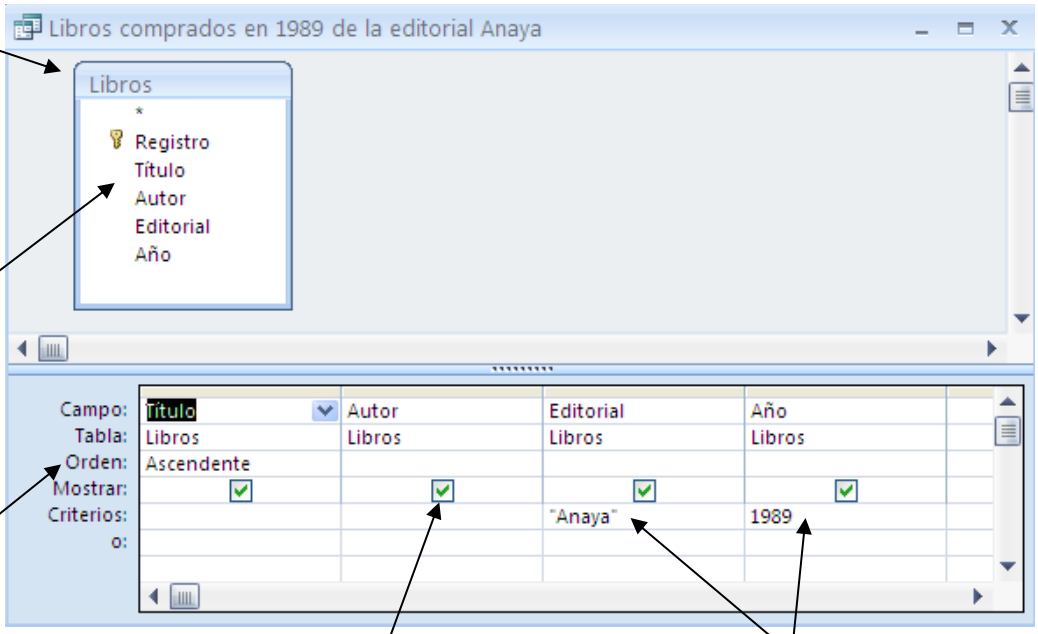
Las BBDD se crean para contener gran cantidad de información, que se guarda para poder ser consultada posteriormente. En múltiples ocasiones se necesita consultar una información concreta, sin que sea necesario recuperar todos los datos almacenados en las tablas de la BBDD.

Las **consultas** le permiten al usuario hacer preguntas a una BBDD, para recibir como respuesta una información que cumpla con determinados criterios (por ejmplo, consultar en la BBDD de un video-club películas de Pedro Almodóvar disponibles sin alquilar).

Al abrir una consulta en "Vista de diseño" podemos **revisar** cómo está definida una consulta ya creada, o **diseñar** una nueva consulta.

Aquí añadimos la tabla o las tablas que necesitemos en nuestra consulta. Para añadir tablas pulsamos el botón "mostrar tablas".

Haciendo doble clic en los campos de la tabla, éstos se van añadiendo a la rejilla QBE inferior.



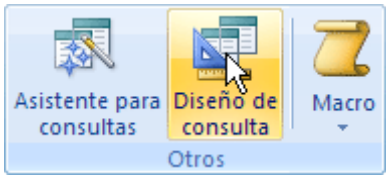
Cada uno de los campos puede ordenarse en ascendente o en descendentemente. Los resultados ofrecidos por la consulta pueden ordenarse por más de un campo.

Podemos elegir si mostrar o no mostrar en los resultados los campos incluidos en la consulta .

En la fila de Criterios se escriben las condiciones que deben cumplir los campos de la consulta.

CREAR UNA NUEVA CONSULTA.

Para crear una nueva consulta acudimos al menú "Crear" > bloque "Otros" → botón "Diseño de consulta".



A continuación agregamos a la consulta las tablas de la BBDD que formarán parte de esa consulta. Después, vmos arrastrando a la zona de diseño de la consulta los campos que deseamos obtener, fijamos el orden en el que queremos obtener los datos, decidimos si necesitamos mostrar ese campo, y especificamos los criterios que deben cumplir los campos consultados.

ACTIVIDAD 7.

Vamos a practicar el diseño de consultas creando las siguientes consultas sencillas en la BBDD "Biblioteca.accdb". Como consejo, asegúrate de guardar siempre tus consultas con nombres significativos, como por ejemplo "Libros comprados en 1989 de la editorial Anaya".

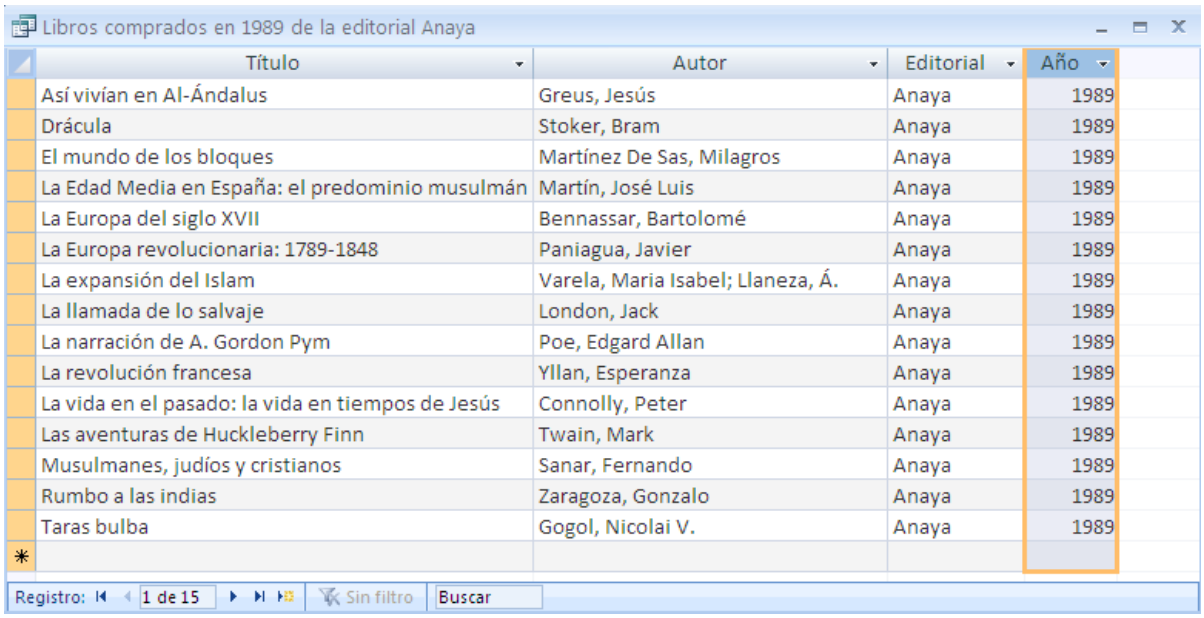
a) Libros de Benito Pérez Galdós, ordenados alfabéticamente por el título.

b) Libros de la editorial Castalia del año 1989, ordenados por el nombre del autor.
c) Libros adquiridos con anterioridad a 1950, mostrando únicamente título del libro y fecha.

EJECUCIÓN DE UNA CONSULTA.

Para obtener los resultados de una consulta previamente diseñada debemos **ejecutar** esa consulta. Hay diferentes formas de ejecutar una consulta:

a) Desde la "Vista de diseño", pinchando en el botón de ejecución de la consulta.
b) Abriendo una consulta en "Vista Hoja de datos" también se produce la ejecución de esa consulta.


Libros comprados en 1989 de la editorial Anaya

Título	Autor	Editorial	Año
Así vivían en Al-Ándalus	Greus, Jesús	Anaya	1989
Drácula	Stoker, Bram	Anaya	1989
El mundo de los bloques	Martínez De Sas, Milagros	Anaya	1989
La Edad Media en España: el predominio musulmán	Martín, José Luis	Anaya	1989
La Europa del siglo XVII	Bennassar, Bartolomé	Anaya	1989
La Europa revolucionaria: 1789-1848	Paniagua, Javier	Anaya	1989
La expansión del Islam	Varela, Maria Isabel; Llaneza, Á.	Anaya	1989
La llamada de lo salvaje	London, Jack	Anaya	1989
La narración de A. Gordon Pym	Poe, Edgard Allan	Anaya	1989
La revolución francesa	Yllan, Esperanza	Anaya	1989
La vida en el pasado: la vida en tiempos de Jesús	Connolly, Peter	Anaya	1989
Las aventuras de Huckleberry Finn	Twain, Mark	Anaya	1989
Musulmanes, judíos y cristianos	Sanar, Fernando	Anaya	1989
Rumbo a las indias	Zaragoza, Gonzalo	Anaya	1989
Taras bulba	Gogol, Nicolai V.	Anaya	1989

Registro: 1 de 15 Sin filtro Buscar

De haber diferentes consultas creadas, podemos acceder a todas ellas desde la ventana "Todos los objetos de Access". Una vez seleccionada la consulta deseada, se ejecuta mediante alguno de los métodos anteriormente indicados.

CRITERIOS Y EXPRESIONES ÚTILES PARA CONSULTAS ACCESS.

La siguiente tabla resume los principales operadores que podemos usar para construir los criterios que deberán obedecer los resultados de nuestras consultas:

DE COMPARACIÓN	ARITMÉTICOS	LÓGICOS	DE COINCIDENCIA
(<) Menor que	(+) Suma	(Y) Las dos proposiciones son verdaderas	(?) Sustituye un solo carácter
(=) Igual que	(-) Resta	(O) Una de las dos proposiciones son verdaderas	(*) Sustituye una cadena de números o caracteres
(>) Mayor que	(*) Multiplicación	(NO) La proposición es falsa	(#) Igual que ? pero para números
(<=) Menor o igual que	(/) División		
(>=) Mayor o igual que	(^) Exponente		
(<>) Distinto que	(MOD) Resto de la división		

Algunos ejemplos de criterios de consults son los siguientes:

CAMPO	CRITERIO	RESULTADO
CiudadEnvío	"Londres"	Muestra los pedidos enviados a Londres.
CiudadEnvío	"Londres" O "Newcastle"	Utiliza el operador O (Or) para mostrar los pedidos enviados a Londres o a Hedge End.
PaísEnvio	NO "EE.UU."	Utiliza el operador Negado (Not) para mostrar los pedidos enviados a países distintos de EE.UU.
FechaEnvío	Entre #5-Ene-20# Y #10-Ene-21#	Utiliza el operador Entre...Y (Between...And) para mostrar los pedidos enviados no antes del 5-Ene-20 ni después del 10-Ene-21.
FechaEnvío	#2/2/21#	Muestra los pedidos enviados el 2-Feb-21.
NombreEnvío	Como "S*"	Muestra los pedidos enviados a los clientes cuyo nombre empieza por S.
NombreCompañía	>="N"	Muestra los pedidos enviados a compañías cuyo nombre comienza por las letras N a Z.
FechaPedido	< Fecha( )-30	Utiliza la función Fecha (Date) para mostrar los pedidos con una antigüedad de más de 30 días.
AñoPedido	< Año(Fecha())-20	Muestra los pedidos con una antigüedad de más de 20 años.
FechaPedido	Año([FechaPedido])=2021	Utiliza la función Año (Year) para mostrar los pedidos con fechas de entrega en 2021.
FechaPedido	Año([FechaPedido])= Año(Ahora()) Y Mes([FechaPedido])= Mes(Ahora())	Utiliza las funciones Año (Year) y Mes (Month) y el operador Y (And) para mostrar los pedidos del año y el mes actual.
RegiónEnvío	Es Nulo	Muestra los pedidos de los clientes cuyo campo RegiónEnvío es Null(está vacío).
RegiónEnvío	NO es Nulo	Muestra los pedidos de los clientes cuyo campo RegiónEnvío contiene un valor.
NombreDestinatario	Como "*S"	Pedidos enviados a los clientes cuyos nombres terminan por la letra S.
NombreDestinatario	Como "*Importaciones"	Pedidos enviados a los clientes cuyos nombres terminan por la palabra "Importaciones".
NombreDestinatario	Como "[A-D]*"	Pedidos enviados a los clientes cuyos nombres empiezan de la A a la D.
NombreDestinatario	Como "*ar*"	Pedidos enviados a los clientes cuyos nombres incluyen la secuencia de letras "ar".
NombreDestinatario	Como "Casa Pac?"	Pedidos enviados al cliente con "Casa" como primera parte del nombre y un segundo nombre de 4 letras, de las cuales las 3 primeras son "Pac" y la última se desconoce.

ACTIVIDAD 8.

En esta actividad vamos a practicar la construcción de consultas. Guarda las consultas que construyas con los nombres Consulta 01, Consulta 02, etc. en su respectiva BBDD. Después, guarda las BBDDs con los nombres "Ejer8 Biblioteca.accdb" y "Ejer8 Animales.accdb".

En la BBDD "Biblioteca.accdb":

1) Consulta para seleccionar todos los libros de la editorial Anaya, ordenados alfabéticamente por Título.
2) Consulta que seleccione los libros de Antonio Buero Vallejo comprados antes de 1992.
3) Consulta que muestre los libros de Cervantes junto a los libros de Lope de Vega.
4) Consulta que seleccione los libros comprados entre 1990 y 1995 ambos inclusive.
5) Libros cuyo autor se llame Carlos, mostrándolos ordenados por año.
6) Obtener un listado de los números de registro de los libros de Cervantes.
7) Autores que hayan sido publicados por las editoriales Planeta o Salvat, ordenados alfabéticamente.
8) Obtener un listado de los libros adquiridos en la década de los 90, con número de registro posterior a 1000.
9) Consulta que muestre los libros cuyo título tenga una "a" en 3º lugar y una "e" en 5º lugar.

En la BBDD "Animales.accdb":

1) Consulta que seleccione las especies carnívoras que habitan en el bosque, no siendo necesario mostrar el campo "orden" de la especie.
2) Consulta que muestre sólo las especies que son ovíparas, ordenadas de mayor a menor peso.
3) Consulta que muestre los nombres y sexo de los animales que tenemos en el zoológico que sean Carnívoros, Roedores o Insectívoros, ordenados alfabéticamente por el nombre.
4) Consulta que seleccione los animales que midan entre 1 y 2 metros, ordenados de menor a mayor.
5) Consulta que muestre las especies que sean mamíferos, pero que no sean ni carnívoros ni roedores, que habiten en el bosque y cuyo tamaño supere el metro y medio.
6) Consulta que muestre las aves cuya primera letra del nombre de especie va de la A a la F, ordenados alfabéticamente por la especie.

PRÁCTICA 1

a) Crea una nueva base de datos, que debes guardar con el nombre "tus iniciales_practica1.accdb" (por ejemplo, "amp_practica1.accdb"). En ella crearás una primera tabla llamada CLIENTES. Esta tabla deberá tener la siguiente estructura de campos:

NOMBRE	TIPO	OTRAS PROPIEDADES A DEFINIR
CODCL	Numérico	Entero, Sin decimales, CLAVE
DENOM	Texto	50, Requerido SI
DIRECC	Texto	70
LOCAL	Texto	15
TELEF	Texto	15, Máscara de Entrada de número de teléfono
CODREF	Texto	Aplicar máscara adecuada observando los datos.

La tabla debe contener los siguientes datos:

CODC	DENOM	DIRECC	LOCAL	TELEF	CODREF
1	Modas Kuki	C/ Arbolito 20	Cádiz	(956) 25 98 60	REF-78-CAD
2	SuperVaqueros	Avda Constitución 30	Chiclana	(956) 13 89 60	REF-99-CHI
3	Textilandia	C/ Columela 30	Cádiz	(956) 28 16 68	REF-18-CAD
4	Confecciones Pepi	C/ Porvera 15	Jerez	(956) 33 69 84	REF-25-JER
5	Modas Fashion	C/ Flamequito 23	Jerez		REF-42-JER
6	Corte Francés	C/ Isla de León 2	San Fernando	(956) 89 56 45	REF-31-SFE

b) Necesitaremos 2 tablas más (ARTÍCULOS y VENTAS), pero para ahorrar tiempo las importarás de la BBDD "TablasPract1.accdb" en los archivos de los alumnos. Para importar las tablas sigue los siguientes pasos:

1) Menú "Datos externos" → bloque "Importar" → "Access".
2) Busca el archivo "TablasPract1.accdb", selecciónalo, y pulsa en "Aceptar".
3) Selecciona las 2 tablas existentes y pulsa "Aceptar".

c) Tras importar las tablas y disponer de los datos, toca estudiarlas a fondo. Ábrelas en "Vista de diseño" y en "Vista hoja de datos". Observa su estructura y los datos, la definición y propiedades de cada tipo de dato. Observa cómo se pueden relacionar las 3 tablas y a través de qué campos. AHORA ESTABLECE LAS CORRESPONDIENTES RELACIONES ENTRE LAS TABLAS.

d) Abre en modo "Vista hoja de datos" la tabla ARTICULOS y realiza las siguientes operaciones:

1) Haz que donde ponga ROJA ponga ROJO en una sola operación. (Utiliza la herramienta Buscar > Reemplazar).
2) Sin utilizar consultas, consigue quedarte sólo con aquellos productos medianos, azules, y de menos de 7000. (Utiliza filtros).
3) Con el resultado anterior, ordénalos ascendentemente por color, y en caso de igualdad, por precio.
4) Copia el resultado en un archivo de Word, y guárdalo con el nombre "Ejercicio d_4.docx" en tu carpeta de trabajo.
5) Vuelve a mostrar todos los registros con el orden original. (Ordenados por código)
6) Sin utilizar consultas, consigue un listado de los clientes de Jerez y de los de Cádiz.
7) Copia el resultado en un archivo de Word, y guárdalo con el nombre "Ejercicio d_7.docx" en tu carpeta de trabajo.

e) Realiza las siguientes consultas, guardándolas como Consulta 01, Consulta 02, etc.

1) → Muestra toda la información de las ventas realizadas por el Cliente "Textilandia".
2) → Muestra el nombre y el teléfono de los clientes cuya localidad empieza por "C" y cuya dirección contiene una "e".
3) Muestra el nombre, color, y tamaño de los artículos cuyo precio de venta sea al menos de 6000 €, ordenados por el Nombre.
4) → Muestra el nombre, color, y tamaño de los artículos cuyo nombre comienza por cualquier letra posterior a la E, ordenados por nombre.
5) Muestra los nombres de artículos rojos o azules que fueron vendidos en la primera quincena del mes de febrero.
6) Muestra la dirección de los clientes que compraron en septiembre, mostrando además el artículo que adquirieron.
7) → Muestra los clientes que tengan en la 3ª letra de su denominación una de estas letras: d, e, f, g, h, i, j, k, l, m, n. (Recuerda: [d-n]).

8) → Muestra el nombre de los clientes que en el primer trimestre realizaron compras inferiores a 10 unidades de cualquier artículo.
9) Muestra los artículos con precios de venta que superen las 6000 € y como mucho lleguen a 8000 €, ordenados de mayor a menor precio.
10) Muestra todas las ventas de los clientes que no tienen un número de teléfono registrado en la BBDD.
11) →¿A qué Cliente de Cádiz se le vendió en febrero una camisa mediana?
12) → Muestra los artículos azules y negros que sean pequeños.
13) → Muestra los artículos específicos de mujer, o los que siendo de hombre son negros, ordenados de más baratos a más caros.
14) Muestra las ventas de los clientes que no son de Cádiz, y que se realizaron después de junio y antes de septiembre.

EXPRESIONES EN UNA CONSULTA.

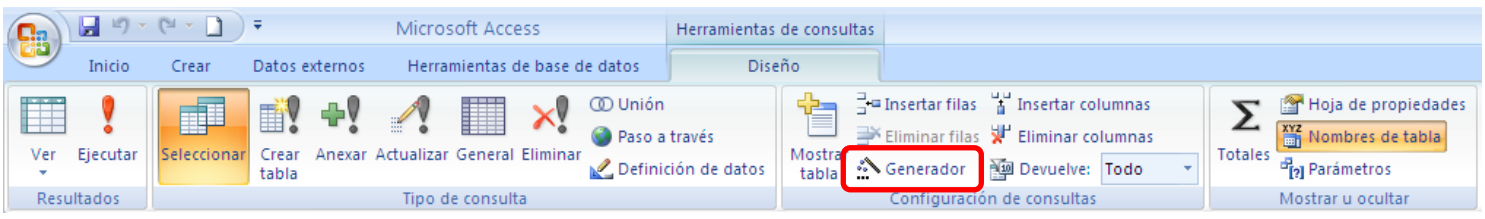
En las consultas, además de recuperar determinados campos y registros que cumplan ciertos criterios, se pueden realizar **cálculos** con los datos recuperados.

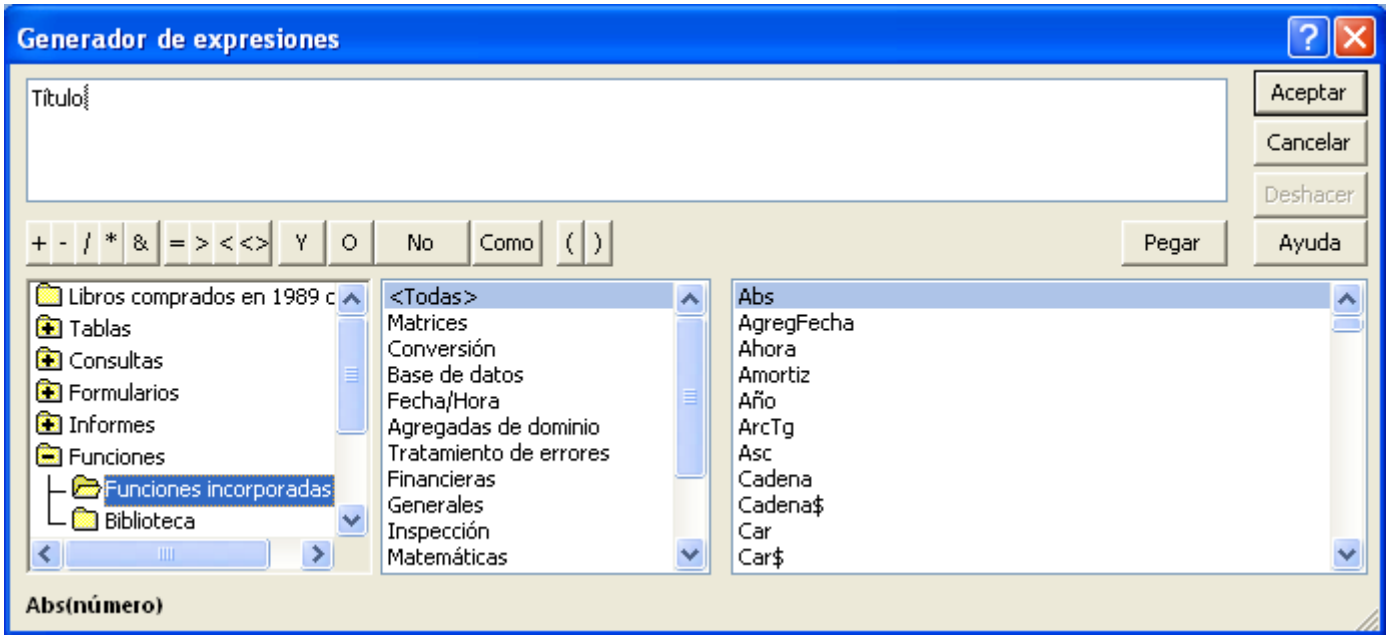
Funciones.

En Access enemos disponibles numerosas funciones para pprocesar los datos arrojados por una consulta, la mayoría de las cuales coinciden con las funciones propias de Excel. Algunas de las funciones disponibles con las siguientes:

FUNCIÓN	DECRIPCIÓN
Suma	Suma los valores
Promedio	Obtiene la media aritmética
Min	El valor más pequeño (mínimo)
Max	El valor más grande (máximo)
Cuenta	Cuenta los campos que no son nulos
Primero	Valor del campo del primer registro
Último	Valor del campo del último registro
Día	Devuelve el día del mes de la fecha especificada
Mes	Devuelve el nº de mes de la fecha especificada
Año	Devuelve el año de la fecha especificada
Fecha	Devuelve la fecha actual, tomándola del SS.OO.
DiaSemana	Devuelve el nº de día de la semana (Lun→1, Do→7)
Izq	Devuelve los caracteres a la izq. del texto dado
Der	Devuelve los caracteres a la der. del texto dado
Longitud	Devuelve el nº de caracteres de un campo tipo texto.

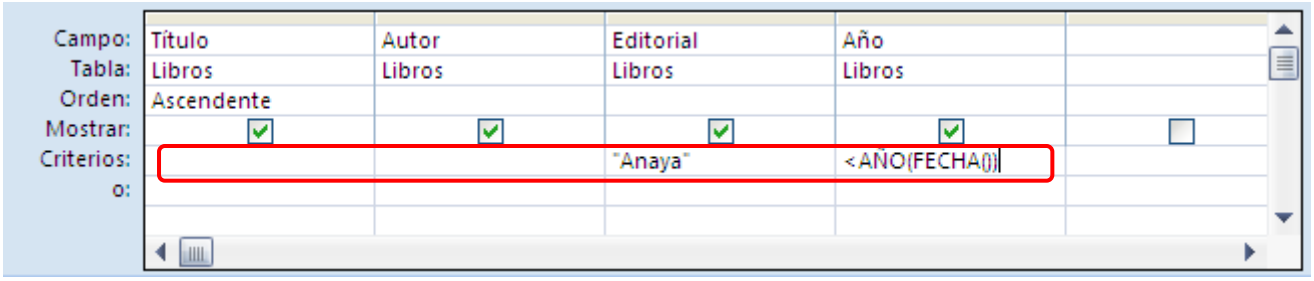
Para revisar las funciones disponibles en Access, acudimos a la herramienta "Generador de expresiones", accesible desde la "Vista de Diseño de Consultas":





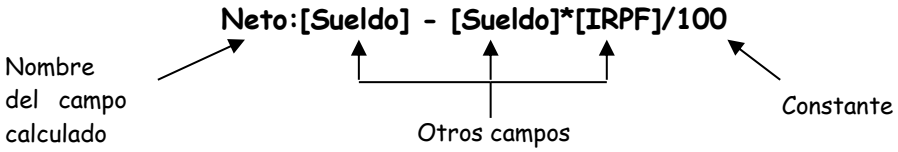
Las funciones pueden utilizarse de dos formas:

a) Como parte del criterio, para escribir la condición qué deben cumplir los registros de la consulta. (Por ejemplo, los artículos vendidos hasta el día de ayer).

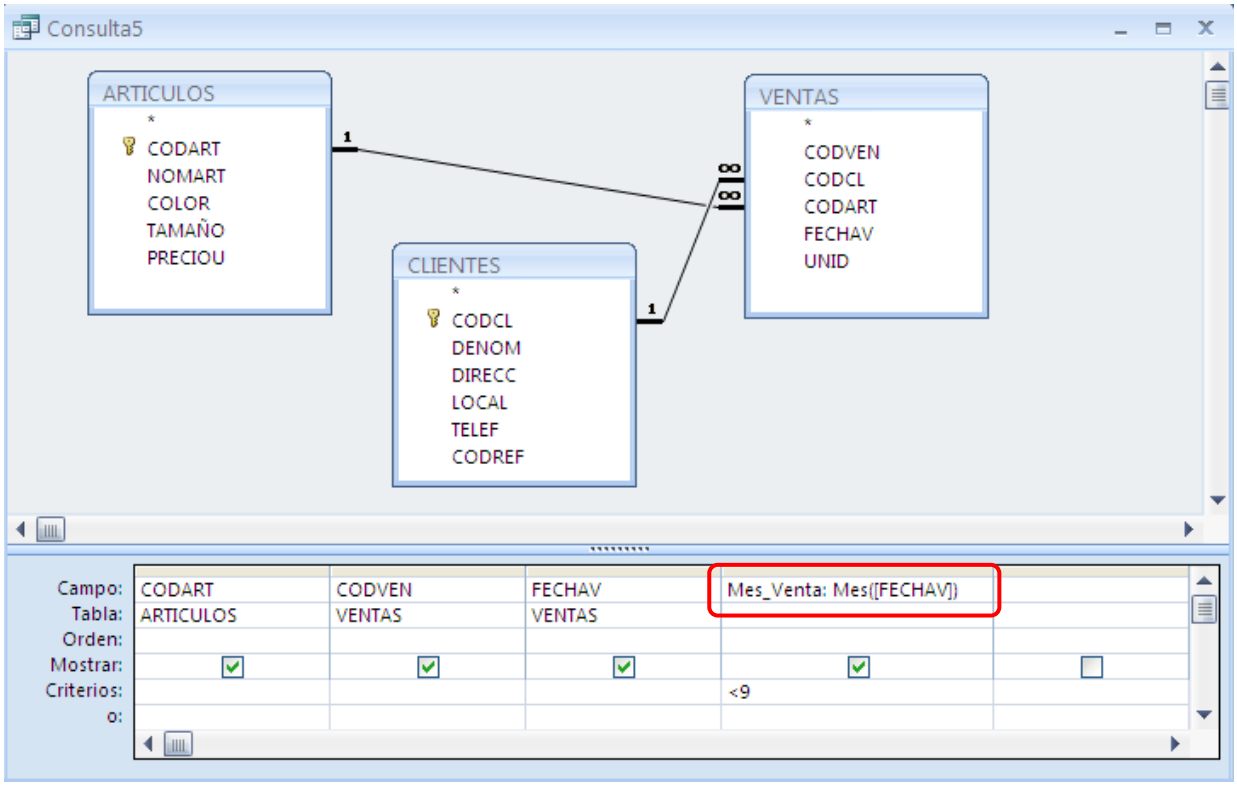


b) Utilizando campos calculados. Un campo calculado en un nuevo campo que se calcula operando los valores de otros campo. Para crear un campo calculado seguimos estos pasos:

1) En la ventana de "Diseño de consultas" nos situamos en una columna de la tabla de consultas que esté vacía.
2) Escribimos el nombre del campo calculado.
3) Escribimos el separador dos puntos (:), que sirve para indicar dónde termina el nombre de campo calculado, y empieza la fórmula que lo define.
4) Escribimos la fórmula para obtener el valor del campo calculado. Esta fórmula puede incluir otros campos, constantes, operadores (+, -, *, /), y funciones.



Un ejemplo de campo calculado sería aquel que obtenga el mes en el que se realizó una determinada venta (y que seleccione sólo aquellas ventas anteriores a septiembre).



Totales (registros agrupados).

Access nos ofrece la posibilidad de agrupar registros con un valor común en uno de sus campos, y realizar un cálculo de totales de los valores de otro de sus campos. Para poder realizar un cálculo de totales deben darse las siguientes condiciones:

1) Que exista más de un registro donde al menos uno de sus campos se repita, siendo los otros diferentes.
2) Que tengan un campo numérico con el que se puedan realizar cálculos (Suma, Max, Min, Media y Contar).

CL	ART	FECHA	UNID
001	100A	01/01/01	10
001	100A	01/02/01	20
001	200B	15/01/01	10
001	200B	25/02/01	7
002	100A	07/01/01	15
002	200B	07/01/01	8

Por ejemplo, en la tabla superior se podría agrupar por Clientes y calcular la media de unidades vendidas por cada uno....

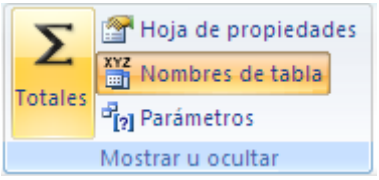
CL	UNID
001	11,75
002	11,5

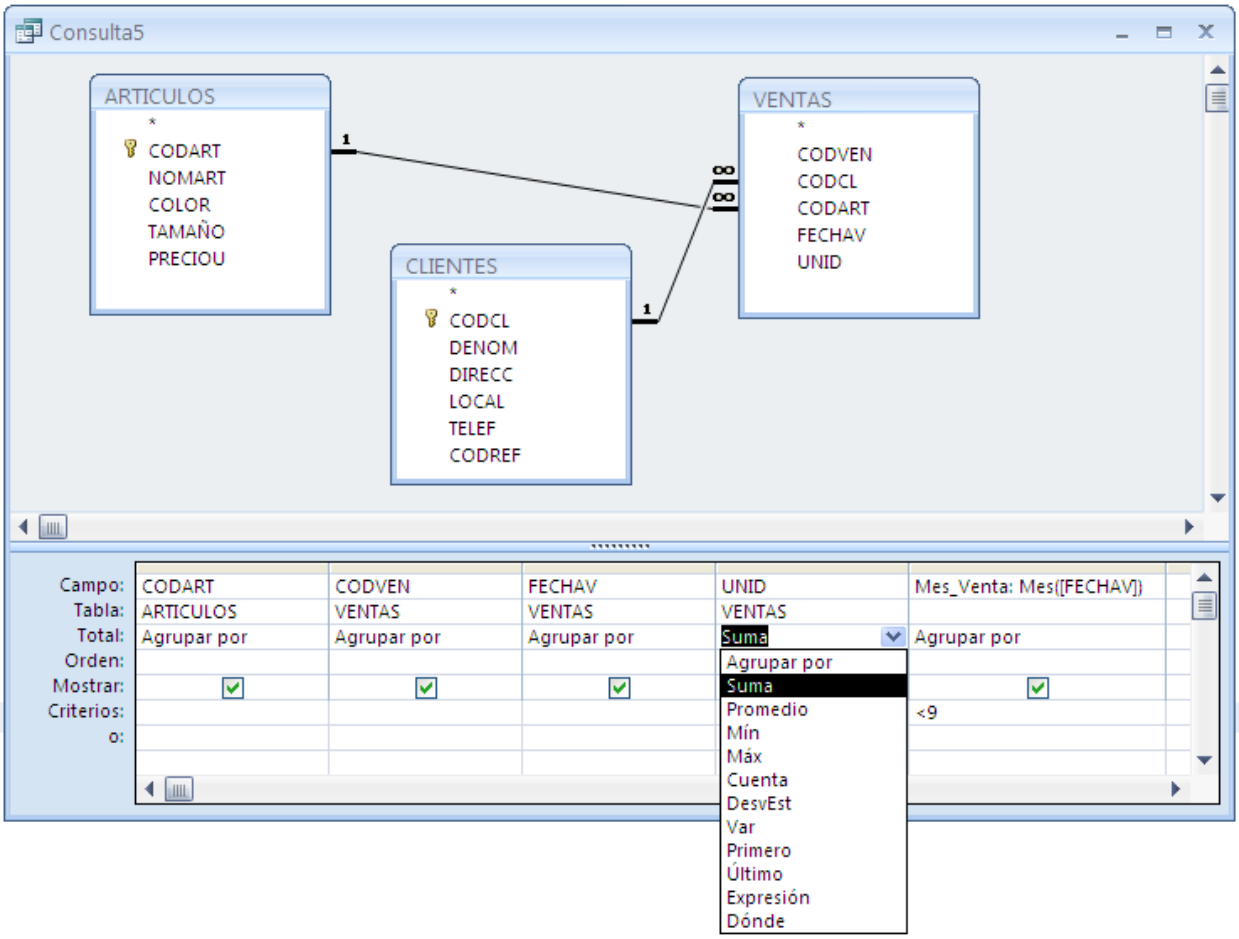
... o agrupar por Artículos y calcular el total de unidades vendidas de cada uno de ellos.

ART	UNID
100A	45
200B	25

Para realizar un cálculo de totales debemos seguir los siguientes pasos:

1) En la ventana de "Diseño de consultas", acudimos al menú "Diseño" > bloque "Mostrar u Ocultar" > botón "Totales".
2) En la tabla de "Diseño de consulta" aparece una nueva fila llamada "Total".
3) En el campo a operar, sobre la fila "Total", escogemos de la lista la operación a realizar con el campo.
4) En el resto de campos que no queremos operar dejamos seleccionada la opción "Agrupar por".





ACTIVIDAD 9.

Abre la base de datos "Socios_del_club.accdb". Crea las siguientes consultas solicitadas, guardándolas como Consulta01, Consulta02, etc. Guarda la BBDD como "Ejer10_Socios_del_club.accdb".

1) Consulta que muestre el nombre, apellido, y teléfono (en este orden) de cada uno de los socios. Los datos deben aparecer ordenados alfabéticamente por apellido.

2) Consulta que muestre el nombre, apellido, y teléfono de los socios con nivel de juego senior. El campo nivel de juego no debe aparecer en el resultado de la consulta.

3) Consulta que muestre los campos nombre, apellido, población, y teléfono de los socios de nivel de juego medio cuya cuota anual sea superior a 3 €. Los datos deben aparecer ordenados primero por localidades y luego por apellido.

4) Consulta que seleccione los socios que no estén federados y que pertenezcan a la categoría Medio viéndose todos los campos.

5) Consulta que seleccione Nombre, apellidos, dirección, localidad y teléfono de los socios que no sean de Logroño.

6) Consulta que seleccione los socios de Arnedo, Calahorra y Alfaro ordenados por localidad.

7) Consulta que seleccione los socios que no sean ni de Logroño ni de Calahorra.

8) Consulta que seleccione a los socios solteros de Arnedo.

9) Consulta que seleccione a los socios solteros de Arnedo y a los socios solteros de Logroño.

10) Consulta que seleccione los socios cuya cuota esté comprendida entre 3 y 6 euros.

11) Consulta que seleccione los socios que residan en las localidades comprendidas alfabéticamente entre Arnedo y Logroño.

12) Consulta que calcule la edad de los socios. Los datos aparecerán ordenados en función de la edad. La consulta se llamará Edad de los socios. Ayuda: la edad se calculará por la resta de la fecha de hoy y la fecha de nacimiento, el resultado es un número de días que se deberá dividir entre 365. Para obtener el resultado sin decimales accedemos a las propiedades del campo calculado (botón derecho sobre el campo en la pantalla de diseño abajo) y elegimos tipo fijo y sin lugares decimales.

13) Consulta que seleccione los socios de más de 30 años. Llamarla Socios de más de 30 años. Ayuda: guardar la consulta anterior con otro nombre, socios de más de 30 años, modificarla y añadir la condición de más de 30 años.

14) Consulta que seleccione los socios con una antigüedad en el club de más de 3 años. Llamarla Socios con más de 3 años de antigüedad. Ayuda: crearemos un campo calculado llamado antigüedad por resta de la fecha de hoy y el campo alta de socio. Estableceremos en ese campo el criterio de > de 3 años

15) Consulta que ordene los socios por la letra del DNI. Llamarla Socios ordenados por letra del DNI. Ayuda: Creamos un campo calculado llamado letra_DNI que contenga la letra del DNI del campo DNI. Una vez obtenido se ordena por ese campo. Para extraer de una cadena un conjunto de caracteres por la derecha emplear la función Der(cadena;número), función que se encuentra en funciones de texto. Ejemplo Der("Helena";4) devuelve "lena".

16) Consulta que calcule, con IVA incluido, la cuota que hay que cobrar a cada socio, al trimestre. Llamarla Cuota Trimestre.

A continuación, sin usar consultas, realizar los siguientes filtrados y ordenaciones:

- Ver a los socios de Arnedo.
- Ver los socios cuyo nombre empiece por 'A'.
- Ver los socios con apellido DIAZ.
- Ordenar por localidad.
- Ver todos los que no son de Logroño.
- Ver todos los socios que tengan un apellido PEREZ y que no sean de Logroño.

ACTIVIDAD 10.

Realiza las consultas avanzadas que se proponen a continuación. Probablemente tendrás que usar funciones, campos calculados, y registros agrupados (totales).

1) Base de datos "Animales.accdb":

   a) Crea un campo calculado que te muestre el tamaño de los animales, pero en lugar de en centímetros, en metros. Llámalo Tamaño_en_metros.

   b) Obtén la media del peso de todos los animales carnívoros (tienes que usar la herramienta de 'Totales').

   c) Crea un campo llamado IMC (Índice de Masa Corporal), en el que se calcule el IMC del animal.
   d) IMC = peso (Kg) / (altura (m))^2

   e) Crea un campo llamado peso redondeado, donde se obtenga el peso sin decimales.

   f) Averigua cuántos mamíferos hay en nuestro zoo.

   g) Crea un campo que cuente el número de caracteres anotado en el campo Observaciones de cada animal.

   Guarda la BBDD como "Ejer9 Animales.accdb".

2) Base de datos "Contactos.accdb":

   a) Obtén el listado de personas nacidas después de 1960.

   b) Calcula los ingresos medios de aquellas personas con nivel educativo de bachillerato.

   c) Crea un campo calculado donde obtengas el día de la semana en que nacieron cada uno de los contactos (lunes, martes,…, domingo).

   d) Obtén el prefijo de los números de teléfono fijo de cada contacto.

   e) Contar el número de mujeres sin hijos.

   f) Obtén las personas cuyo sueldo es superior al sueldo medio.

   g) Obtén el listado de teléfonos móviles de aquellas personas que no disponen de dirección de correo

   Guarda la BBDD como "Ejer9 Contactos.accdb".

PRÁCTICA 2.

a) Crea una nueva base de datos en tu carpeta de trabajo con el nombre "tus iniciales_practica2.accdb" (por ejemplo, "amp_practica2.accdb").

b) Importa las tres tablas (clientes, artículos, y ventas) de la práctica 1.

c) Realiza las siguientes consultas, guardándolas como Consulta 01, Consulta 02, etc.

1) → Muestra la tabla VENTAS añadiéndole un campo que visualice únicamente:
   - El año en que se realizó cada una de las ventas.
   - El mes en que se realizó cada una de las ventas.
   - El día de la semana (lunes, martes, etc.) en que se realizó cada una de las ventas.
2) Localiza las ventas realizadas hace más de 1 mes (Hoy – 30 días).
3) Localiza las ventas realizadas hace más de 1 semana.
4) Localiza las ventas realizadas hace menos de 1 mes.
5) Localiza las ventas realizadas en el mes de febrero.
6) → Averigua el número de letras del nombre de cada artículo.
7) Muestra la suma de unidades vendidas de cada cliente (ejemplo de los apuntes).
8) → Muestra la media de cada artículo vendidos a cada cliente.
9) → Averigua cuántos artículos hay de cada color.
10) Averigua el total de unidades vendidas de cada artículo para cada cliente, ordenado por cliente.
11) → Muestra los artículos con una rebaja de un 10% en su precio.
12) → Averigua cuánto vale el artículo más caro de cada tipo (el tipo lo indica el nombre).

2.7. FORMULARIOS.

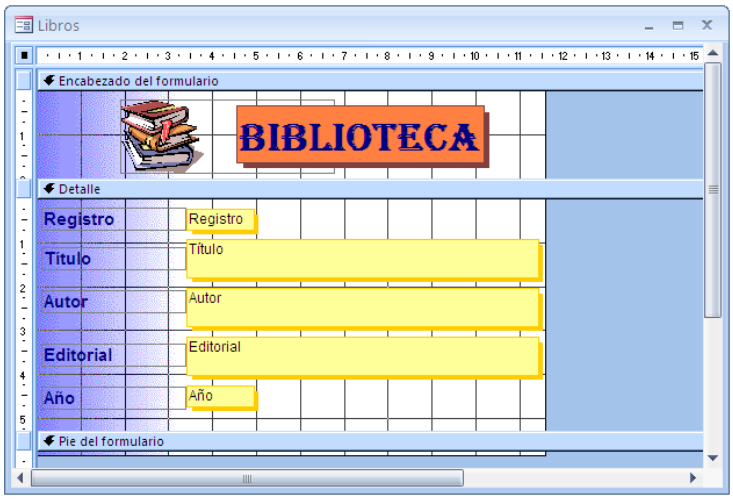
Hasta ahora hemos estudiado que la información de una BBDD se almacena en forma de tablas. En las tablas se pueden realizar operaciones de adición, modificación, y eliminación de registros, operaciones de filtrado, ordenaciones, búsquedas, consultas, etc. Sin embargo, Access también ofrece los formularios, una herramienta para presentar los datos de forma más atractiva que las tablas. Un **formulario** es básicamente una ventana de Windows con aspecto de "ficha", donde se pueden colocar etiquetas, cuadros de texto, checkbox, listas desplegables, botones, y otros elementos típicos de Windows. Las aplicaciones de los formularios son:

- Presentar una ficha individual por registro, con una apariencia más atractiva.
- Ahorrar mucho tiempo en la inserción y modificación de datos.
- Crear un sistema de menús o pantallas enlazadas, creando algo parecido a un programa.

ACTIVIDAD 11.

Visualiza el formulario "Libros" de la BB.DD "Biblioteca.accdb". Observa también la "Vista de diseño", y familiarízate con su aspecto y opciones.

Vista de Formulario



Vista de diseño.

CREAR UN FORMULARIO UTILIZANDO EL ASISTENTE.

Access ofrece varias opciones para crear formularios. Una de ellas es utilizar el asistente para formularios. El asistente para formularios es el método más rápido y sencillo, pero también el más limitado en cuanto a posibilidades. Los pasos que debemos seguir ara crear un formulario desde el asistente son los siguientes:

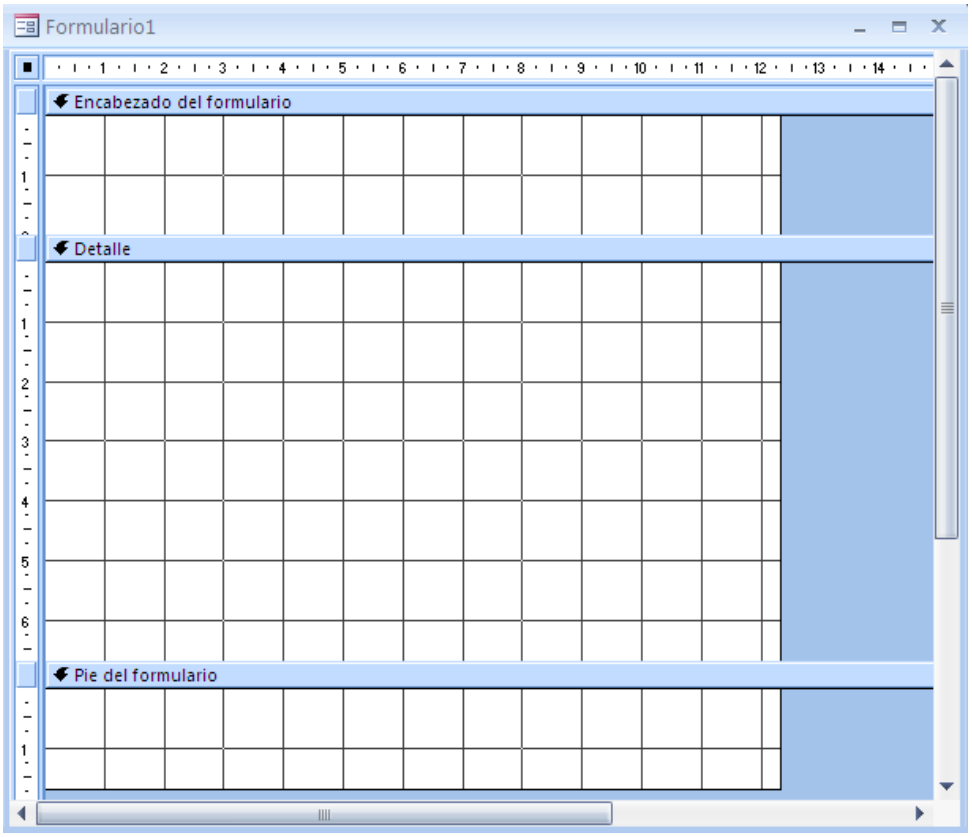
1) Acudimos a menú "Crear" > bloque "Formularios" > "Más Formularios" > "Asistente para formularios".
2) Seleccionamos las tablas de las que beberá el formulario, y los campos que queremos incluir en el formulario.
3) Elegimos la distribución deseada para el formulario (en columnas, tabular, hoja de datos, justificado).
4) Seleccionamos el estilo a aplicar al formulario, entre una gran variedad de estilos prediseñados. Posteriormente podremos modificar o editar el estilo seleccionado.
5) Le damos un nombre al formulario. También tenemos la opción abrir el formulario y modificarlo en "Vista de diseño".Tras obtener el formulario podemos acudir en cualquier momento a la "Vista de Diseño" para editarlo.

Por cierto, que pdemos crear un formulario de forma todavía más sencilla simplemente seleccionando la tabla de interés y acudiendo a la herramienta "Autoformulario". (Esta opción podría no estar disponible dependiendo de la versión de Access con la que estemos trabajando).

CREAR UN FORMULARIO EN VISTA DE DISEÑO.

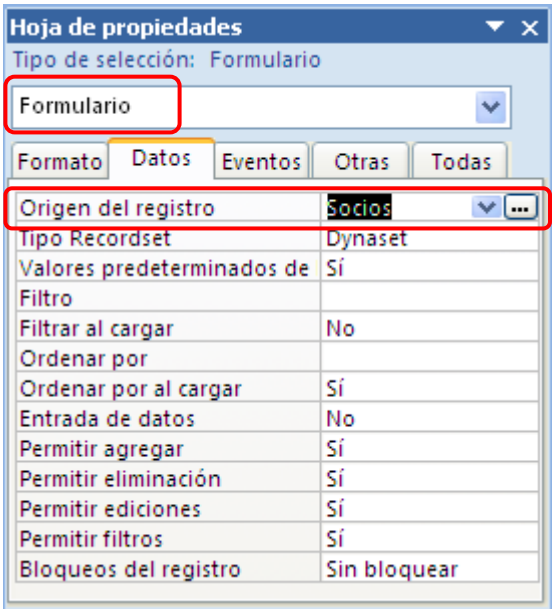
1) Para hacer un formulario desde la "Vista de diseño", seleccionamos el menú "Crear" > bloque "Formularios" > "Diseño de formulario".

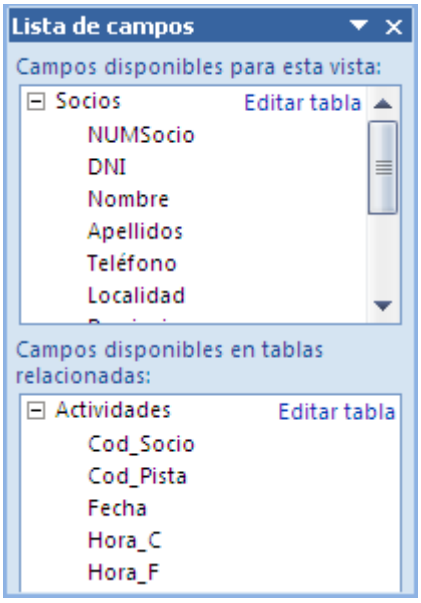
2) A continuación, hacemos visibles todas las zonas del formulario, a saber, detalle, encabezado, y pie del formulario. Esto podemos hacerlo desde el menú "Herramientas de diseño de formulario" > pestaña "Organizar" > bloque "Mostrar u ocultar" > Botón "Encabezado o pie de Formulario".



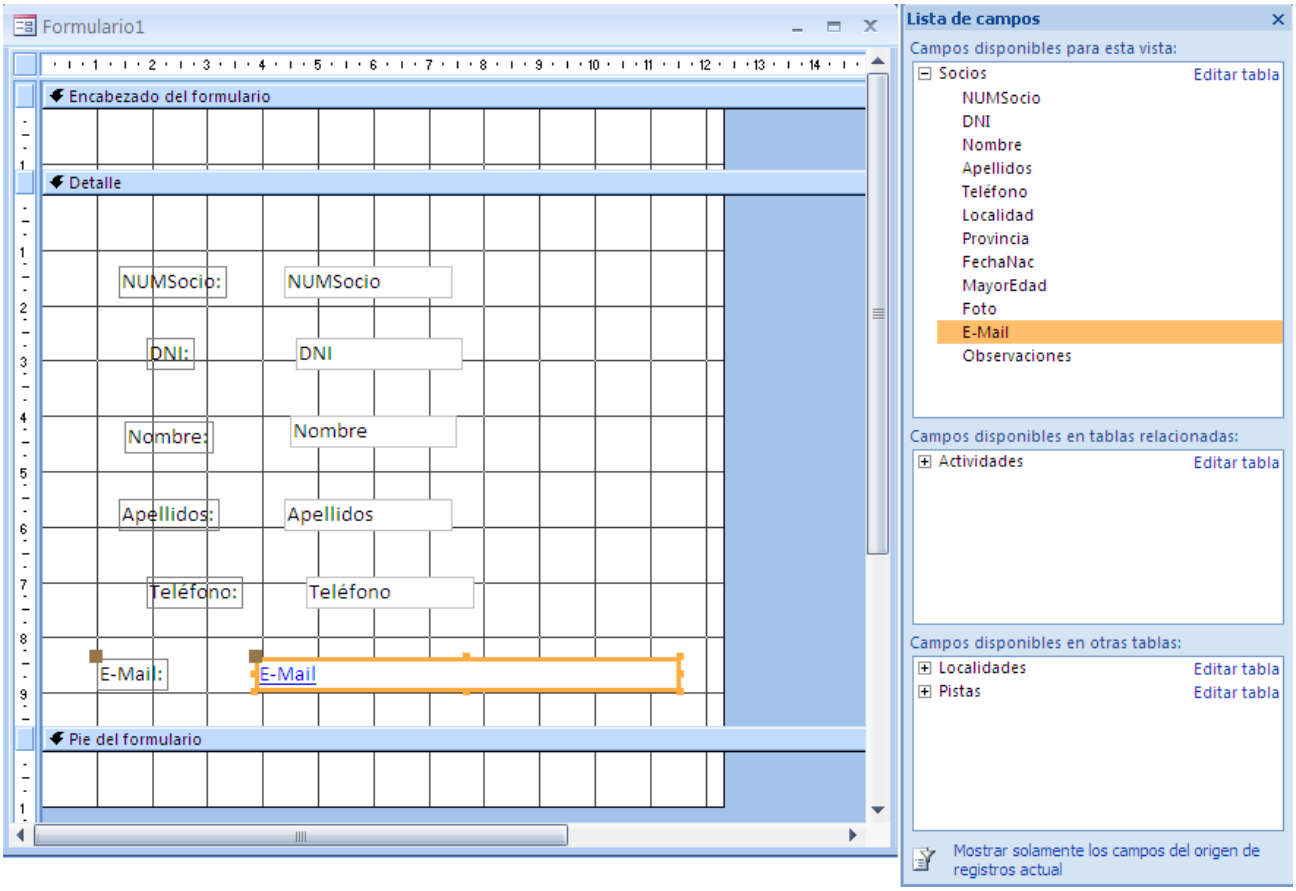
3) Ahora ajustamos el tamaño de cada una de las zonas, según deseemos.

4) Indicamos desde qué tabla se va a generar el Formulario:
   - Acudimos al menú "Herramientas de diseño de formulario" > pestaña "Diseño" > bloque "Herramientas" > "Hoja de Propiedades".
   - En la hoja de propiedades del formulario, seleccionamos la pestaña "Datos" > propiedad "Origen del Registro", y seleccionamos la tabla a utilizar.
   - Una vez seleccionada la tabla, acudimos a menú "Herramientas de diseño de formulario" > pestaña "Diseño" > bloque "Herramientas" > "Agregar Campos Existentes".



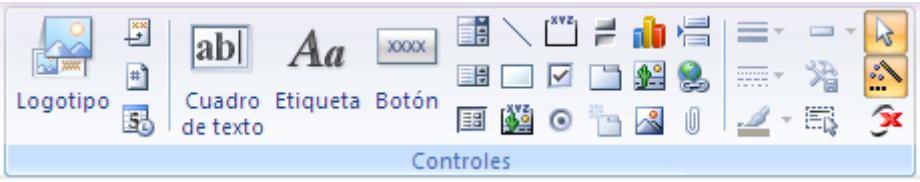


5) Arrastramos los campos deseados desde la ventana de campos de la tabla hasta la zona de "Detalle del formulario". Automáticamente se crea un cuadro de texto y una etiqueta para cada campo seleccionado.



OBJETOS PARA LA CONSTRUCCIÓN DE UN FORMULARIO.

Access ofrece diversos objetos que podemos usar para construir nuestros formularios. Dependiendo de la versión de Access, algunOs de estos objetos podrían estar o no estar disponibles.



Para probar la funcionalidad de cada tipo de objeto, vamos a ir añadiéndolos uno a uno a un formulario de ejemplo. Si pinchamos con el botón derecho del ratón sobre cualquier objeto añadido, y elegimos la opción "Propiedades", podremos acceder al conjunto de propiedades que definen la apariencia y comportamiento del objeto en cuestión, como son el color del objeto, el color y el tamaño de la fuente, su posición exacta, efectos, origen del control, y muchos más, en función del tipo de objeto.

Etiquetas. Las etiquetas se utilizan para mostrar textos estáticos. Su formato se puede variar mediante las opciones clásicas de formato (tipo, tamaño, color de fuente, fondo, negrita, cursiva, subrayado, etc.).

Cuadros de texto. Son objetos que muestran información asociada al campo de una tabla, por lo que su contenido cambia con cada registro. Podemos obtenerlos pinchando y arrastrando desde la Lista de Campos, o tomarlos de las herramientas de diseño para posteriormente seleccionar el campo sobre el que actuará en la propiedad "origen del control del cuadro de texto".

Grupos de opciones. Sirven para crear grupos de controles del tipo "botón de opción". Suelen asociarse a campos de tipo Sí/No, permitiendo visualizar varias opciones de las cuales el usuario puede elegir una. También se suelen utilizar con campos numéricos. Las distintas opciones pueden adoptar varios aspectos: Casillas de verificación, botones de opción, etc. Este objeto presenta un asistente muy útil para su correcta utilización. (Por cierto, que cuando asociamos este objeto a un campo de tipo Sí/No, la opción "Sí" toma el valor -1 y la opción "No" toma el valor 0).

Botones de alternar. Se tratan de los clásicos botones de las herramientas de Windows para seleccinar o deseleccionar una opción. Va asociado a un campo de tipo Sí/No. Cuando toma el valor "Sí", el botón aparecerá hundido.

Botones de opción. Se usan para activar o desactivar opciones. Se suelen asociar a campos de tipo Sí/No mediante la propiedad "Origen del Control". Se suelen concentrar en grupos de opciones. Sólo puede haber una opción seleccionada.

Casillas de verificación (checkbox). Son muy similares a los anteriores, pero en este caso sí que puede haber más de una casilla activada a la vez.

Cuadros combinados (controles de formulario). Son listas desplegable donde podemos ver todos los valores posibles y seleccionar cualquiera de ellos. Los valores posibles los pueden obtener de una tabla, de una consulta, se les pueden proporcionar por escrito, etc. Disponen de un asistente que salta al activarlo (siempre que esté instalado el complemento "Asistentes Adicionales"), y es de gran ayuda para guiar al usuario en su utilización.

Cuadros de lista (controles de formulario). Son muy similares a los anteriores, solo que en lugar de desplegarse muestran todas las opciones en una sola línea. Otro aspecto iferenciador es que, en

los cuadros combinados tenemos la opción de escribir y añadir información o de seleccionar de la lista, mientras que en los cuadros de lista tan solo podemos elegir una opción de entre una lista cerrada.

Botones. Los botones permiten realizar acciones, como llamar a un formulario, llamar a una consulta, llamar a otro programa, cerrar ventanas, etc. En los botone spodemos elegir el texto que aparecerá sobreinpreso en ellos, o elegir una imagen representativa. (El asistente de controles nos guiará para elegir dichas opciones). Los botones también permiten mostrar un texto de ayuda que nos indique su función cuando pasamos el puntero sobre él. Esta funcionalidad podemos activarla en el menú "Propiedades" > Pestaña "Otras" > "Texto de Ayuda del Control".

Las acciones de llamar a otro formulario, a una consulta, o a un informe requieren que éstos existan previamente y que tengan un nombre. Por tanto, antes de añadir un botón que "salte" a un cierto informe lamado "Informe A", ese informe debería previamente estar creado y guedado con el nombre "Informe A". Por ello, los botones de llamadas entre componentes se suelen dejar siempre para el final del diseño del formulario.

Imagen. Las imágenes son objetos para mostrar imágenes estáticas. Serán imágenes de decoración que no cambiarán al cambiar de Por tanto, se usan para mostrar imágenes que no van asociadas a una tabla o consulta).

Marcos de objetos independientes. Objetos similares a los anteriores, pero que no sólo sirven para mostrar imágenes, sino cualquier objeto externo (documentos, hojas de cálculo, sonidos, vídeos, presentaciones, etc.). Son independientes, lo que significa que el objeto que muestren será estático y no cambiará en función del registro.

Marcos de objetos dependientes. En este caso son marcos para contener objetos dependientes, lo que significa que el objeto mostrado dependerá del registro que esté activado en cada momento, para lo cual, el objeto debe estar en la tabla con el resto de datos. Por ejemplo, con estos componentes podemos mostrar una foto distinta por cada registro.

Saltos de línea. Sirven para colocar un final de página y comienzo de una nueva.

Controles de ficha. Se utilizan para crear varias fichas con sus correspondientes solapas. No se asocian a ninguna tabla o consulta, sino que permanecen fijas. El texto de cada solapa se cambia en la propiedad "Título". Sobre estas fichas podemos colocar cualquier otro control.

Subformularios / Subinformes. Son formularios secundarios que se usan para mostrar en ellos los datos de otra tabla o consulta relacionada con la del formulario principal. Si no hemos hecho la relación de tablas previamente, la debemos establecer al colocar el Subformulario (en caso contrario mostrará todos los datos a la vez para cada registro de la tabla principal).

Líneas. Sirven para colocar líneas rectas a modo de decoración.

Rectángulos. Se usan para decorar con rectángulos cuyo aspecto podemos modificar mediante sus propiedades.

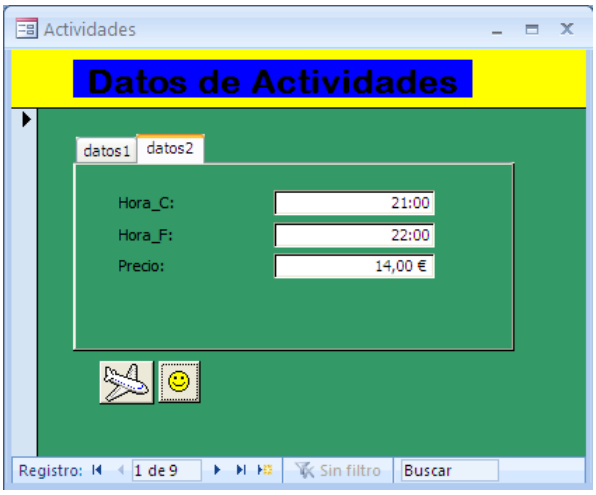
ACTIVIDAD 12.

Vualve a leer detenidamente la descripción de cada uno de los objetos disponibles para construir formularios hasta haberlos entendido correctamente, y a continuación, diseña los siguientes formularios en la BBDD "Club_Depotivo.accdb". Cuando termines, guarda la BBDD como "Ejer12 Club_Depotivo.accdb".

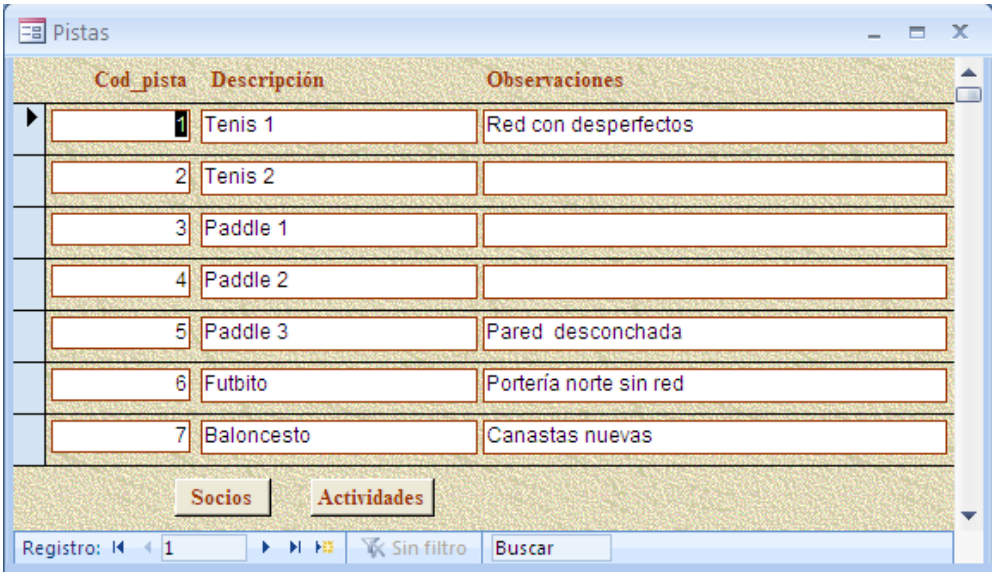
Algunas observaciones:

- Los 2 botones del formulario "Actividades" enlazan con los otros dos formularios ("Pistas" y "Socios").
- La pestaña "Datos2" del formulario "Actividades" contiene los otros tres campos de la tabla "Actividades".
- El formulario "Pistas" lo empezaremos a construir empleando la utilidad "Crear un formulario utilizando el asistente". Después los modificaremos en "Vista de Diseño" hasta conseguir el acabado final deseado.
- El cuadro combinado localidad del formulario "Socios" toma los valores de la tabla "Localidades".
- En el formulario "Socios", tanto el grupo de opciones como la casilla de verificación se basan en el campo "MayorEdad".

a) Formulario "Actividades":



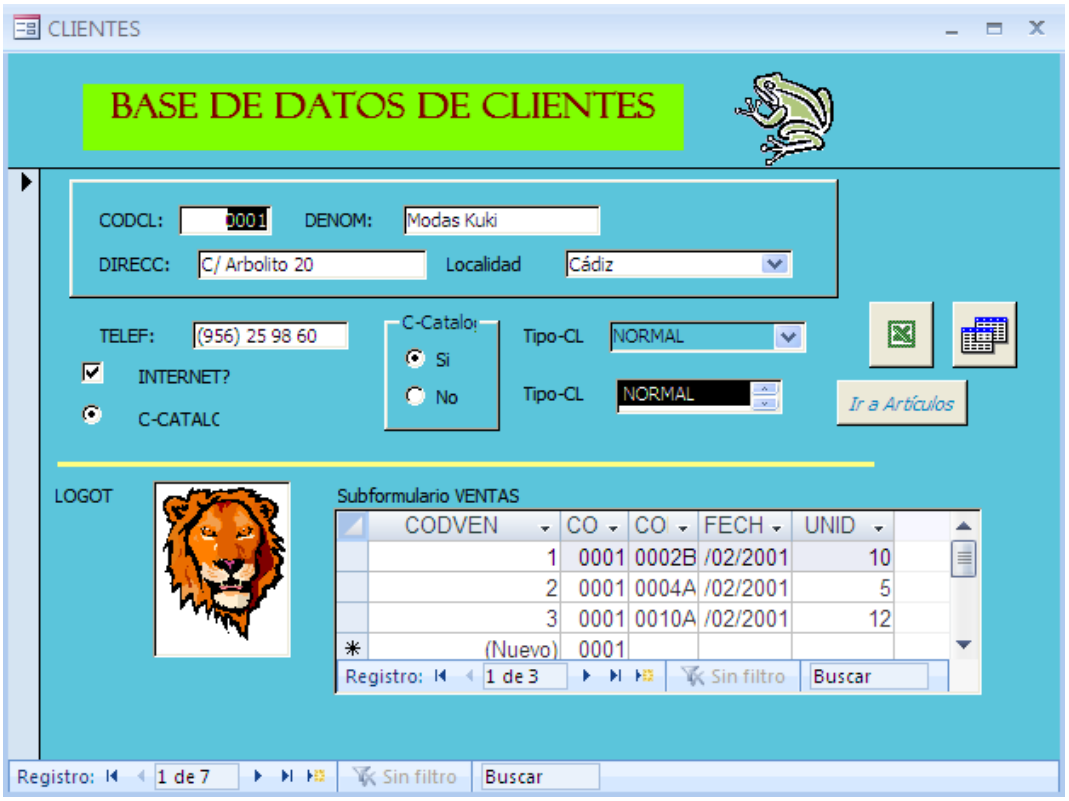
b) Formulario "Pistas":



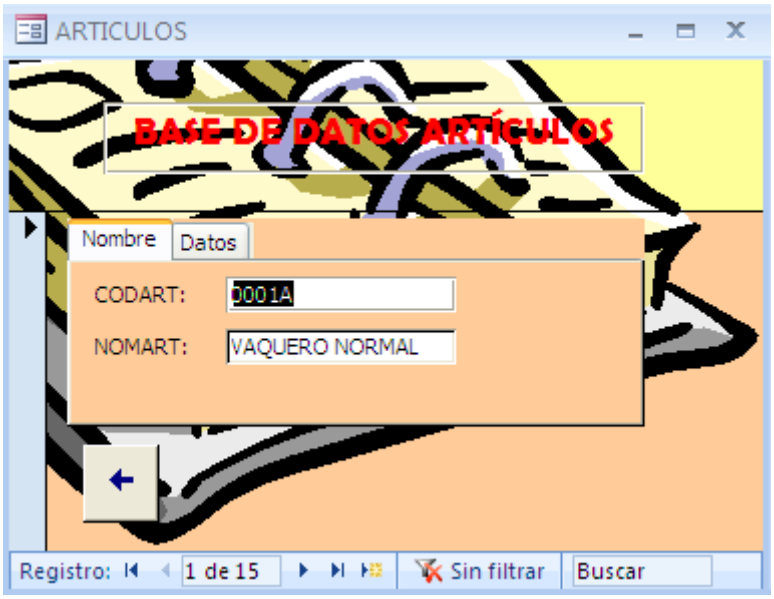
c) Formulario "Socios":

PRÁCTICA 3.

En esta práctica vamos a construir los formularios mostrados en los modelos sin la ayuda del asistente.



Formulario "Clientes".



Formulario "Artículos".

Para construir estos formularios, sigue los siguientes pasos:

a) Crea en tu carpeta de trabajo una nueva base de datos con nombre "tus iniciales_practica3.accdb" (por ejemplo, "amp_practica3.mdb"), e importa las tablas de la práctica 1 ("Clientes", "Artículos", y "Ventas").

b) Modifica la tabla "Clientes":
- Añade los campos LOGOT (Objeto OLE), CCATAL (Sí/no), INTERNET (Sí/no), TIPOCL (Texto, 5).
- Rellena aleatoriamente los campos CCATAL e INTERNET.
- Rellena el campo LOGOT con una imagen en al menos 3 registros. Para ello, acude al menú "Insertar" > "Objeto" > "Crear nuevo" > "Imagen Microsoft Word".

c) Crea una nueva tabla llamada "Localidades" con dos campos: NÚMERO (Autonumérico) y LOCALIDAD (Texto). Rellena los registros según la figura:

NUMERO	LOCALIDAD
1	Cádiz
2	San Fernando
3	Chiclana
4	Puerto Real
5	Puerto Sta Mª
6	Jerez
7	Rota
(Nuevo)	

d) Crea un nuevo formulario en "Vista de Diseño" con el nombre "Clientes". Sigue estas pautas:
- Activa el Encabezado y Pie de Formulario, eliminando el Pie.
- Establece al encabezado y al detalle el color de fondo celeste (14402907).
- Asocia el Formulario a la tabla "Clientes", mediante la propiedad "Origen del registro".

e) Coloca en el encabezado una etiqueta que contendrá el título, y establece en ella las siguientes propiedades:
- Título: Base de Datos de Clientes (Felix Titling, Negrita, Rojo).
- Estilo de Fondo: Normal.
- Color de Fondo: Verde (65408).

f) Coloca en el Encabezado una Imagen.
- Modo de cambiar tamaño: Zoom
- Distribución de la Imagen: Centro

g) Coloca en la sección Detalle mediante Cuadros de Texto los siguientes campos: CODCL, DENOM, DIRECC y TELEF. Si los arrastras desde la tabla se crearán automáticamente.

h) Coloca un Rectángulo para el cuadro con relieve y fija el efecto especial "con relieve".

i) Coloca una Casilla de Verificación y un Botón de Opción para los campos INTERNET Y CCATAL.
- Debes asociarles los campos correspondientes en Propiedades mediante "Origen del Control".
- Cambia el texto de las etiquetas correspondientes mediante "Título" o con un clic sobre ella.

j) Coloca un Grupo de Opción para C-Catalog.
- Etiquetas: Si y No
- Estilo: Sobre Relieve.
- Valores: Si → -1 y No → 0.
- Guardar valores en campo CCATAL.

k) Coloca dos Cuadros Combinados para:
- Localidad: Se asocia al campo LOCAL de la tabla LOCALIDADES como Origen, y al campo LOCAL de la tabla clientes para guardar los valores.
- Tipo-CL: Se escriben los valores (NORMAL, PREFER y V.I.P.), y se asocian a campo TIPOCL para guardar los valores.

l) Coloca un Cuadro de Lista para Tipo-CL escribiendo los valores como en el ejercicio (k).

m) Coloca una Línea para realizar la separación amarilla.
- Ancho de bordes: 2 pt.
- Efecto Especial: Sin relieve.

n) Coloca un Marco de Objeto Dependiente asociado al campo LOGOT.
- Distribución de la Imagen: Extender.

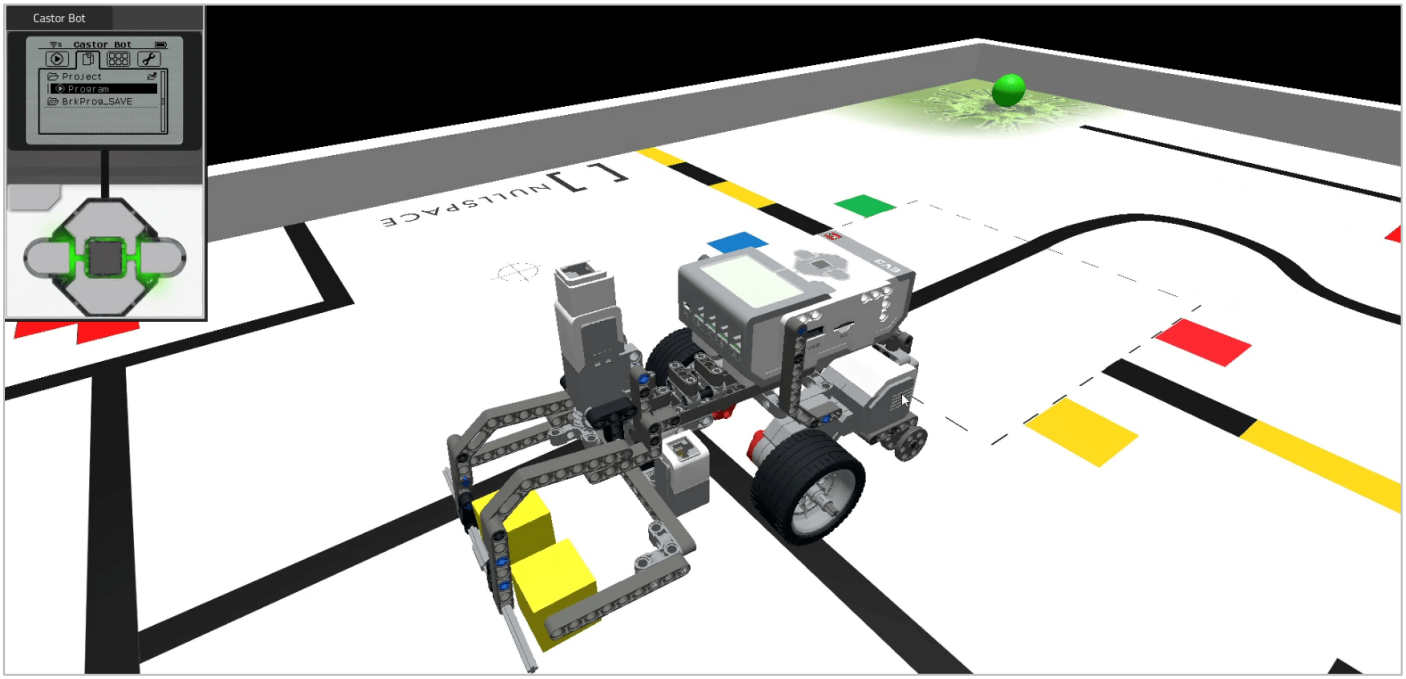
o) Coloca un Subformulario de la tabla VENTAS.

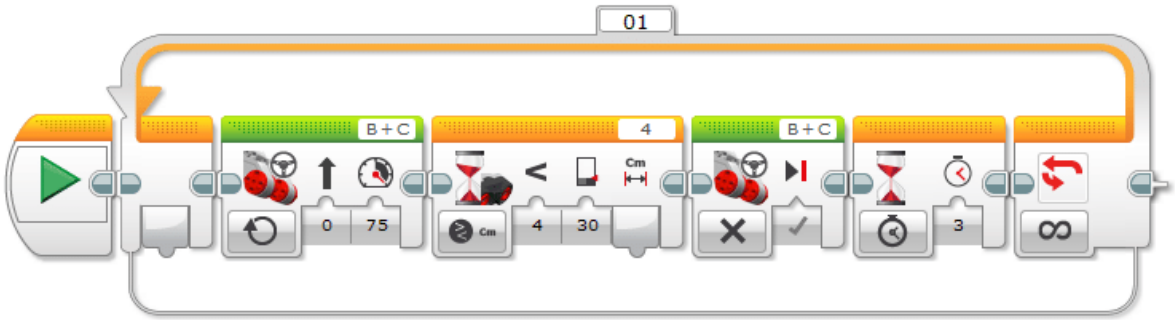
p) Crea un nuevo Formulario con nombre ARTÍCULOS y asócialo a dicha tabla.
- Colorea el Encabezado de amarillo y el Detalle de naranja pastel.
- Coloca una imagen como fondo (BS00975) estableciendo las propiedades adecuadas.
- Coloca el texto en el Encabezado y haz los cambios necesarios.

q) Coloca en el formulario ARTICULOS un Control Ficha con dos páginas:
- Pág1 → Título: Nombre. Coloca en ella los campos CODART y NOMAR.
- Pág2 → Título: Datos. Coloca el resto de campos de la tabla.

r) Vamos a colocar los cuatro Botones:
- Coloca un botón en el Formulario ARTÍCULOS para ir al Formulario CLIENTES.
- (Imagen: flecha azul) (Texto de Ayuda del Control: Volver a Formulario Clientes).
- Coloca un botón similar en el Formulario CLIENTES para ir al Formulario ARTÍCULOS.
- (Texto: Ir a Artículos (Tahoma 8, Celeste, Cursiva))
- Coloca un botón en el Formulario CLIENTES, que ejecute Excel.
- Coloca un botón en el Formulario CLIENTES, que muestre los clientes que disponen de Internet (ejecutar una consulta).

PARTE V:
ROBÓTICA.





1. PROGRAMACIÓN DEL ROBOT VIRTUAL ROBOMIND.

1.1. INTRODUCCIÓN.

Un robot es una máquina que se puede reprogramar para realizar diversas tareas. ¿Pero, qué utilidad tienen los robots? Los robots sirven para realizar trabajos aburridos o repetitivos, para efectuar tareas que pueden ser peligrosas para los seres humanos, y para ocuparse de trabajos que deben realizarse mucho más rápido de lo que cualquier persona podría hacerlo. Además, ciertas tareas se efectúan de forma mucho más eficiente y barata si las ejecuta un robot.

¿Cómo podemos indicarle a un robot la tarea que debe realizar? El funcionamiento de un robot (y de cualquier otro sistema automático) se define mediante un **programa de control**. Un programa no es más que un conjunto de instrucciones que le indican a un robot las tareas que debe realizar.

Para escribir un programa de control se utiliza un **lenguaje de programación** adaptado a la máquina a controlar. El lenguaje de programación proporciona el **juego de instrucciones** que la máquina puede entender, y define el conjunto de reglas, símbolos, y normas de sintaxis que se deben obedecer. Desarrollar un programa de control consiste en escribir una secuencia de instrucciones que le indiquen a la máquina la tarea a realizar, respetando las normas y reglas que el lenguaje de programación impone.

A modo de ejemplo inventado, imaginar que queremos programar una barrera de aparcamiento. Suponer que el juego de instrucciones de la barrera es el siguiente:

- `leerSensorPresencia()`
- `subirBarrera()`
- `esperar()`
- `bajarBarrera()`

Un ejemplo de programa de control para la barrera sería el siguiente:

```
por siempre
{
      si (leerSensorPresencia() == "coche detectado")
      {
            subirBarrera()
            esperar(leerSensorPresencia() == "coche no detectado")
            bajarBarrera()
      }
}
```

Existen multitud de lenguajes de programación que se utilizan para diferentes aplicaciones. Algunos ejemplos son C, C++, Java, Python, LOGO, etc. Los distintos lenguajes de programación son muy parecidos, ya que utilizan similares metodologías y estructuras de programación. Aquí vamos a aprender a utilizar **Robomind**, un lenguaje muy sencillo pensado para introducir a los principiantes en el mundo de la programación y la robótica. Con Robomind podremos programar con nuestro ordenador el comportamiento de un robot móvil virtual para que sea capaz, por ejemplo, de moverse evitando obstáculos, de encontrar objetos, de seguir una línea negra, o de salir de un laberinto.

1.2. ¿QUÉ ES ROBOMIND?

EL ROBOT VIRTUAL ROBOMIND.

La figura 1.1 muestra el robot virtual Robomind que vamos a programar. Como todo robot, dispone de una serie de sensores y actuadores. Los **sensores** son las videocámaras que le permiten "ver", y que utiliza como sensores de presencia y sensores de color. Los **actuadores** incluyen varios motores y ruedas para moverse, un brazo que le permite recoger unos objetos llamados balizas, y una brocha con la que puede pintar en blanco y negro. Nosotros solo escribiremos el programa de control que define su comportamiento. Este programa de control leerá la información que los sensores capten del entorno, y en base a ella, activará los actuadores adecuados para hacer que el robot cumpla su función.

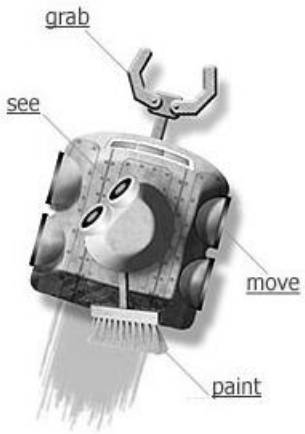


Figura 1.1. El robot virtual Robomind.

EL ENTORNO DE PROGRAMACIÓN ROBOMIND.

La figura 1.2 muestra la ventana principal de Robomind, programa que podemos descargar desde la página web https://www.robomind.net/en/download.html.

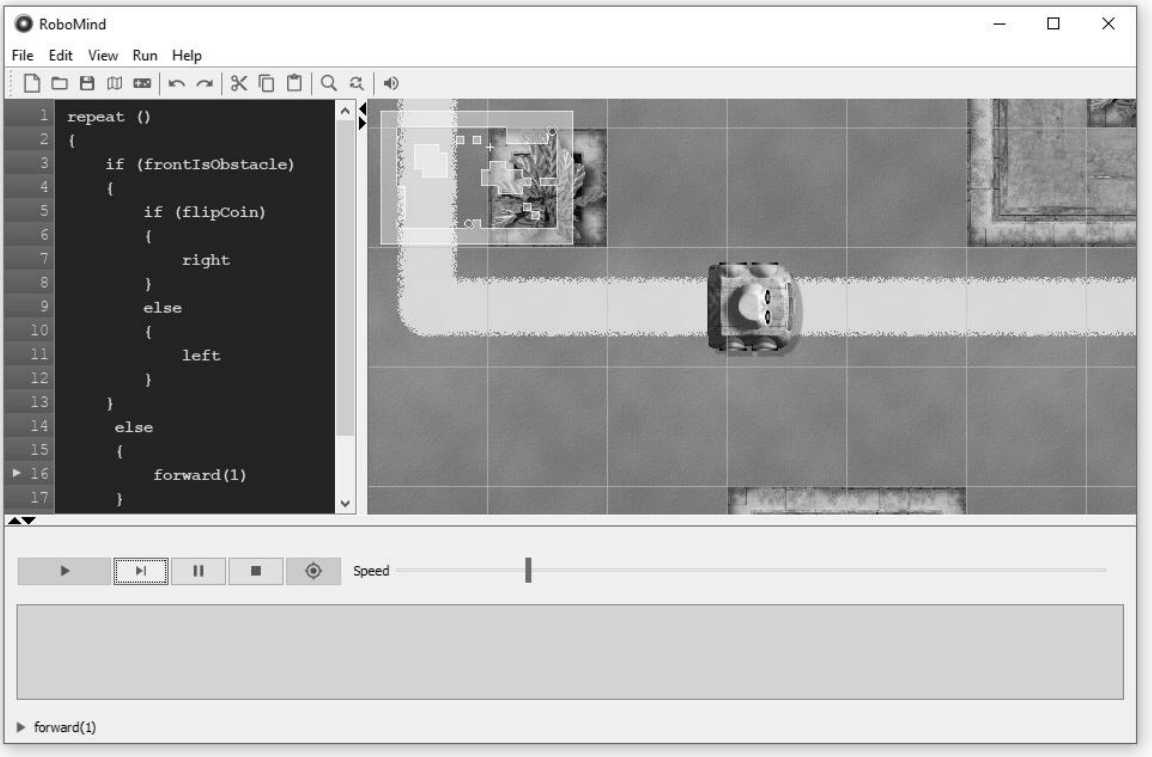


Figura 1.2. El entorno de programación de Robomind.

Las zonas más importantes de la ventana principal de Robomind son las siguientes:

- **Panel de código**: El panel oscuro a la izquierda es el editor donde escribimos nuestros programas de control.
- **Panel del entorno**: El panel a la derecha del editor de código muestra el entorno donde el robot ejecuta su programa de control. Robomind dispone de todo un conjunto de entornos preconstruidos para comprobar el correcto funcionamiento de nuestros programas en distintos ambientes. (También podemos crear nuestros propios entornos en el editor de mapas integrado, o incluso generarlos aleatoriamente). Por cierto, que a los entornos en los que vive el robot se les denomina **mapas**.
- **Botones**: Tras haber escrito el programa de control, acudimos a estos botones para lanzar la ejecución del programa, detenerla, realizar una ejecución paso a paso, acelerar o ralentizar la velocidad con la que el robot ejecuta el programa, resetear el entorno, etc.
- **Panel de mensajes**: El panel en la parte baja de la ventana es donde el compilador[20] nos informa de posibles errores de sintaxis en el programa escrito, y donde el robot nos avisa de eventuales problemas o incidencias en tiempo de ejecución.

En la zona superior de la ventana también disponemos del típico menú que nos permite abrir, guardar, o crear nuevos programas, cargar mapas, modificar los ajustes, acercarnos o alejarnos del robot, etc.

La secuencia de trabajo en Robomind es la siguiente:

1) En primer lugar, escribimos el programa de control en el panel de código.
2) A continuación, pulsamos en el botón de ejecución. Si el programa contiene errores, el compilador nos lo indicará en el panel de mensajes.
3) Si el programa es sintácticamente correcto, el robot ejecuta el programa de control en el mapa que hayamos cargado. El robot nos informará de cualquier evento relevante en tiempo de ejecución a través del panel de mensajes.

Los mensajes que lanza el robot en tiempo de ejecución son muy importantes, y no debemos ignorarlos. Aquí es donde el robot nos informa de si se ha chocado contra una pared, de si está intentando recoger una baliza donde no la hay, o de si está tratando sacar el brazo contra una pared, por poner unos pocos ejemplos. Un programa correcto nunca produce ningún tipo de mensaje en tiempo de ejecución.

A modo de ejemplo, la figura 1.3 muestra un programa de control que hace que el robot se mueva siguiendo un cuadrado. Escribe el programa en el panel de código, ejecútalo, y observa el resultado.

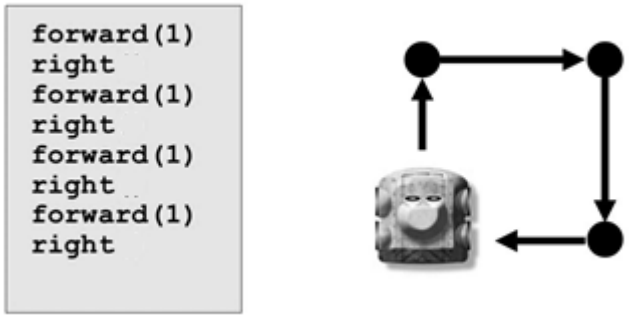


Figura 1.3. Nuestro primer programa en Robomind.

[20] El **compilador** es el software que traduce el programa escrito en un lenguaje de programación de alto nivel a código máquina (ceros y unos).

Este programa está bien escrito, no genera errores de compilación ni mensajes en tiempo de ejecución, y realiza la tarea correctamente. Pero vamos a cambiarlo ligeramente para ver qué ocurre:

a) Comenzamos escribiendo incorrectamente la primera instrucción, por ejemplo:

```
forwar(1)
```

Como la instrucción no está bien escrita el compilador no la entiende y lanza un error de sintaxis, nos informa del tipo de error, y nos indica (aproximadamente) la línea donde se produjo el error.

b) Ahora corregimos la primera instrucción, y la cambiamos para escribirla sin el último paréntesis:

```
forward(1
```

Vemos que el compilador lanza un nuevo error de sintaxis. Cuidado: Cuando nos olvidamos de escribir un paréntesis en una instrucción que la necesita, o una llave en una estructura de programación, la localización de la línea donde se produjo el error se vuelve muy imprecisa.

c) Para terminar, volvemos a escribir correctamente la primera instrucción, y cambiamos el 1 por un 7 para que el robot comience avanzando 7 casillas antes de girar:

```
forward(7)
```

A pesar del cambio el programa es sintácticamente correcto y se ejecuta, pero ocurren dos cosas: (1) Aunque el programa no contiene errores, está mal, ya que no hace la tarea planteada inicialmente (mover al robot siguiendo un cuadrado). (2) Además, el robot colisiona contra una pared, hecho del que nos informa a través del panel de mensajes (Robot: "I can't drive through the wall"). Este es uno de los múltiples mensajes a los que siempre debemos prestar atención, porque un programa que haga que el robot colisione contra un obstáculo no es correcto.

1.3. EL LENGUAJE ROBOMIND (1): JUEGO DE INSTRUCCIONES.

Como todo lenguaje de programación, el lenguaje de Robomind se caracteriza por el juego de instrucciones con las que podemos controlar al robot, por las estructuras de programación de las que disponemos para controlar el flujo de ejecución, y por las reglas sintácticas a obedecer para escribir el programa correctamente.

Presentar de golpe todas las instrucciones, estructuras, y reglas sería abrumador, incluso a pesar de la simplicidad del lenguaje. Por esta razón, vamos a ir presentándolas y poniéndolas en práctica gradualmente.

COMENTARIOS.

Todo texto que aparezca después de una almohadilla (#) no se interpreta como instrucción y no se ejecuta. El flujo de ejecución continúa con la siguiente línea del programa de control. Los comentarios se utilizan para hacer observaciones o anotaciones en el programa que expliquen ciertas partes del código.

INSTRUCCIONES DE MOVIMIENTO.

El juego de instrucciones básicas de Robomind incluye comandos para controlar los sensores (cámaras de visión) y los actuadores (ruedas, brazo, y brochas) del robot. Aquí comenzamos presentando las instrucciones que nos permiten controlar el movimiento del motor.

La tabla 1.1 muestra las instrucciones de movimiento del robot, junto con el botón asociado en el mando de control remoto (accesible en el menú Run >> Remote control).

INSTRUCCIONES DE MOVIMIENTO.		
`forward(n)`		El robot avanza n pasos en la dirección en la que está mirando.
`backward(n)`		El robot retrocede n pasos respecto a la dirección en la que está mirando.
`left(m)`		El robot gira 90º hacia la izquierda m veces sobre la casilla en la que se encuentra.
`right(m)`		El robot gira 90º hacia la derecha m veces sobre la casilla en la que se encuentra.
`north(n)`		El robot se orienta hacia el norte y avanza n pasos.
`south(n)`		El robot se orienta hacia el sur y avanza n pasos.
`east(n)`		El robot se orienta hacia el este y avanza n pasos.
`west(n)`		El robot se orienta hacia el oeste y avanza n pasos.

Tabla 1.1. Instrucciones de movimiento.

De estas 8 instrucciones de movimiento, nosotros solo usaremos las 4 primeras (`forward(n)`, `backward(n)`, `left(m)`, y `right(m)`), porque con ellas podemos efectuar cualquier movimiento que necesite hacer el robot. A modo de ejemplo, podemos escribir y ejecutar las siguientes instrucciones de forma aislada:

```
forward(1)        # El robot avanza 1 paso en la dirección en la que está
                  # mirando.
```

Para hacer que el robot avance un solo paso, también podemos escribir `forward()`, o simplemente, `forward`. Por lo tanto, `forward(1) = forward() = forward`.

```
forward(3)        # El robot avanza 3 pasos en la dirección en la que está
                  # mirando.
```

Notar que el robot solo puede avanzar o retroceder un número entero de pasos. Por ejemplo, la instrucción `forward(4.5)` generará un error sintáctico.

```
right(1)          # El robot gira 90° hacia la derecha sobre la casilla en la que
                  # se encuentra.
```

Para hacer que el robot gire 90º una sola vez también podemos escribir `right()`, o simplemente, `right`. Por lo tanto, `right(1) = right() = right`.

```
right(2)          # El robot gira 90° hacia la derecha dos veces. Esto equivale a
                  # realizar un giro de 2 × 90° = 180° (media vuelta).

right(4)          # El robot gira 90° hacia la derecha cuatro veces. Esto
                  # equivale a realizar un giro de 4 × 90° = 360°
                  # (una vuelta completa).
```

Para terminar, vamos a combinar varias instrucciones de movimiento en un solo programa:

```
# map: default.map

forward(3)
right
forward(2)
left
backward(3)
right(2)
forward
```

Como ya sabemos, la primera línea de código es un comentario. Sin embargo, este comentario en particular no es una anotación, sino una orden especial que nos permite cargar el mapa indicado tras la sentencia `map:` (en este caso, el mapa `default.map`). Ahora, con la primera instrucción el robot avanza 3 pasos hacia adelante en la dirección en la que está mirando (hacia el norte). A continuación, el robot gira 90º hacia la derecha en la posición en la que está, y pasa a mirar hacia el este. Luego avanza 2 pasos hacia el este. Después gira 90º hacia la izquierda y pasa a mirar al norte otra vez. Seguidamente, el robot retrocede 3 pasos (esto es, se mueve marcha atrás hacia el sur mientras sigue mirando hacia el norte). Posteriormente, el robot gira 90º hacia la derecha dos veces (esto es, gira 180º hacia la derecha y da media vuelta), y como estaba mirando hacia el norte, pasa a mirar hacia el sur. Para terminar, el robot avanza 1 paso hacia el sur, y como no quedan más instrucciones, el programa termina.

Si hemos entendido estos ejemplos estamos preparados para escribir nuestros primeros programas de control. Como comprobarás, los programas propuestos están pensados para funcionar en un solo mapa, en varios mapas, o en todos los mapas. Para tener claro en qué mapas debe funcionar cada programa, es muy útil indicar con un comentario al principio del programa los mapas donde debemos ejecutar ese programa. Además, y cuando el programa deba funcionar en un solo mapa, podemos cargar ese mapa particular usando la instrucción `# map: mapName.map`.

> NOTA: Aunque Robomind viene con algunos mapas instalados, nosotros necesitaremos usar muchos mapas más. Todos los mapas que utilizaremos están disponibles en el archivo comprimido "maps.rar", que debemos descomprimir en nuestra carpeta de trabajo.

Para cambiar del mapa actual a otro mapa distinto seleccionamos menú File >> Open map, y en la ventana de diálogo, acudimos en la carpeta donde hemos descomprimido el archivo maps.rar, y seleccionamos el mapa deseado.

> NOTA: Antes de ejecutar un programa, es recomendable activar la opción View >> Track robot para que la pantalla siga al robot en su movimiento a lo largo del mapa.

EJERCICIO 1. Línea blanca. (Mapa: default.map). Escribe un programa que lleve al robot al principio del sendero blanco, y que le obligue a recorrerlo sin chocarse contra ningún obstáculo. El robot se parará en la última casilla del sendero blanco. Guarda el programa como `prog1.irobo`.

> NOTA: Para guardar un archivo, acudimos al menú File >> Save. Se abrirá una ventana de diálogo en la que indicamos dónde guardar el archivo (elegimos nuestra carpeta de trabajo) y el nombre del archivo. Es importante recordar que, para poder escribir un nuevo programa tras haber guardado un programa previo, debemos elegir la opción File >> New.

EJERCICIO 2. Rectángulo 3×6. (Mapa: openArea.map). Programa al robot para que realice un recorrido en forma de rectángulo con una altura de 3 pasos y una anchura de 6 pasos. El robot termina de recorrer el rectángulo en su posición inicial, donde se detiene. CUIDADO: La casilla en la que aparece el robot cuenta para la altura y anchura del rectángulo a recorrer. Guarda el programa como `prog2.irobo`.

NOTA: Las nuevas versiones de Robomind incluyen la instrucción `goTo(x,y)`, que sirve para llevar al robot a la casilla con las coordenadas (x,y) indicadas. El origen de coordenadas $(0,0)$ se sitúa en la casilla en el extremo superior izquierdo del mapa, con la coordenada x incrementándose hacia la derecha, y la coordenada y hacia abajo. Para mostrar las coordenadas de cada casilla del mapa, basta con pulsar la tecla F9. A modo de prueba, podemos acudir al mapa default.map y ejecutar la instrucción `goTo(5,3)`. Es importante notar que la instrucción no funcionará si la casilla de destino está ocupada (por un muro, un obstáculo, una baliza, etc.). La instrucción `goTo` devuelve el número de casillas que el robot ha recorrido hasta llegar al destino. Para comprobarlo, podemos escribir `show(goTo(5,3))` y observar la respuesta en el panel de mensajes (Robot: "12").

INSTRUCCIONES PARA RECOGER BALIZAS Y PINTAR EN EL SUELO.

Las tablas 1.2 y 1.3 incluyen las instrucciones que le permiten al robot trazar puntos y líneas en el suelo (en blanco o negro), así como las instrucciones para recoger, soltar, y destruir **balizas** (los únicos objetos que el robot es capaz de agarrar con su brazo).

INSTRUCCIONES PARA PINTAR.		
`paintWhite`		El robot saca la brocha mojada con pintura blanca, y pinta un punto blanco en el suelo. A partir de ahora, el robot puede moverse para trazar una línea blanca detrás de él.
`paintBlack`		El robot saca la brocha mojada con pintura negra, y pinta un punto negro en el suelo. A partir de ahora, el robot puede moverse para trazar una línea negra detrás de él.
`stopPainting`		El robot guarda la brocha y deja de pintar. A partir de ahora, el robot puede moverse sin dejar un rastro de pintura.

Tabla 1.2. Instrucciones para pintar en el suelo.

INSTRUCCIONES PARA RECOGER BALIZAS.		
`pickUp`		El robot saca su brazo, recoge la baliza situada en la casilla justo delante de él, y se la guarda dentro. A partir de ahora, el robot puede moverse y transportar la baliza que se ha guardado. El robot no puede transportar más de una baliza al mismo tiempo.
`putDown`		El robot saca su brazo para soltar la baliza que transporta en la casilla situada justo delante de él.
`eatUp`		El robot saca su brazo, recoge la baliza situada en la casilla justo delante de él, y la destruye.

Tabla 1.3. Instrucciones para recoger, soltar, y destruir balizas.

A modo de ejemplo, vamos a ejecutar unos programas sencillos:

```
# Ejemplo 1: Este programa muestra cómo pintar puntos.
# map: default.map
paintBlack
stopPainting
forward(2)
right
paintWhite
stopPainting
forward(1)
```

```
# Ejemplo 2: Este programa muestra cómo trazar una línea.
# map: default.map
paintWhite
forward(2)
left
forward(3)
stopPainting
forward(1)

# Ejemplo 3: Este programa muestra cómo recoger y soltar balizas.
# map: default.map
right(2)
forward(2)
left
forward(2)
pickUp
left
forward(4)
putDown
```

A la hora de usar las instrucciones para pintar debemos tener algunas precauciones: (1) Si ejecutamos la instrucción `paintWhite` para sacar la brocha blanca, y volvemos a ejecutar la misma instrucción sin haberla guardado la brocha antes, el robot lanza un mensaje de error en tiempo de ejecución (Robot: "I already painted white"). Y lo mismo ocurre con la brocha negra (Robot: "I already painted black"). (2) Si tratamos de guardar la brocha sin haberla sacado antes, el robot da un mensaje de error (Robot: "I already stopped painting"). Y lo mismo pasa si ejecutamos la instrucción de guardar la brocha dos veces. Para terminar vale la pena indicar que los programas en los que el robot use la brocha siempre deben acabar con la brocha escondida.

También debemos estar alerta a la hora de usar las instrucciones para recoger balizas: (1) El robot solo puede recoger una baliza que esté localizada en la casilla justo delante de él. Si el robot intenta recoger una baliza donde no la hay, lanzará un mensaje de error en tiempo de ejecución (Robot: "There is no beacon to get"). (2) Si el robot intenta sacar su brazo delante de una pared o de cualquier otro obstáculo, lanzará un mensaje de error (Robot: "I can't get a beacon if there's an obstacle in front of me"). (3) El robot no puede transportar más de una baliza al mismo tiempo, a no ser que las vaya destruyendo conforme las recoge. Si intentamos hacerlo, el robot lanzará un mensaje de error (Robot: "I can only carry one beacon at a time"). (4) Si el robot lleva dentro una baliza y la queremos soltar, debemos hacerlo en una casilla que esté libre. De otra forma, el robot lanzará un mensaje de error (Robot: "I can't place the beacon over here. There's an obstacle in front of me"). (5) Si intentamos soltar una baliza sin haberla cogido antes, o si la hemos destruido, el robot lanzará un mensaje de error (Robot: "I don't have a beacon on me").

NOTA: Para escribir rápidamente las instrucciones (y no equivocarnos al escribirlas), Robomind ofrece un atajo: Escribimos las primeras dos o tres letras de la instrucción, y a continuación, pulsamos CTRL + ESPACIO. Aparecerá una ventana auxiliar que nos muestra las instrucciones que comienzan con esas letras, y en la que podemos elegir aquella que deseamos escribir (ver figura 1.4).

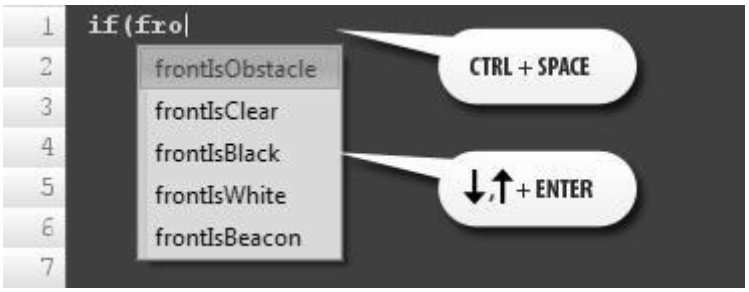


Figura 1.4. Asistente de escritura de instrucciones.

NOTA: Podemos personalizar el aspecto del robot y del entorno acudiendo al menú File » Settings » pestaña View.

Figura 1.5. Personalización del robot.

EJERCICIO 3. Transporta la baliza. (Mapa: default.map). Escribe un programa que lleve al robot a recoger la baliza de la esquina superior derecha, y que la lleve a la esquina inferior izquierda. Guarda el programa como `prog3.irobo`.

EJERCICIO 4. Atraviesa las balizas (1). (Mapa: passBeacons.map). Escribe un programa para conseguir que el robot llegue hasta el punto blanco atravesando el pasillo de balizas que se interpone ante él. NOTA: No está permitido usar la instrucción `eatUp()`. Guarda el programa como `prog4.irobo`.

EJERCICIO 5. Árbol de navidad. (Mapa: christmastree.map). Escribe un programa para que el robot lleve las balizas a los puntos negros del árbol de navidad que hay pintado en el suelo, a modo de bolas decorativas. Guarda el programa como `prog5.irobo`.

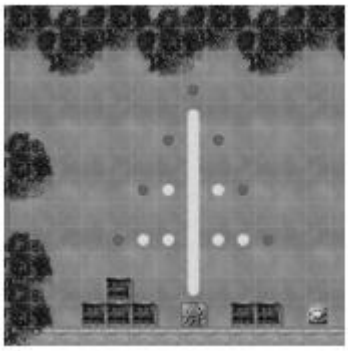


Figura 1.6. Ejercicios 5 y 6.

EJERCICIO 6. Escalera. (Mapa: openArea.map). Escribe un programa para que el robot dibuje una escalera blanca de 3 escalones. Guarda el programa como `prog6.irobo`.

EJERCICIO 7. Iniciales. (Mapa: openArea.map). Escribe un programa para que el robot escriba las iniciales de tu nombre y apellidos (3 letras) en color blanco. Entre cada inicial y tras la última, debe haber un punto de separación de color negro (por ejemplo, A.M.P.). Guarda el programa como `prog7.irobo`.

1.4. EL LENGUAJE ROBOMIND (2): CONTROL DE FLUJO.

INSTRUCCIONES DE VISIÓN Y DE ALEATORIEDAD.

Las instrucciones de visión controlan las videocámaras con las que el robot puede "ver". Las videocámaras actúan como sensores que le permiten detectar la presencia de obstáculos (paredes, cajas, etc.), detectar balizas, y distinguir colores pintados en el suelo (blanco y negro). La tabla 1.4 muestra las instrucciones de visión.Por su parte, la instrucción de aleatoriedad le permite al robot lanzar una moneda al aire para tomar una decisión aleatoria (como por ejemplo, girar hacia la derecha o a la izquierda, pintar de negro o de

blanco, etc.). Esta función devuelve verdadero (True) o falso (False) con una probabilidad del 50%−50%. La tabla 1.5 explica esta instrucción de aleatoriedad.

INSTRUCCIONES DE VISIÓN.		
IZQUIERDA	EN FRENTE	DERECHA
`leftIsObstacle`	`frontIsObstacle`	`rightIsObstacle`
`leftIsClear`	`frontIsClear`	`rightIsClear`
`leftIsBeacon`	`frontIsBeacon`	`rigthIsBeacon`
`leftIsWhite`	`frontIsWhite`	`rightIsWhite`
`leftIsBlack`	`fontIsBlack`	`rightIsBlack`

Tabla 1.4. Instrucciones de visión

TOMA DE DECISIONES ALEATORIAS.	
`flipCoin`	El robot lanza una moneda al aire para tomar una decisión aleatoria. El resultado puede ser cara (TRUE) o cruz (FALSE) con una probabilidad del 50%−50%.

Tabla 1.5. Instrucción para la toma de decisiones aleatorias.

Para entender cómo funcionan las instrucciones de visión, veamos algunos ejemplos: (1) La instrucción `leftIsObstacle` le permite al robot mirar la casilla que está justo a su izquierda, y determinar si en esa casilla hay un obstáculo contra el que pueda colisionar, como una pared, una caja, una planta, una baliza, etc. (Las líneas o puntos pintados en el suelo no son obstáculos, ya que el robot puede pasar por encima de ellos). (2) La instrucción `frontIsClear` le permite al robot mirar la casilla justo al frente y determinar si esa casilla está despejada de obstáculos. (3) La instrucción `rightIsWhite` le permite al robot mirar a la casilla justo a su derecha y determinar si está pintada de blanco. (4) La instrucción `leftIsBeacon` le permite al robot mirar la casilla de su izquierda y determinar si hay una baliza en ella (recordemos que las balizas son los únicos objetos que el robot puede recoger). Notar que el robot solo es capaz de "ver" lo que hay en las casillas justo a su izquierda, justo en frente, y justo a su derecha; el robot no puede ver detrás de él, ni a dos casillas de distancia[21], ni en diagonal.

Una observación importante respecto a las instrucciones de visión y de aleatoriedad: Si en un programa usamos estas instrucciones por sí solas, veremos que no sirven para nada. Pongamos como ejemplo el programa escrito más abajo. Si ejecutamos este programa, veremos que el robot mueve la cabeza a la izquierda para observar si la casilla está despejada, y después mueve la cabeza a la derecha para ver si esa casilla está pintada de negro, pero al margen de ello, no ocurre nada más. El robot también lanza una moneda al aire y obtendrá un resultado (cara o cruz), pero no hace nada más. Como veremos a continuación, estas instrucciones no se usan solas, sino como condiciones de las estructuras de control de flujo (bucles y condicionales).

```
leftIsClear
rightIsBlack
flipCoin
```

BUCLES.

Hasta ahora, los programas que hemos escrito se ejecutaban secuencialmente, esto es, instrucción tras instrucción desde la primera hasta la última línea de código. Estos programas eran muy poco flexibles, y

[21] Algunas instrucciones de visión no cumplen exactamente con esta regla, como veremos más adelante.

solo funcionaban para un mapa en particular. (Por ejemplo, el ejercicio 4 solo funciona para el mapa christmastree.map. Si lo ejecutamos en cualquier otro mapa, parecerá que el robot hace cosas raras). La mayoría de veces querremos que el programa funcione correctamente con independencia del mapa en el que lo ejecutemos. Para ello, necesitamos usar estructuras de control de flujo. Las **estructuras de control de flujo** (bucles y condicionales) permiten definir comportamientos no secuenciales, como repetir la ejecución de un cierto conjunto de instrucciones, o ejecutar unas instrucciones u otras dependiendo de si se cumple una condición. En esta sección comenzamos con los **bucles**, que son estructuras que permiten repetir la ejecución de las instrucciones que llevan entre llaves. En Robomind tenemos tres tipos de bucles:

Bucle estándar.

```
repeat(n)
{
    … instrucciones …
}
```

Este bucle nos permite repetir las instrucciones entre llaves un número n de veces que es conocido a priori. Después de repetir n veces las instrucciones del bucle, el programa continúa con las instrucciones que haya tras el bucle. A modo de ejemplo, tenemos el siguiente programa:

```
# Dibuja un cuadrado.

paintWhite
repeat(4)
{
      forward(2)
      right
}
stopPainting
```

Notar que este programa es una forma más eficiente y compacta de escribir el programa de la figura 1.3. Recorrer una trayectoria cuadrada implica repetir cuatro veces la acción de avanzar y girar 90º a la derecha.

Bucle infinito.

```
repeat
{
    … instrucciones …
}
```

Cuando escribimos el bucle `repeat` sin número de repeticiones ni paréntesis, tenemos un **bucle infinito**: El programa estará ejecutando las instrucciones entre llaves para siempre. Notar que en un bucle infinito nunca se ejecutarán las instrucciones que vienen tras el bucle, a no ser que añadamos una sentencia `break` (de la que hablaremos más adelente). Un ejemplo es el siguiente programa, que hace que el robot avance y retroceda una posición para siempre.

```
repeat
{
      forward(1)
      backward(1)
}
```

Bucle condicional.

```
repeatWhile (condición)
{
    … instrucciones …
}
```

Este tipo de estructura se denomina **bucle condicional**, y repite las instrucciones entre las llaves mientras se cumpla la condición indicada entre paréntesis. (Esta condición siempre es una instrucción de visión, como por ejemplo, `frontIsClear`). Cuando la condición deja de cumplirse, el programa sale del bucle y continúa ejecutando las instrucciones que le siguen.

Un ejemplo de programa es el siguiente:

```
repeatWhile(frontIsClear)
{
     forward(1)
}
```

Este programa hace que el robot avance paso a paso mientras la casilla justo al frente esté despejada de obstáculos. Cuando detecta un obstáculo, la condición ya no se verifica y el programa sale del bucle. Este programa le permite al robot avanzar hasta encontrar un obstáculo, momento en el que se detiene sin colisionar contra él. (Notar que este programa no podríamos hacerlo con un bucle infinito, porque al final terminaría chocando contra un obstáculo).

> NOTA: Como veremos, al utilizar las estructuras de control de flujo el código se vuelve más complejo, y se hace difícil de leer y entender. Afortunadamente, la combinación de teclas CTRL + ALT + F nos permite dar formato automático al código del programa para que quede más limpio.

EJERCICIO 8. Escalera con bucle. (Mapa: openArea.map). Como en el ejercicio 6, escribe un programa que haga que el robot pinte una escalera de 3 escalones, pero esta vez, utilizando un bucle. Guarda el programa como `prog8.irobo`.

EJERCICIO 9. Rodea la caja. (Mapa: goRightAtWhite1.map). Escribe un programa que lleve al robot a la caja aislada situada arriba y a la izquierda. Una vez allí, el robot empieza a rodear la caja indefinidamente mientras pinta su contorno de color negro. Guarda el programa como `prog9.irobo`.

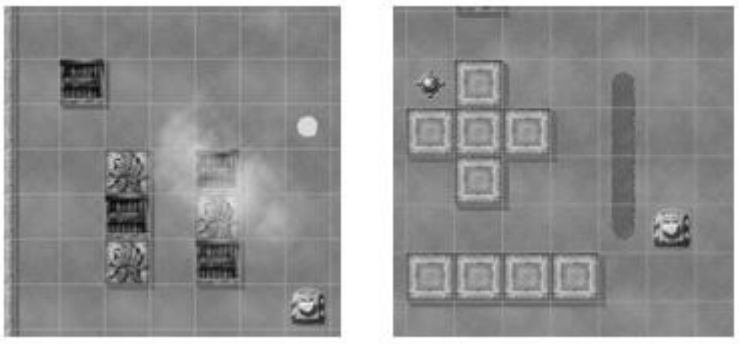


Figura 1.7. Ejercicios 9 y 10.

EJERCICIO 10. Paralelo a la línea. (Mapas: copyLine1.map y copyLine2.map). Crea un programa que haga que el robot camine paralelo a la línea negra que tiene a su izquierda. El robot debe avanzar paralelo mientras haya línea negra. Asegúrate de que el mismo programa funciona igual de bien para ambos mapas. Guarda el programa como `prog10.irobo`.

EJERCICIO 11. Trazando el camino. (Mapas: Cualquier mapa donde haya al menos dos casilla de espacio libre para avanzar). Escribe un programa en el que el robot avance indefinidamente sin chocarse. Mientras avanza, el robot va pintando el suelo de blanco. Al llegar a un obstáculo y no poder avanzar más, el robot para, pinta un punto negro justo en la casilla frente al obstáculo, y a continuación, se aparta para dejarnos ver el patrón que ha dibujado. (CUIDADO: Al apartarse, el robot no debería chocarse contra ningún obstáculo). El programa termina en este punto. Guarda el programa como `prog11.irobo`.

NOTA: Una herramienta muy útil para detectar problemas de funcionamiento en nuestros programas es la **ejecución paso a paso**. El botón de ejecución paso a paso se localiza junto al botón de ejecución del programa. Pero al contrario que el botón de ejecución, que ejecuta el programa completo, el botón de ejecución paso a paso sólo ejecuta la siguiente instrucción del programa. Pulsando continuamente el botón de ejecución paso a paso podemos ejecutar nuestro programa instrucción a instrucción, lo que nos permitirá localizar en qué parte del código se encuentra el problema de funcionamiento.

Figura 1.8. Ejecución paso a paso.

ESTRUCTURAS CONDICIONALES.

Condicional simple.

La forma más sencilla de estructura condicional es la siguiente:

```
if (condición)
{
    … instrucciones …
}
```

Si la condición se cumple, el robot ejecuta las instrucciones que van entre llaves, y si no se cumple, el robot se las salta. En ambos casos, el robot continúa ejecutando el código que haya tras el condicional. Como en el caso de los bucles condicionales, la condición que controla esta estructura condicional es una instrucción de visión. Notar que un condicional *no* es un bucle: Si se cumple la condición, las instrucciones se ejecutan una sola vez (o no se ejecutan si la condición no se cumple). A modo de ejemplo, consideremos el siguiente programa:

```
# El robot avanza mientras no haya obstáculos. Si mientras avanza ve una
# baliza al frente, la destruye.

repeatWhile(frontIsClear)
{
      forward(1)
      if (frontIsBeacon)
      {
            eatUp
      }
}
```

Bifurcación.

Otra forma de estructura condicional es la siguiente:

```
if (condición)
{
      instrucciones
}
```

```
else
{
      instrucciones
}
```

Si la condición se cumple, se ejecutan las instrucciones dentro del bloque `if`; si no se cumple, se ejecutan las instrucciones dentro del bloque `else`. Notar que este condicional es distinto al anterior. El primero ejecuta el código del `if`, o no ejecuta nada. El segundo siempre ejecuta algo: El código del `if`, o el código del `else` (por esta razón se le llama **bifurcación**).

La condición que gobierna la bifurcación puede ser una instrucción de visión (por ejemplo, `frontIsClear`), o una instrucción de aleatoriedad (`flipCoin`). Un ejemplo de uso es el siguiente:

```
# El robot entra en un bucle infinito donde mira constantemente si al frente
# está despejado. Si lo está avanza un paso, y si no, gira a la derecha para
# evitar el obstáculo.

repeat
{
      if (frontIsClear)
      {
            forward(1)
      }
      else
      {
            right
      }
}
```

LA INSTRUCCIÓN BREAK.

La instrucción `break` nos permite forzar la salida de un bucle. Al salir, el robot continúa ejecutando las instrucciones que vienen tras el bucle.

Esta instrucción suele usarse dentro de una estructura condicional que controle la forma en la que salimos del bucle. A continuación, mostramos un ejemplo:

```
# El robot avanza mientras no haya obstáculos. En cuanto encuentra una línea
# blanca, fuerza la salida del primer bucle y entra en el segundo, donde la va
# pintando de negro hasta que la línea termina.

repeatWhile (frontIsClear)
{
      forward(1)
      if (frontIsWhite)
      {
            break
      }
}

repeatWhile (frontIsWhite)
      {
            forward(1)
            paintBlack
      }
stopPainting
```

LA INSTRUCCIÓN END.

La instrucción `end` fuerza la finalización del programa, incluso aunque tras ella queden instrucciones por ejecutar. Notar la diferencia de la instrucción `end` con la instrucción `break`: Un `break` solo fuerza la salida del bucle en el que está, pero el programa sigue ejecutándose; por el contrario, un `end` finaliza el programa por completo.

De nuevo, esta instrucción suele usarse dentro de una estructura condicional que controle la forma en la que finaliza el programa.

A continuación mostramos un programa de ejemplo:

```
# Mientras no haya obstáculos, el robot avanza paso a paso pintando de negro.
# Si en su avance ve un punto blanco, se sube encima y termina el programa.

paintBlack
repeatWhile(frontIsClear)
{
      forward(1)
      if(frontIsWhite)
      {
            stopPainting
            forward(1)
            end
      }
}
```

EJERCICIO 12. Robot borracho. (Cualquier mapa). Escribe un programa en el que el robot avance paso a paso indefinidamente, mientras a cada paso cambia de dirección de forma aleatoria. Para cambiar de dirección aleatoriamente, el robot debe usar la instrucción `flipCoin` para decidir al azar si gira a la derecha (`right`) o a la izquierda (`left`).

La función `flipCoin` siempre se utiliza como condición de una estructura condicional, de la siguiente forma:

```
if (flipCoin)         # Si sale cara …
{
      instrucciones   # … haz esto.
}
else                  # En caso contrario (si sale cruz) …
{
      instrucciones   # … haz esto otro.
}
```

Este ejercicio es el único programa donde admitiremos que el robot pueda chocarse contra las paredes (al fin y al cabo, es un robot borracho). Guarda el programa como `prog12.irobo`.

EJERCICIO 13. Blanco y negro. (Cualquier mapa). Escribe un programa en el que el robot avance hacia adelante paso a paso sin chocarse. A cada paso, el robot pinta aleatoriamente un punto blanco o un punto negro. Al llegar al primer obstáculo, el robot se para. Guarda el programa como `prog13.irobo`.

EJERCICIO 14. Robot copión. (Mapas: copyLine1.map y copyLine2.map). En los mapas indicados tenemos una línea negra a la izquierda del robot (ver figura 1.7). El objetivo de este programa es copiarla, dibujando una línea blanca *de igual longitud* a la derecha de la línea negra. Haz el programa de forma que se ejecute correctamente incluso si no conocemos a priori el tamaño de la línea negra. Guarda el programa como `prog14(a).irobo`. AMPLIACIÓN: Modifica el programa para que funcione incluso cuando la línea pueda

ser blanca o negra. El robot debe copiarla, pero con el color contrario (si la línea es blanca la copia negra, y viceversa). Este programa debe funcionar para los mapas copyLine1.map, copyLine2.map, y copyLine3.map. Guarda el programa como `prog14(b).irobo`.

EJERCICIO 15. Aparcando. (Mapas: Todos los mapas findSpot.map). Programa al robot para que se mueva pegado a la pared y "aparque" en el hueco marcado con una señal blanca. Guarda el programa como `prog15.irobo`.

EJERCICIO 16. Atraviesa las balizas (2). (Mapas: Todos los mapas passBeacons.map). Vuelve a realizar el programa de atravesar las balizas, pero esta vez, debes generalizar la solución para que el mismo programa funcione bien independientemente del número de balizas que haya en el pasillo, de que haya huecos entre las balizas, y de la posición final del punto blanco donde el robot debe deternerse. Guarda el programa como `prog16.irobo`.

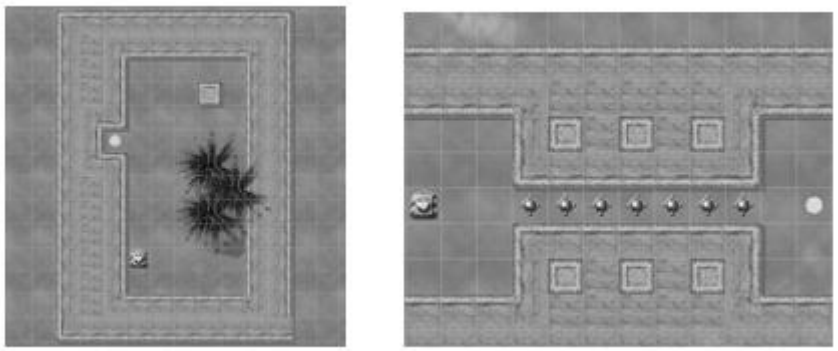


Figura 1.9. Ejercicios 15 y 16.

EJERCICIO 17. Esquivar objetos aislados. (Mapas: todos los mapas avoidObstacles.map hasta el número 3). Programa al robot para que avance paso a paso hacia adelante a lo largo de un camino vertical. Cuando el robot se encuentra con un obstáculo, lo evita y vuelve inmediatamente al camino vertical. El objetivo es llevar al robot al punto blanco al final del camino, encima del cual se detiene. Guarda el programa como `prog17(a).irobo`. AMPLIACIÓN 1: En el mapa avoidObstacles5.map hay un punto negro al final del recorrido, y un punto blanco al principio. Programa al robot para llegar al punto negro igual que antes, pero una vez allí, en vez de pararse, da la vuelta y retorna hasta llegar al punto blando, donde el robot se detiene y el programa termina. OJO: El mismo programa debe funcionar para este nuevo mapa, pero también para todos los mapas previos, excepto el 4. Guarda el programa como `prog17(b).irobo`. AMPLIACIÓN 2: Modifica tu programa para que funcione también en el mapa avoidObstacles4.map. En este mapa hay un punto negro al final, pero no hay un punto blanco al que regresar. Haz que el robot dé la vuelta en el punto negro, y si no encuentra punto blanco, que se detenga cuando ya no pueda avanzar más. Este nuevo programa debería funcionar correctamente para todos los mapas avoidObstacles.map. Guarda el programa como `prog17(c).irobo`.

EJERCICIO 18. Esquivar objetos continuados. (Mapas: todos los mapas avoidContinuousObstacles.map). En el programa previo evitar los obstáculos era sencillo, porque todos tenían una longitud de un bloque. En este caso vamos a reescribir el programa para que el robot sea capaz de esquivar objetos de longitud desconocida (esto es, de uno, dos, tres bloques, etc.). Para conseguirlo, deberás colocarte al lado del obstáculo, a la altura del primer bloque, y avanzar usando la visión lateral para ver cuándo finaliza el obstáculo. Cuando finaliza el obstáculo, el robot debe volver al camino vertical que le lleva al punto blanco. Guarda el programa como `prog18.irobo`.

EJERCICIO 19. Robot busca balizas. (Mapa: Cualquiera que tenga al menos una baliza). Escribe un programa para que el robot avance por el mapa de forma autónoma, sin colisionar contra ningún obstáculo. Mientras recorre el mapa, el robot debe estar alerta en busca de balizas. En cuanto el robot encuentre la primera baliza, la recoge y el programa termina. Prueba el programa en todos los mapas donde haya al

menos una baliza, pero especialmente, en los mapas openArea2baliza.map y rm-lf-task1.map. Guarda el programa como `prog19.irobo`. PISTA: Tal vez pienses que, para mover al robot, bastará con hacer que el robot avance paso a paso, cambiando de dirección aleatoriamente cada vez que encuentre un obstáculo. Esta técnica de búsqueda funciona en algunos mapas, pero no en todos (por ejemplo, no funciona en los dos mapas indicados). Para asegurarnos de que el robot recorre todas y cada una de las casillas del mapa buscando balizas, el mejor método es cambiar de dirección aleatoriamente tras cada paso, incluso aunque el robot no se haya encontrado con un obstáculo.

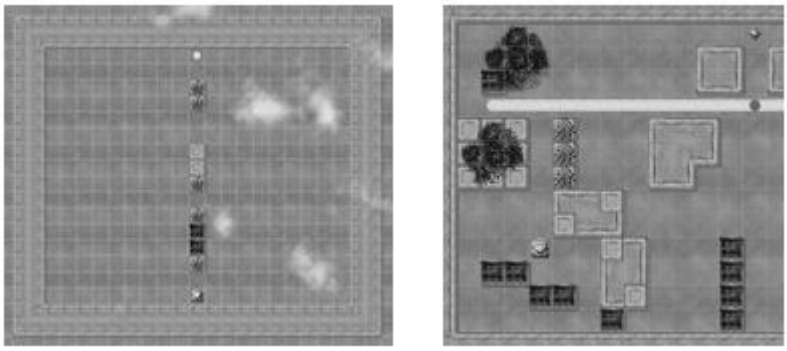


Figura 1.10. Ejercicios 18 y 19.

NOTA: Además de probar nuestros programas en los mapas indicados, podemos probarlos en mapas similares. Para ello, Robomind nos permite crear nuestros propios mapas usando el editor de mapas integrado (File ›› Map editor). Sin embargo, una opción más sencilla es usar la herramienta de generación de mapas aleatorios (File ›› Generate map). Esta herramienta sirve para crear mapas aleatorios eligiendo el tipo de mapa (forest, dungeon, island, cave, maze, openArea, villa, y text), y el tamaño deseado (ver figura 1.11). Además, podemos hacer que el propio programa de control genere un mapa aleatorio mediante código. Por ejemplo, la instrucción `# map: maze(10,10)` creará un mapa aleatorio de tipo laberinto, y de tamaño 10×10 casillas.

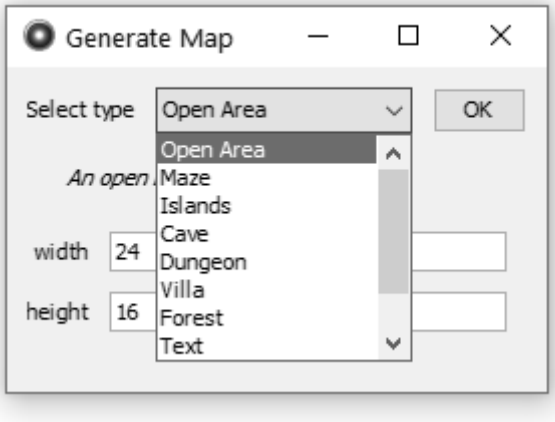


Figura 1.11. Generación de mapas aleatorios.

1.5. EL LENGUAJE ROBOMIND (3): VARIABLES.

ARITMÉTICA.

Los operadores aritméticos (`+`, `-`, `*`, `/`) le permiten al robot hacer las operaciones de suma, resta, multiplicación, y división. Los números a operar deben ser números enteros. Si el resultado de una operación no es un entero, Robomind redondea al entero más bajo. A continuación tenemos un par de ejemplos de uso de los operadores aritméticos:

```
backward(2+3)          # El robot retrocede 5 pasos.
forward(-(3+2)/2)      # El robot avanza -2 pasos (esto es, retrocede 2 pasos).
```

VARIABLES.

Las variables nos permiten recordar un cierto valor bajo un nombre que nosotros podemos elegir. Los valores almacenados en una variable pueden referenciarse usando el nombre de esa variable (ver la segunda y tercera instrucciones en el código de ejemplo). En las variables podemos guardar valores numéricos (línea 1 del ejemplo), el resultado de una expresión aritmética o lógica (líneas 2 y 4 del ejemplo, respectivamente), o el valor de retorno de las instrucciones de visión y de los procedimientos (como en la línea 5).[22]

```
num = 3
doble = 2*num
forward(doble)
boolVar = (leftIsClear or frontIsWhite) and not rightIsBeacon
casillasBlancasDelante = frontIsWhite
```

Por cierto, que nos sorprenderá saber que ciertas instrucciones básicas de Robomind no solo realizan la acción para la que están creadas, sino que también "devuelven" algunos valores útiles. Por ejemplo, las instrucciones `frontIsClear`, `leftIsClear`, y `rightIsClear` no solo nos dicen si la casilla indicada está libre de obstáculos, sino también cuántas casillas están libres en esa dirección. Así mismo, las instrucciones `frontIsWhite`, `frontIsBlack`, etc. no solo nos dicen si el suelo está pintado, sino también cuántas casillas en esa dirección están pintadas de ese color. En general, todos los comandos básicos (`forward(n)`, `pickUp`, etc.) devuelven un valor de "éxito o fracaso". Por ejemplo la función `forward(n)` no solo mueve el robot hacia adelante; también devuelve el número real de pasos dados hasta que el robot se choca. Y la función `pickUp` nos indica si consiguió recoger una baliza o no. Todos estos valores de retorno son muy útiles, y los podemos guardar y rescatar utilizando variables (ver ejemplo a continuación).

```
# La instrucción frontIsClear devuelve el número de casillas libres de
# obstáculos justo delante del robot. En este ejemplo, guardamos este dato en
# la variable 'casillasLibres', y a continuación, avanzamos hacia adelante
# ese número de pasos.

casillasLibres = frontIsClear
forward(casillasLibres)
```

EJERCICIO 20. Distancias a los obstáculos. (Mapas: Todos los mapas). Escribe un programa en el que el robot trace las distancias hasta los ostáculos, como se indica: Desde su posición inicial, el robot se mueve hacia adelante trazando una línea sobre el suelo de color aleatorio (blanco o negro) hasta el primer obstáculo que encuentre. A continuación, cambia de dirección aleatoriamente, y vuelve a moverse hacia adelante trazando otra línea de color aleatorio hasta el siguiente obtáculo que encuentre. El programa repite este ciclo indefinidamente. Guarda el programa como `prog20.irobo`.

EJERCICIO 21. Robot copión (2). (Mapas: copyLine1.map y copyLine2.map). Escribe un programa que copie la línea negra a la izquierda del robot como una línea blanca cuya longitud sea el *doble* de la línea negra original. PISTA: Si en el mapa copyLine2.map crees que no te cabe la línea que debes pintar, piensa que siempre puedes empezar a pintar un poco más abajo. Guarda el programa como `prog21.irobo`.

EJERCICIO 22. Sendero al punto negro. (Mapas: Todos los mapas verticalLines.map). Escribe un programa que lleve al robot hasta el punto negro siguiendo el camino pintando de blanco. El robot sólo puede moverse sobre casillas pintadas de blanco. Guarda el programa como `prog22.irobo`.

[22] Hablaremos sobre los valores de retorno de las instrucciones de visión en esta misma sección. Los procedimientos y sus valores de retorno son el objeto de estudio de la siguiente sección.

1.6. EL LENGUAJE ROBOMIND (4): PROCEDIMIENTOS. (*)

PROCEDIMIENTOS.

A estas alturas ya deberíamos dominar las instrucciones básicas de Robomind, como `forward(n)`, `right`, `pickUp`, `paintWhite`, `frontIsObstacle`, `flipCoin`, etc. Al recurrir a estas funciones, las usamos sin saber cómo están construidas internamente. Pero Robomind no solo incluye estas instrucciones preconstruidas, sino que también nos ofrece la posibilidad de construir nuestras propias instrucciones, a las que se llamamos procedimientos. Un **procedimiento** es una especie de miniprograma dentro de nuestro programa, que luego podemos reutilizar como si de una nueva instrucción se tratase.

Definir un procedimiento.

La sentencia `procedure` nos permite definir un nuevo procedimiento con el nombre que queramos:

```
procedure nombre (param 1, param 2, …)
{
      instrucciones
}
```

El procedimiento puede tener **parámetros** de entrada con nombres a nuestra elección. Estos parámetros son los datos que le debemos "pasar" a nuestro procedimiento para que funcione, y que podemos utilizar como variables en el código que define su funcionamiento. El código del procedimiento no se ejecuta automáticamente, sino que debemos hacer una **llamada al procedimiento** desde el programa principal cada vez que queramos ejecutarlo. Un ejemplo de procedimiento es el siguiente:

```
# Este procedimiento permite dibujar un rectángulo de anchura y una altura
# configurables mediante parámetros.

procedure rectangulo(anchura, altura)
{
      paintWhite
      repeat(2)
      {
            forward(altura)
            right
            forward(anchura)
            right
      }
      stopPainting
}
```

Llamar a un procedimiento.

Una llamada a un procedimiento tiene la siguiente estructura:

```
nombre(param 1, param 2, …)
```

La llamada al procedimiento se hace mediante el nombre que le hemos dado al procedimiento en su definición, escribiendo entre paréntesis los valores de los parámetros que necesita el procedimiento para funcionar.

A continuación tenemos un ejemplo de llamada al procedimiento rectángulo:

```
# Este es el programa principal, desde el cual llamamos al procedimiento.

forward(1)
rectangulo(3,2) # Una llamada al procedimiento rectangulo().
forward(3)
rectangulo(1,4) # Otra llamada con diferentes valores para los parámetros.

# Definición del procedimiento rectangulo(). Las llamadas al procedimiento
# deben proporcionar los valores para los parámetros 'anchura' y 'altura'.

procedure rectangulo(anchura, altura)
{
      paintWhite
      repeat(2)
      {
            forward(altura)
            right
            forward(anchura)
            right
      }
      stopPainting
}
```

LA INSTRUCCIÓN RETURN.

La instrucción `return` nos permite forzar la salida de un procedimiento, incluso aunque no hayamos llegado a la última instrucción de su definición. El programa principal continuará ejecutando el resto de instrucciones que haya tras la llamada al procedimiento del que acabamos de salir. A continuación tenemos un ejemplo de uso:

```
# Programa principal.

aLaPared() # Llamada la procedimiento aLaPared().
right
forward

# Defínción del procedimento aLaPared().
# Este procedimiento permite avanzar paso a paso hasta llegar a una pared.
# (Notar que este procedimiento no necesita parámetros para funcionar).

procedure aLaPared()
{
      repeat
      {
            if(frontIsObstacle)
            {
                  return   # Si vemos un obstáculo, salimos del procedimiento.
            }
            else
            {
                  forward    # En caso contrario, avanzamos un paso.
            }
      }
}
```

Como vemos, la instrucción `return` nos permite salir de un procedimiento, de la misma forma que la instrucción `break` nos permite salir de un bucle. Sin embargo, una diferencia importante entre ambas es

que la instrucción `return` permite que el procedimiento pueda devolver un cierto valor, que luego podemos usar en el programa principal tras la llamada al procedimiento. Un ejemplo es el siguiente:

```
# Programa principal.

casillas = doble(3)   # Guardamos en la variable 'casillas' el valor devuelto
                      # por la llamada doble(3), a saber, 6.

forward(casillas)     # Avanzamos el valor guardado en la variable 'casillas'.

# Definición del procedimiento doble(n).
# Este procedimiento calcula el doble del valor que le pasamos como argumento.

procedure doble(n)
{
    return(2 * n)
}
```

RECURSIVIDAD.

Un procedimiento puede llamarse a sí mismo desde el código de su propia definición. A esta técnica de programación se la denomina **recursividad**, y es muy útil para resolver ciertas tareas. A continuación tenemos un ejemplo de uso de la recursividad:

```
# Programa principal.

aLaPared()
right
forward

# Definición recursiva del procedimiento aLaPared().

procedure aLaPared()
{
     # Notar que en esta versión recursiva no usamos bucles.

     if(frontIsObstacle)
     {
          return     # Si encontramos un obstáculo, salimos del bucle.
     }
     else
     {
          forward    # En caso contrario, damos un paso hacia adelante y …
          aLaPared() # … hacemos una llamada recursiva al mismo procedimiento!
     }
}
```

EJERCICIO 23. Cuadrado. (Cualquier mapa). Escribe un procedimiento llamado `cuadrado` que reciba dos parámetros (`lado` y `color`), y que le permita al robot dibujar un cuadrado con el tamaño definido por `lado`, y del color definido por `color` (donde 0 significa negro, y 1 blanco). El procedimiento debe funcionar en cualquier situación, de forma que si un obstáculo le impide terminar el cuadrado, el programa se detiene. Para comprobar su correcto funcionamiento, realiza unas cuantas llamadas al procedimiento en el programa principal. Guarda el programa como `prog23.irobo`.

EJERCICIO 24. Curso y grupo. (Mapa: openArea.map). Escribe un programa en el que el robot pueda dibujar por pantalla el curso y el grupo que el usuario decida en el programa principal. Por simplicidad, para el curso solo podrá elegir 1, 2, 3, o 4; y para el grupo, solo A o B. Para ello necesitarás escribir 4 procedimientos para el curso, y dos procedimientos para el grupo. El usuario llamará en el programa

principal a los dos procedimientos adecuados para pintar su curso y grupo. Guarda el programa como `prog24.irobo`.

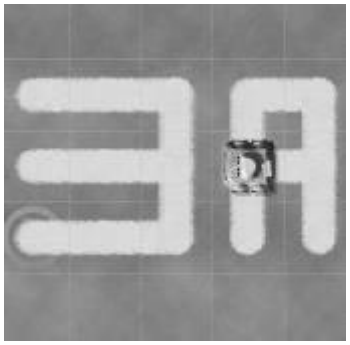


Figura 1.12. Ejercicio 24.

1.7. EL LENGUAJE ROBOMIND (5): CONDICIONES MÚLTIPLES.

EXPRESIONES LÓGICAS Y OPERADORES BOOLEANOS.

A las condiciones que gobiernan los bucles y las estructuras condicionales se les llama **expresiones lógicas**, porque al evaluarlas siempre devuelven True (verdadero, se cumple, sí, etc.) o False (falso, no se cumple, no, etc.). Las expresiones lógicas que hemos usado hasta ahora constaban de una sola instrucción de visión:

```
repeatWhile(frontIsClear)
{
    forward(1)
}
```

Sin embargo, a veces necesitaremos controlar nuestros bucles y condicionales de forma más precisa, mediante múltiples instrucciones de visión combinadas con los operadores booleanos `and`, `or`, y `not`:

```
repeatWhile(frontIsClear and leftIsWhite)
{
    forward(1)
}
```

La tabla 1.6 muestra un resumen de estos operadores booleanos:

OPERADORES BOOLEANOS.			
OPERADOR	NOTACIÓN ALTERNATIVA	EXPLICACIÓN	EJEMPLO
`not`	~	Niega la expresión lógica que le sigue.	`not frontIsBeacon`
`and`	&	Devuelve True solo cuando las dos expresiones lógicas que opera son ambas True.	`frontIsClear and rightIsWhite`
`or`	|	Devuelve True cuando cualquiera de las dos expresiones lógicas que opera es True.	`leftIsWhite or rightIsWhite`

Tabla 1.6. Operadores booleanos.

Como los operadores aritméticos (suma, resta, multiplicación, y división), los operadores booleanos obedecen una jerarquía. En operaciones combinadas, el operador que primero actúa es el `not`, luego el `and`,

y finalmente el `or`. Podemos modificar el orden en el que se efectúan las operaciones poniendo entre paréntesis aquellas a las que queramos dar prioridad. A continuación mostramos un par de ejemplos:

```
# Ejemplo 1.
if(leftIsBlack and not rightIsWhite)
{
      forward(1)
}

# Ejemplo 2.
repeatWhile(frontIsClear and (leftIsWhite or rightIsWhite))
{
    forward(1)
}
```

EFICIENCIA Y ESTADÍSTICAS.

A la hora de escribir un programa de control, no basta con que funcione correctamente; además debe ser eficiente. Un programa eficiente es aquel que, funcionando correctamente, necesita la menor cantidad de código posible, se ejecuta en el mínimo tiempo posible, y usa los mínimos recursos posibles. A modo de ejemplo, imaginar que tenemos dos programas de control que hacen que el robot salga de un laberinto. Obviamente, el mejor programa será aquel que lo haga salir en el mínimo tiempo posible, ejecutando el menor número de movimientos posibles, y realizando el mínimo número de comprobaciones posibles. Para evaluar la eficiencia, Robomind incorpora una herramienta que muestra las estadísticas del programa que estamos ejecutando. Para activar las estadísticas, acudimos al menú View >> Show stats. Como vemos, la ventana de estadísticas muestra cuántas instrucciones ha efectuado el robot para resolver una cierta tarea.

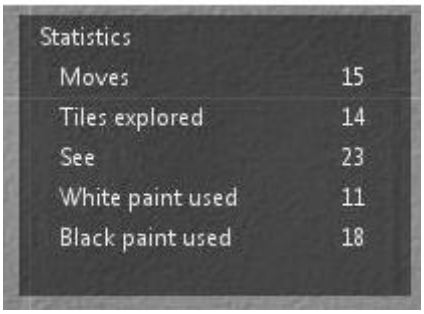


Figura 1.13. Estadísticas.

EJERCICIO 25. Enjaulado. (Mapas: roboCage.map). En este mapa verás que el robot se encuentra dentro de un recinto delimitado por una línea negra. Programa al robot para que se mueva aleatoriamente dentro de este recinto, pero sin poder abandonarlo. Guarda el programa como `prog25.irobo`.

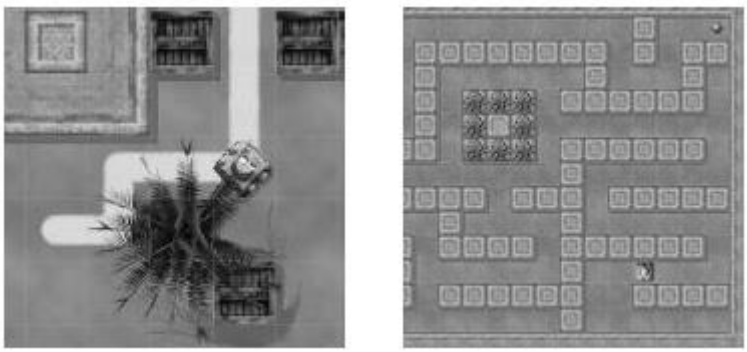


Figura 1.14. Ejercicio 26 y 27.

EJERCICIO 26. Sigue líneas (1). (Mapas: Cualquier mapa con una línea blanca). Escribe un programa para conseguir un robot seguidor de línea blanca. El programa consta de dos partes: (1) En primer lugar el robot

debe moverse por el mapa de forma autónoma, buscando la línea blanca. (2) Cuando el robot encuentra una línea blanca, debe empezar a seguirla de forma automática (sin saber a priori por dónde discurre la línea), deteniéndose cuando detecta que la línea blanca se ha acabado. Puedes probar el programa en el mapa default.map. Guarda el programa como `prog26.irobo`.

EJERCICIO 27. Laberinto (1). (Mapas: Todos los mapas maze.map). El objetivo del programa es conseguir que el robot escape del laberinto de forma autónoma. El robot encuentra la salida al localizar y recoger la baliza, momento en el que termina el programa. Para salir de estos laberintos, el método más sencillo es seguir siempre la pared de la derecha, o la pared de la izquierda. Guarda el programa como `prog27.irobo`.

1.8. EJERCICIOS FINALES.

Para terminar os propongo una serie de ejercicios en cuya resolución podréis emplear libremente todos los conceptos y herramientas que hemos estudiado en el capítulo.

EJERCICIO 28. Sigue líneas (2). (Mapas: rm-lf-task2.map). En este ejercicio debemos programar al robot para seguir la línea blanca. Pero en el mapa rm-lf-task2.map la línea blanca está interrumpida por puntos negros. Consigue que el robot sea capaz de seguir la línea blanca a pesar de las interrupciones. Guarda el programa como `prog28.irobo`.

EJERCICIO 29. Parking. (Mapas: rm-lf-task3.map). En este programa el robot debe avanzar de forma aleatoria sobre la rejilla de líneas blancas, en busca de un lugar donde aparcar (el punto negro). Para recorrer el parking de forma automática, el robot debe cambiar de dirección aleatoriamente cada vez que llega a un cruce de caminos (derecha, al frente, o izquierda). ¿Cómo decidir aleatoriamente entre tres opciones posibles? Tal vez necesites "anidar" 2 instrucciones `flipCoin` (esto es, meter una instrucción `flipCoin` dentro de otra). Guarda el programa como `prog29.irobo`.

EJERCICIO 30. Campo de minas. (Mapas: rm-lf-task4.map). El programa es muy similar al anterior: El objetivo es que el robot llegue al punto negro, recorriendo aleatoriamente la rejilla de líneas blancas. Sin embargo, cuando encuentra una baliza, el robot debe evitarla y dirigirse por otro camino. Guarda el programa como `prog30.irobo`.

EJERCICIO 31. Borra las marcas. (Mapas: paintSpots.map). Escribe un programa que haga que el robot avance de forma aleatoria y autónoma por el mapa. Cuando el robot encuentre una marca negra, deberá pintarla blanca. El programa no acabará nunca, puesto que el robot no sabe cuántas marcas negras debe pintar. Guarda el programa como `prog31.irobo`.

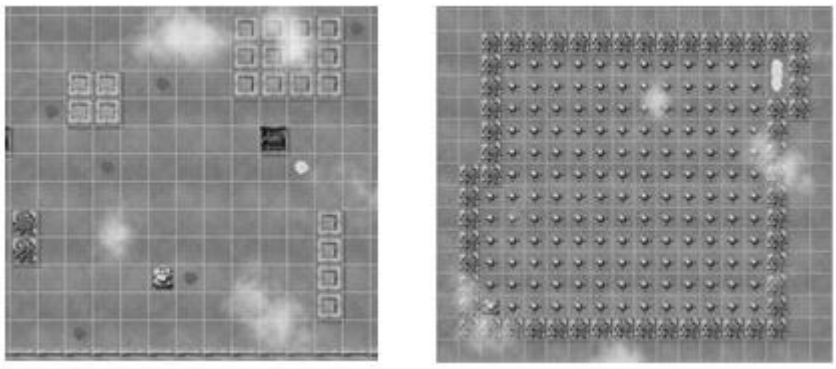


Figura 1.15. Ejercicios 31 y 32.

EJERCICIO 32. A través de las balizas. (Mapas: Todos los mapas findLines.map). Escribe un programa que permita al robot llegar a la meta, que está marcada mediante líneas blancas. Para ello, hay que tener en cuenta que el robot no sabrá dónde está la meta, por lo que deberá moverse aleatoriamente a través de las

balizas, retirándolas para poder avanzar. El programa termina automáticamente cuando el robot llega a la meta y se posiciona encima. IMPORTANTE: En este programa está prohibido destruir las balizas. Para quitar la baliza que te impide pasar, debes recogerla y retirarla allá donde puedas. Guarda el programa como `prog32.irobo`.

EJERCICIO 33. Busca el punto negro. (Mapas: rm-lf-task5.map). En este mapa hay un punto negro escondido en algún lugar. Tu misión es programar al robot para encontrarlo. Guarda el programa como `prog33.irobo`.

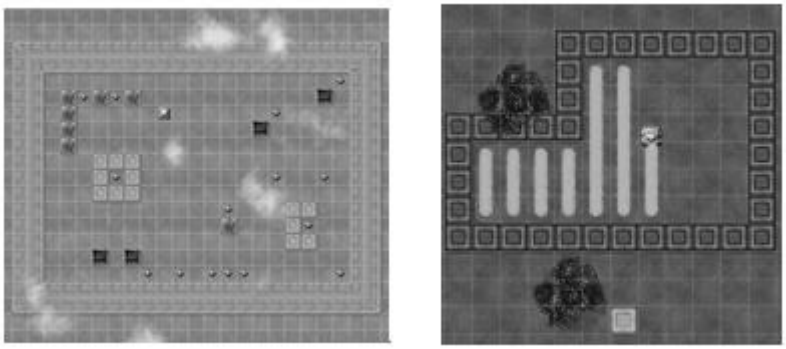


Figura 1.16. Programas 33 y Cortacésped.

AMPLIACIÓN 1: Cortacésped. (Mapa: Todos los mapas mowGrass.map). Escribe un programa que le permita al robot cortar todo el césped del recinto. El corte de césped lo simularemos pintando el suelo de blanco. Guarda el programa como `ampliacion1.irobo`.

AMPLIACIÓN 2. Ordena balizas. (Mapa: potholes.map). Escribe un programa que permita que el robo recoja las balizas situadas en el sendero blanco, y la coloque en las posiciones marcadas con un punto negro. Aunque solo lo vamos a probar en un mapa, el programa debe funcionar igual de bien independientemente del númro de balizas ubicadas en el sendero blanco, y de las ubicaciones de los puntos negros donde debemos colocarlas. Guarda el programa como `ampliacion2.irobo`.

AMPLIACIÓN 3. Espiral. (Cualquier mapa). Escribe un programa que le permita al robot dibujar una espiral como las mostradas en las figuras. El robot continuará dibujando hasta que encuentre un obstáculo que le impida seguir progresando. Diseña una solución genérica que funcione en cualquier mapa. Guarda el programa como `ampliacion3.irobo`.

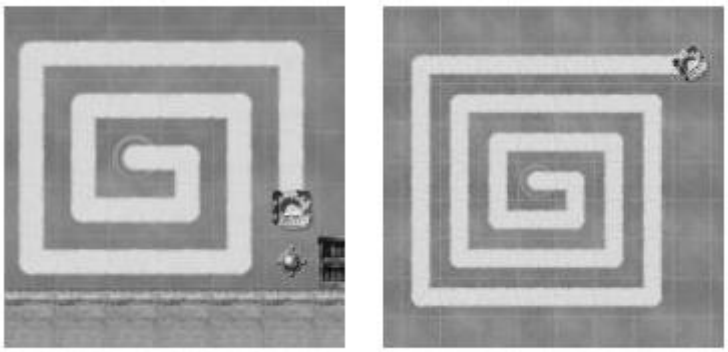


Figura 1.17. Espiral.

AMPLIACIÓN 4. El cuadrado más grande. (Mapas: Todos los mapas con espacio para trazar un cuadrado de tamaño 2×2). Escribe un programa que le permita al robot trazar el cuadrado más grande que pueda, partiendo de la posición inicial en la que el robot aparece en el mapa, y trazándolo hacia la derecha. (El colo con el que el robot trazará el cuadrado es aleatorio, pudiendo ser blanco o negro). Este programa debe funcionar en cualquier mapa que permita trazar, al menos, un cuadrado de tamaño 2×2. Por ejemplo, si probamos el programa en el mapa default.map, el robot debería trazar un cuadrado de tamaño 4×4. Si

probamos el programa en el mapa paintSpots.map, el robot trazará un cuadrado de tamaño 5×5. Y si lo probamos en el mapa boxpath1.map, el robot traza un cuadrado de tamaño 3×3. Guarda el programa como `ampliacion4.irobo`.

AMPLIACIÓN 5: Laberinto (2). (Mapas: Todos los mapas maze.map). El algoritmo aplicado en el ejercicio 28 no es válido para cualquier tipo de laberinto, solo para algunos de ellos (como los laberintos en los que lo has probado). Ahora recurriremos a un método distinto: El **algoritmo de Tremaux** sirve para salir de cualquier laberinto. Si el laberinto no tiene salida, nos lleva de vuelta a la entrada. Observa el algoritmo en funcionamiento: http://www.youtube.com/watch?v=dfwzjtjndks. Los pasos a seguir son los siguientes:

- Conforme el robot avanza, dibuja una línea blanca detrás del robot para marcar el camino que ya ha recorrido.
- Si el robot llega a un callejón sin salida, da la vuelta y vuelve por el mismo camino.
- Cuando el robot llega a un cruce que no haya visitado antes, elige un camino aleatoriamente.
- Cuando el robot llega a un cruce que ya ha visitado antes, lo considera un callejón sin salida y vuelve por el mismo camino.
- Si el robot está recorriendo un camino que ya ha recorrido antes (es decir, un camino que ya esté pintado) y encuentra un cruce, el robot va en la dirección que no haya recorrido antes, y si no la hay, va por un camino que ya haya recorrido.
- Si el robot está recorriendo un camino por el que ya ha pasado antes (es decir, pintado en blanco), y se ve obligado a retroceder en dirección opuesta, dibuja una línea detrás de él en color negro. Ese camino no debe volver a recorrerlo.
- Todos los caminos estarán sin pintar (los que no haya recorrido nunca), pintados de blanco (los que ha recorrido una vez), o pintados de negro (los que ya ha recorrido una vez y se ha visto obligado a retroceder en la dirección opuesta, los cuales debe evitar de ahora en adelante).
  (Fuente: http://www.astrolog.org/labyrnth/algrithm.htm).

Prueba tu programa en el mapa bigmaze.map. Guarda el programa como `ampliacion5.irobo`.

2. ROBÓTICA AVANZADA CON GEARS.

2.1. INTRODUCCIÓN A GEARS.

¿QUÉ ES GEARS?

Gears (también conocido como **GearsBot**) es un simulador robótico en 3D. En Gears, los robots pueden programarse mediante bloques (usando el lenguaje Blocky), mediante conversión automática de bloques a lenguaje Python, o directamente en Python (con las librerías Ev3dev y Pybricks). El código Python generado puede userse para controlar robots reales sin variaciones del programa, o con muy pocas modificaciones.

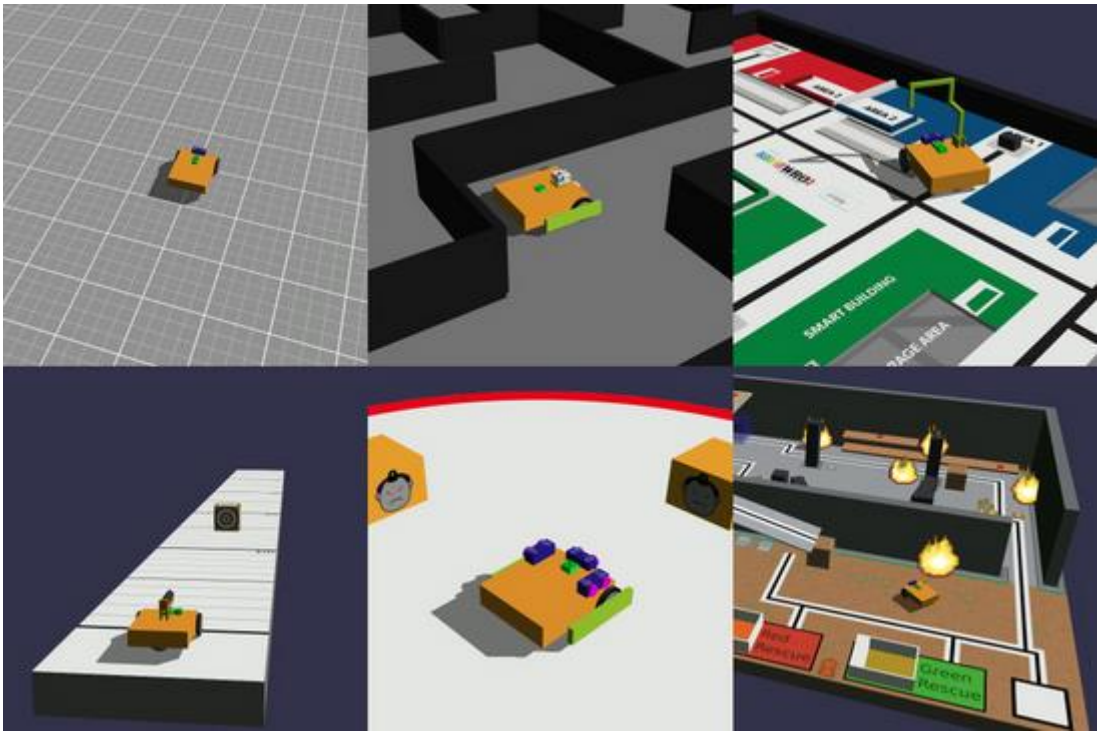


Figura 1. Gears.

Otros simuladores robóticos (como Robomind) simplifican la física de las simulaciones para ser más accesibles para los principiantes, pero Gears intenta conseguir una simulación lo más realista posible. Esto significa que, como les ocurre a los robots reales, los robots de Gears experimentarán derrapes en sus neumáticos, y otros efectos que impedirán que los robots puedan moverse recto sin la ayuda de una línea pintada en el suelo, o de un giróscopo. Por ello, es posible que los estudiantes noveles encuentren Gears un poco más difícil que otros simuladores, pero a cambio, nos permitirá aprender cómo tratar con las dificultades propias de los robots reales.

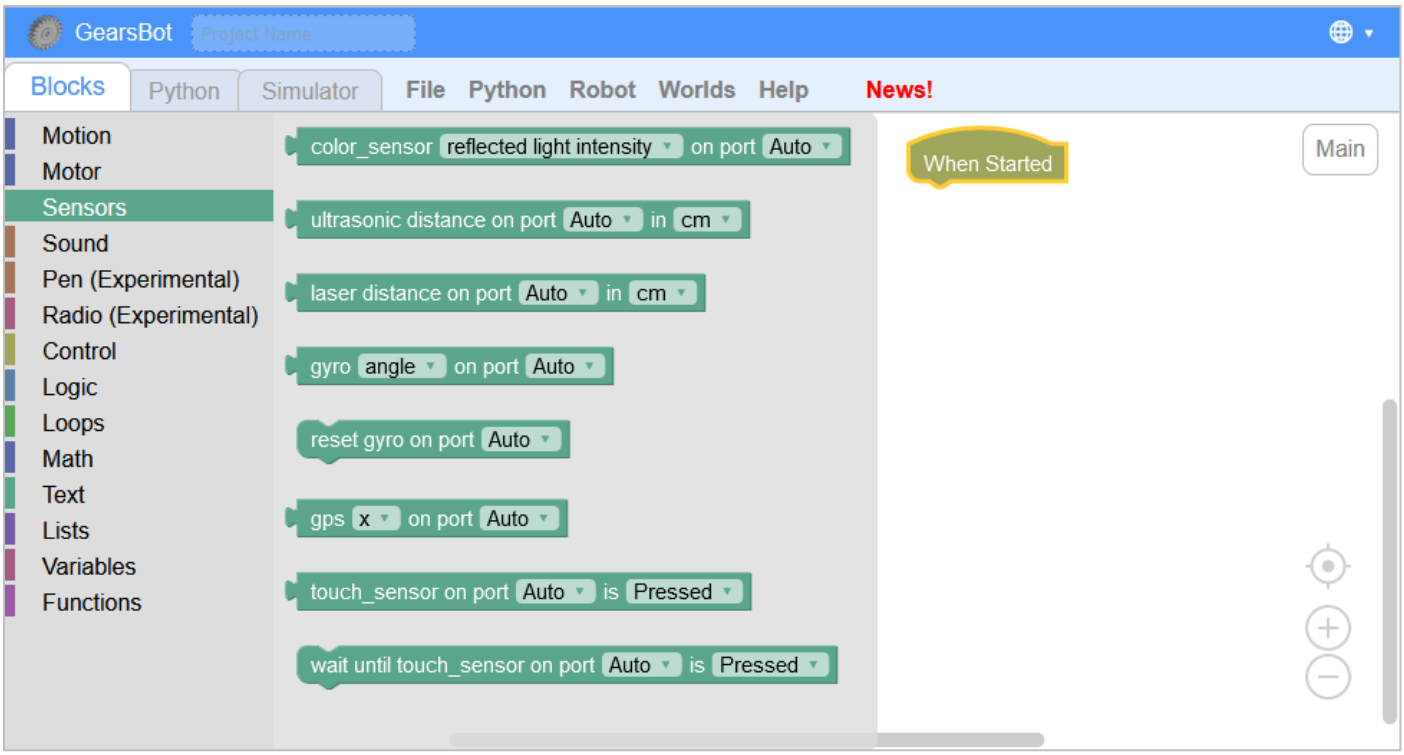
EL ENTORNO DE PROGRAMACIÓN DE GEARS.

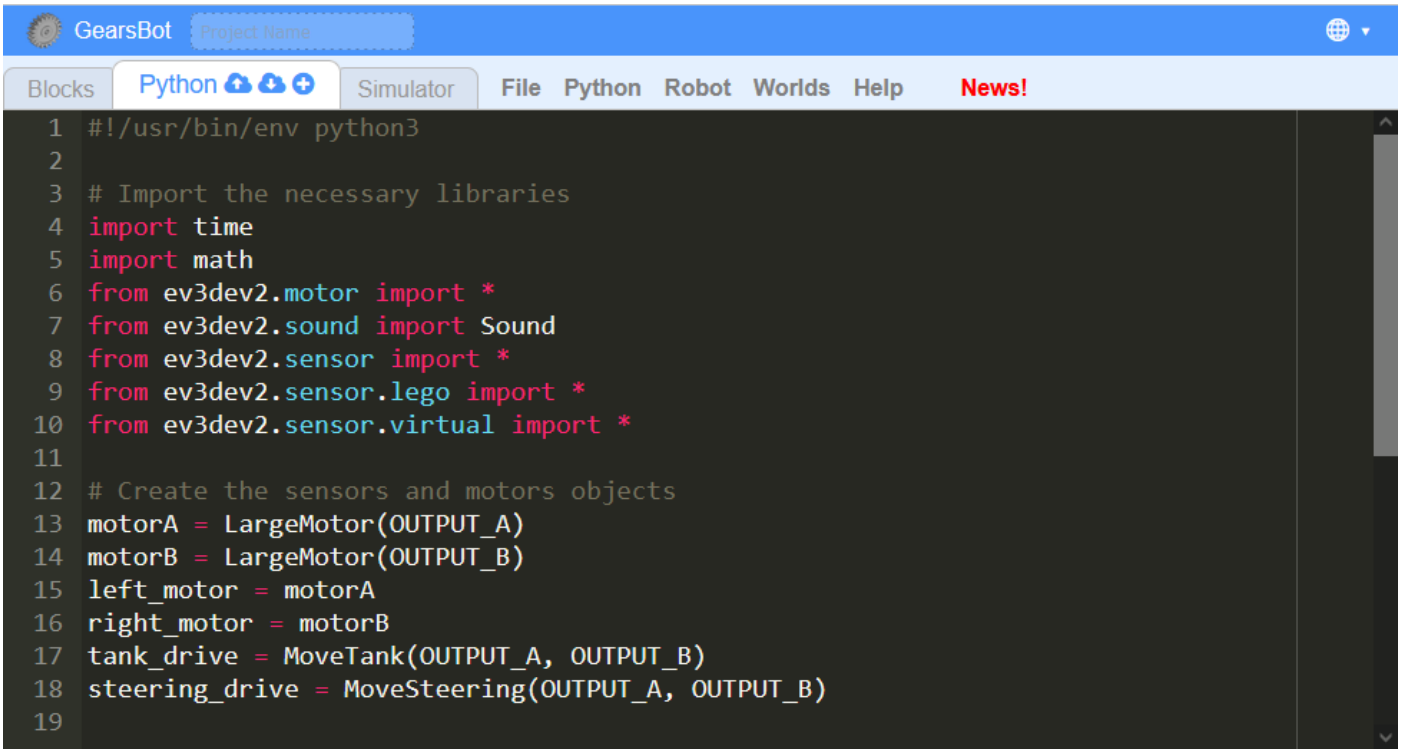
Para acceder al entorno de programación de Gears, abrimos nuestro navegador favorito y accedemos a la web https://gears.aposteriori.com.sg/. Aquí nos encontramos con tres pestañas: Bloques, Python, y Simulador.

En la pestaña de **Bloques** podemos programar nuestro robot mediante un lenguaje similar a Scratch. Para construir el programa de control seleccionamos los bloques necesarios de las paletas de la izquierda, y los arrastramos y soltamos en la zona de trabajo de la derecha.

La pestaña **Python** nos ofrece otra forma de programar al robot, usando un lenguaje de uso profesional como es Python. Gears es capaz de convertir los bloques a código Python, pero los estudiantes más avanzados pueden decidir programar al robot directamente en lenguaje Python.

La pestaña **Simulador** nos permite ver al robot ejecutando el programa de control que hayamos escrito. El botón de Ejecución ▷ sirve para lanzar el programa. También disponemos de botones para cambiar la vista del robot, para cambiar el mapa donde se ejecuta el programa, para conocer su posición y orientación actuales, para consultar el estado de sus sensores, etc.





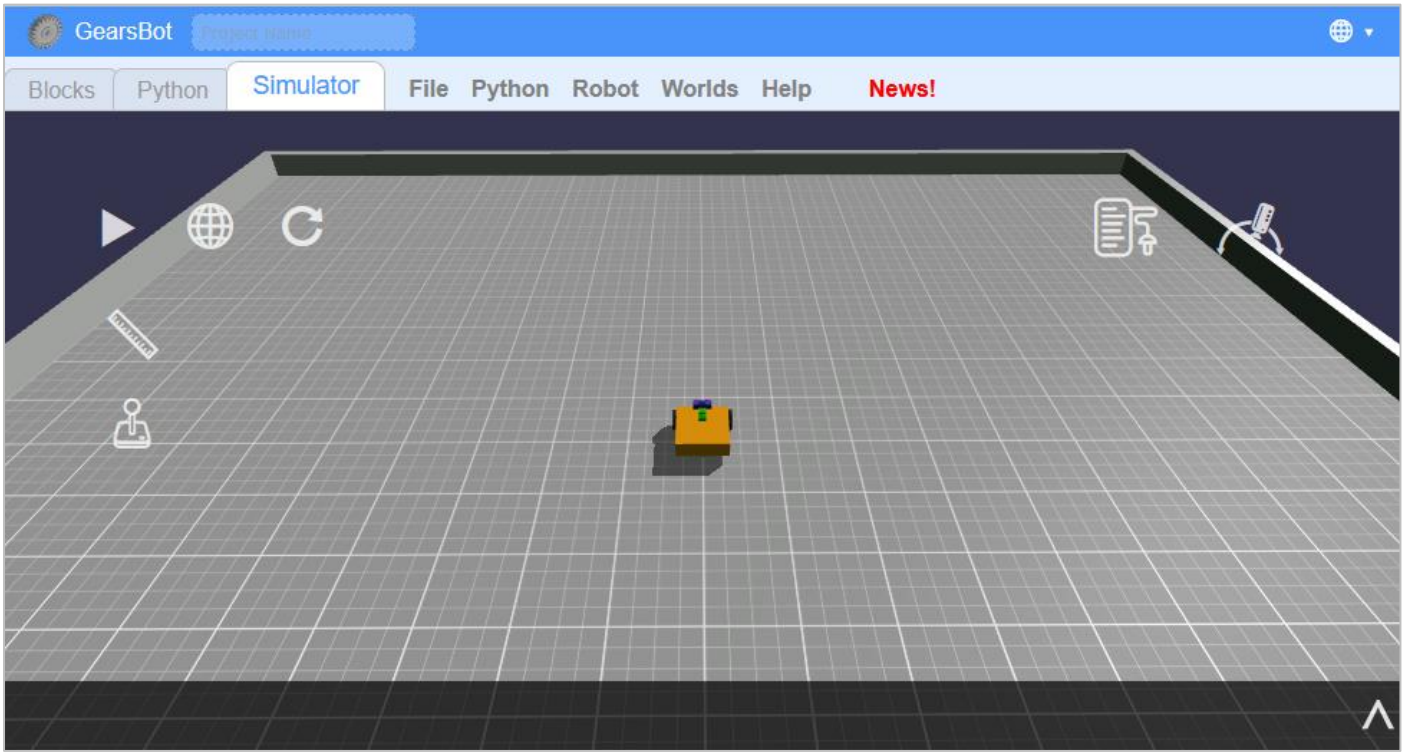


Figura 2. Las pestañas de Bloques, Python, y Simulador.

CAMBIAR DE MUNDO.

Para cambiar el **mundo** (esto es, el entorno en el que el robot ejecuta el programa), acudimos a la pestaña del Simulador, y pinchamos en el icono de la bola del mundo, y seleccionamos uno de los mundos o desafíos (challenges) preconstruídos que hay disponibles en la lista desplegable. Para cada uno de estos **mundos preconstruidos**, podemos configurar su tamaño (anchuro y longitud), si estará delimitado por paredes (y la atura y anchura de estas paredes), así como la posición inicial del robot.

También podemos cargar **mundos personalizados** si disponemos del archivo JSON que lo describe. Para cargar un mundo personalizado, pulsamos el botón "Load" para buscar y añadir el archivo JSON del mundo personalizado.

CONFIGURAR EL ROBOT.

En Gears podemos seleccionar entre distintos tipos de robots, que se distinguen principalmente por los sensores y motores que incluyen. Para seleccionar uno de los robots que Gears trae por defecto, acudimos al menú "Robot" seleccionamos la opción "Select robot". Para cada uno de los robots disponibles en la lista, la ventana emergente nos informa sobre las dimensiones del robot elegido (diámetro de las ruedas, espciado entre ruedas, etc.), sus actuadores (motores, brazos robóticos, electroimanes, etc.), y sus sensores (color, distancia, GPS, giróscopo, etc.).

Como con los mundos, también tenemos la opción de caragr un robot personalizado si disponemos del archivo JSON que lo describe. Para ello, en el menú "Robot" seleccionamos la opción "Load from file". En la ventana emergente, seleccionamos el archivo JSON el robot y los cargamos.

Si en cualquier momento queremos consultar las características del robot seleccionado, podemos acudir a la pestana Simulador y pulsar en el botón de Sensores. Aparecerá una lista que muestra el estado actual de todos los sensores y motores que incorpora el robot. Notar que todos los sensores están conectados a un puerto de entrada identificado como `in#` (donde `#` es el número de puerto de entrada al que están conectados). Por su parte, los motores están conectados a un puerto de salida identificado como `outX`

(donde X es la letra que identifica el puerto de salida al que están conectados). Por cierto, que si el robot incluye un actuador de tipo electroimán, el motor no aparecerá en la lista de sensores (se localizará en el siguiente puerto de salida disponible). Sin embargo, todavía podremos consultar el puerto de salida del motor en la pestaña Python.

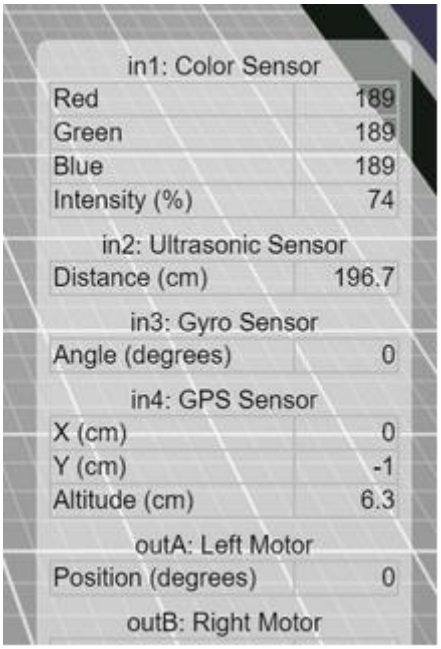


Figura 3. Características del robot seleccionado.

2.2. BLOQUES PARA MOVER AL ROBOT.

Para empezar a programar el movimiento del robot, utilizaremos el robot preconstruido "Single Sensor Line Follower" (el robot por defecto) y el mundo preconstruido "Grid Map" (el mundo por defecto). Las características de este robot son las siguientes:

- Dimensiones:
  - Diámetro de las ruedas: 5,6 cm.
  - Espaciado entre ruedas: 15,2 cm.

- Actuadores:
  - Puerto A: Motor izquierdo.
  - Puerto B: Motor derecho.
  - Puerto C: Electroimán.

- Sensores:
  - Puerto 1: Sensor de color.
  - Puerto 2: Sensor de distancia ultrasónico.
  - Puerto 3: Giróscopo.
  - Puerto 4: GPS.
  - Puerto 5: Lápiz.

Los bloques que nos permiten constrolar el movimiento del robot se encuentran en las bandejas "Motion" y "Motor". Los bloques de la bandeja "Motion" nos permiten mover los dos motores del robot de forma simultánea. Por su parte, los bloques de la bandeja "Motor" sirven para controlar cada uno de los motores de forma independiente, y para registrar la velocidad y posición actuales de estos motores. A menos que especifiquemos la cantidad de rotación, los motores que activemos continuarán moviéndose de forma indefinida; para detener los motores debemos utilizar los bloques de "Stop". Podemos consultar las

lecturas ofrecidas por los motores en la ventana de Sensores disponible en la pestaña del Simulador, como por ejemplo, la posición angular (en grados) de la rueda que están moviendo.

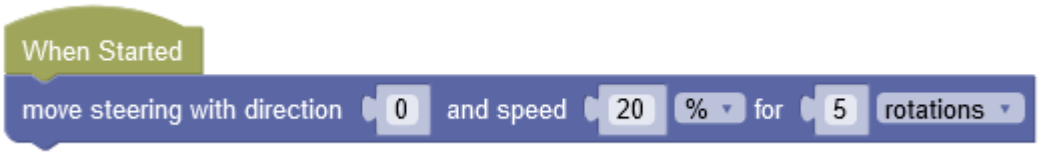
Figura 4. Bloques de movimiento (bandejas Motion y Motor).

A modo de ejemplo, vamos a escribir unos cuantos programas con los bloques de movimiento:

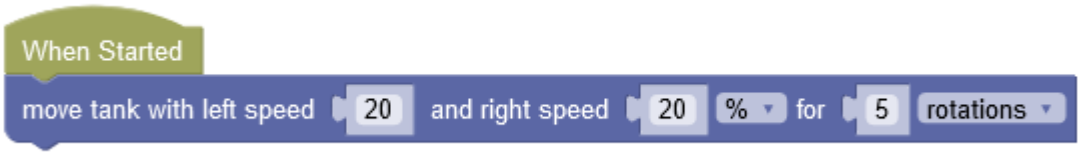
Hacer avanzar al robot en línea recta.

Vamos a utilizar el bloque "move steering" de la bandeja "motion" para hacer que el robot avance en línea recta. Escribimos el programa mostrado en la figura, y después acudimos a la pestaña de Simulación y ejecutamos el programa. En este programa, el bloque "When Started" indica que el programa se ejecutará al pulsar el botón de Ejecución ▷. En el campo de dirección indicamos la orientación del movimiento del robot, la cual varía desde -100 hasta 100: Valores negativos hacen girar al robot a la izquierda, valores positivos lo hacen girar a la derecha, y el valor 0 hace que el robot avance en línea recta. El campo de

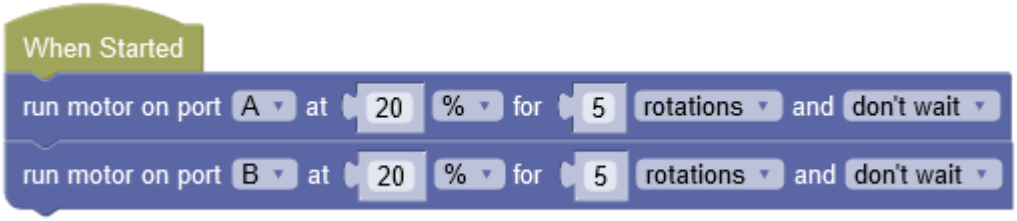
velocidad nos permite especificar lo rápido que se moverá el robot, y el último campo nos sirve para indicar la distancia o el tiempo del movimiento efectuado (en rotaciones, grados, segundos, o milisegundos). En este programa le estamos ordenando al robot que avance hacia adelante a una velocidad del 20% y una distancia equivalente a cinco rotaciones de sus ruedas (distancia que podemos calcular multiplicando el diámetro de las ruedas por π).



Otra alternativa para conseguir un resultado similar sería usar el bloque "move tank", como en el programa mostrado en la figura. En este caso, configuramos por separado la velocidad a la que se mueven los motores izquierdo y derecho del robot. Notar que, para que el robot avance en línea recta, hemos fijado el mismo valor de velocidad en ambos motores; de otra forma, el robot noo avanzaría en línea recta, sino girando hacia la derecha o hacia la izquierda.



También tenemos la opción de activar los dos motores por separado. El bloque "run motor on port" de la bandeja "motor" nos permite activar cada motor de forma independiente. El siguiente programa conseguiría el mismo movimiento que los anteriores. En este caso, es imporante seleccinar la opción "don't wait", porque de seleccionar la opción "wait for completion", el motor del puerto B no se activaría hasta que el motor del puerto A hubiese terminado de ejecutar su instrucción de movimiento, y percibiríamos que el robot giraría al tener uno de sus motores parado.

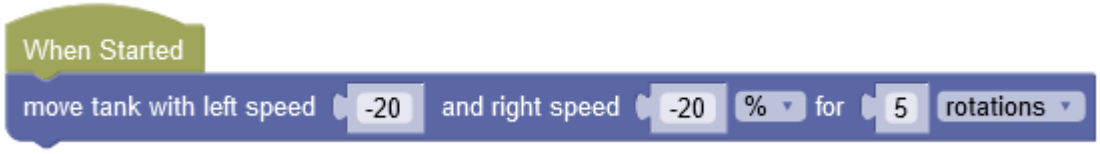


Es importante notar que, cuanto más brusco sea el movimiento del robot (por ejemplo, cuanto más rápido tratemos de moverlo), más difícil será conseguir que el robot avance en línea recta. Esta es una situación habitual en los robots reales, debido a causas muy variadas (los neumáticos pueden patinar, los dos motores podrían no estar bien equilibrados, el suelo podría no ser totalmente llano, los motores podrían no girar exactamente el número de rotaciones especificadas, un motor podría girar ligeramente más rápido que el otro, etc.). Gears considera todas estas situaciones para conseguir una simulación lo más realista posible. Más adelante veremos cómo conseguir que los robots se mueven en líena recta usando marcas en el suelo, o el sensor giróscopo del robot.

Hacer retroceder al robot en línea recta.

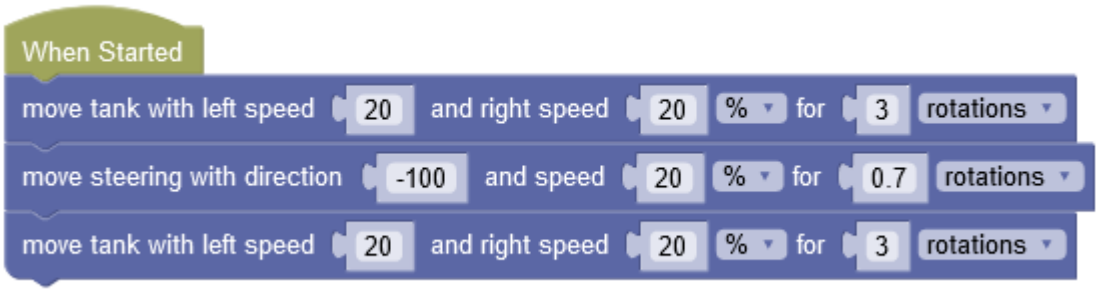
Para hacer retroceder al robot, basta con fijar un valor negativo en el campo de velocidad del bloque en cuestión. Por ejemplo, el siquiente programa haría retroceder al robot a una velocidad del 20% una distancia equivalente a cinco rotaciones de sus ruedas.

Notar que, para que el robot retroceda en línea recta, hemos fijado el mismo valor negativo de velocidad en ambos motores.

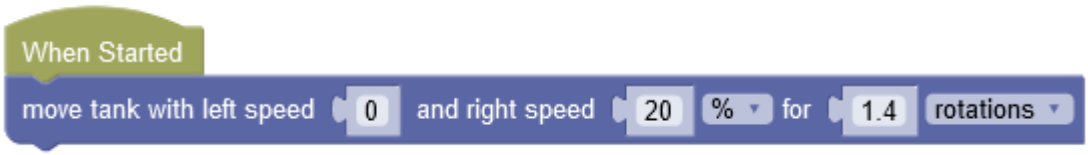


Hacer girar al robot.

Para hacer girar al robot disponemos de multiples alternativas. Tal vez la más sencilla sea usar el bloque "move steering", ajustando adecuadamente el valor del campo de dirección y el número de rotaciones para conseguir el movimiento deseado. A modeo de ejemplo, el siguiente programa hace que el robot avance en línea recta, efectúe un giro de 90º hacia la izquierda, y una vez orientado en esa nueva dirección, vuelva a avanzar en líena recta. Observar que, para conseguir que el robot gire 90º hacia la izquierda, hemos ajustado el valor de dirección a -100, y le hemos hecho girar durante 0,7 rotaciones de sus ruedas. Este último valor lo hemos fijado por el método de prueba y error, hasta conseguir la rotación deseada.



Otra forma de hacer girar al robot es usar el bloque "move tank" y ajustar distintos valores de velocidad a los motores izqiuerdo y derecho del robot. El siguiente programa para el motor izquierdo y solo mueve el motor derecho a una velocidad del 20% durante 1,4 rotaciones, consiguiendo así que el robot efectúe un giro a la izquierda de 90º aproximadamente. El valor del número de rotaciones se ha ajustado mediante prueba y error para conseguir el giro deseado.



2.3. BLOQUES PARA DIBUJAR E IMPRIMIR POR CONSOLA.

Algunos robots vienen con un lápiz incorporado, que pueden utilizar para dibujar un rastro sobre el suelo conforme se van moviendo. Para comenzar a dibujar y dejar de hacerlo, basta con usar los bloques de la bandeja "Pen". Esta bandeja también incluye bloques para cambiar el color y el grosor del trazo con el que el robot pinta sobre el suelo.

Además, los robots pueden comunicarse con el usuario imprimiendo textos por consola. Los bloques que nos permiten imprimir información por consola se localizan en la bandeja "Text". El bloque "Print" nos permite imprimir por consola cualquier texto, variable, o lectura de sensor que le conectemos a modo de argumento.

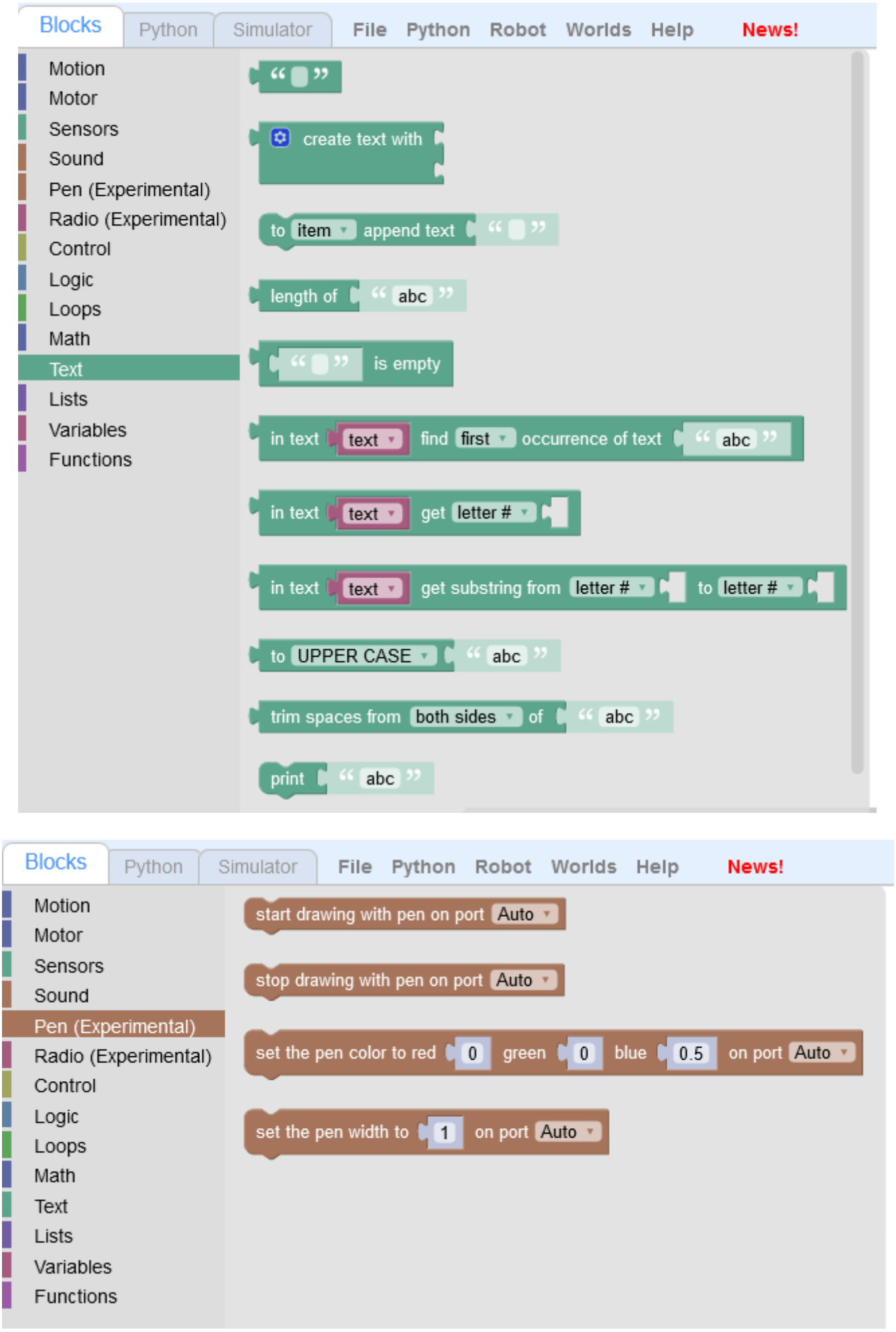
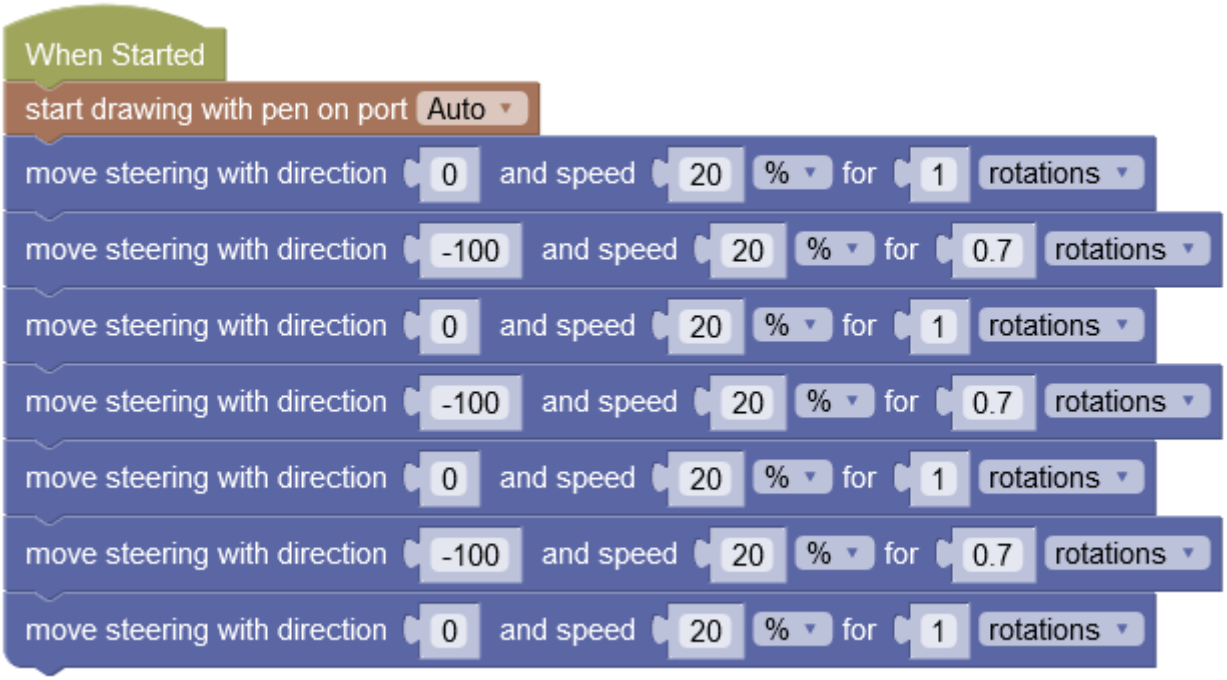
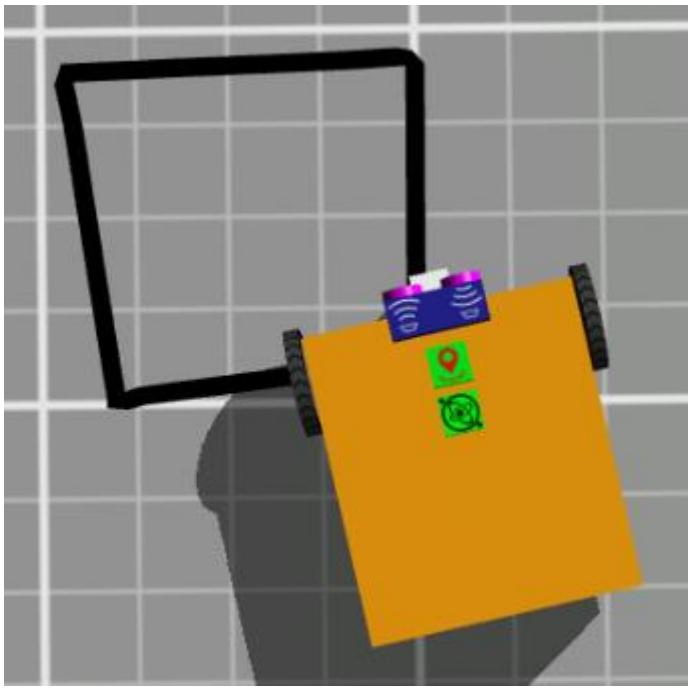


Figura 5. Bloques para imprimir por consola y para dibujar.

EJEMPLO: DIBUJAR UN CUADRADO.

Escribe el siguiente programa y ejecútamos en el mapa "Grid map":





https://sites.google.com/view/mintgenie/home/gearsbot-tutorial
https://www.canva.com/design/DAET1S-Efq4/jlOJJdAkWdGt-ND8Zw2n2Q/view?utm_content=DAET1S-Efq4&utm_campaign=designshare&utm_medium=link&utm_source=sharebutton
https://github.com/QuirkyCort/gears/wiki/Sensors-and-Actuators
https://github.com/QuirkyCort/gears/wiki/Challenges

https://www.aposteriori.com.sg/gearsbot-tutorial/ ()

https://www.aposteriori.com.sg/resources/

https://www.aposteriori.com.sg/oasis-primary-robotics/ (sumo bot)

https://github.com/QuirkyCort/gears/wiki/Tutorials (videotutoriales)

2.X. REFERENCIAS.

https://gears.aposteriori.com.sg/
https://www.aposteriori.com.sg/oasis-primary-robotics/
https://www.canva.com/design/DAET1S-Efq4/jlOJJdAkWdGt-ND8Zw2n2Q/view?utm_content=DAET1S-Efq4&utm_campaign=designshare&utm_medium=link&utm_source=sharebutton
https://sites.google.com/view/mintgenie/home/gearsbot-tutorial
http://www.legoengineering.com/program-virtual-robots-using-gears/
https://github.com/QuirkyCort/gears/wiki

https://github.com/QuirkyCort/gears/wiki/Tutorials
https://sites.google.com/view/mintgenie/home/gearsbot-tutorial
https://www.aposteriori.com.sg/oasis-primary-robotics/
https://www.aposteriori.com.sg/resources/

3. COMBATES DE ROBOTS CON GLADIABOTS.

3.1. QUÉ ES GLADIABOTS.

Gladiabots es un juego de estrategia orientado a los combates de robots. Nuestra misión es construir la **Inteligencia Artificial** (IA) de nuestro equipo de robots antes de enviarlos a combatir contra un escuadrón de robots enemigos. En Gladiabots configuramos nuestro equipo de robots, y escribimos su IA en un sencillo editor visual para hacer que las distintas unidades ataquen, recojan recursos, capturen bases enemigas, auxilien a sus compañeros, se aproximen o se alejen del enemigo, se retiren para regenerar sus escudos, etc. Dependiendo del resultado del combate, necesitaremos refinar y mejorar las IAs de nuestros robots hasta conseguir derrotar al escuadrán de bots oponentes.

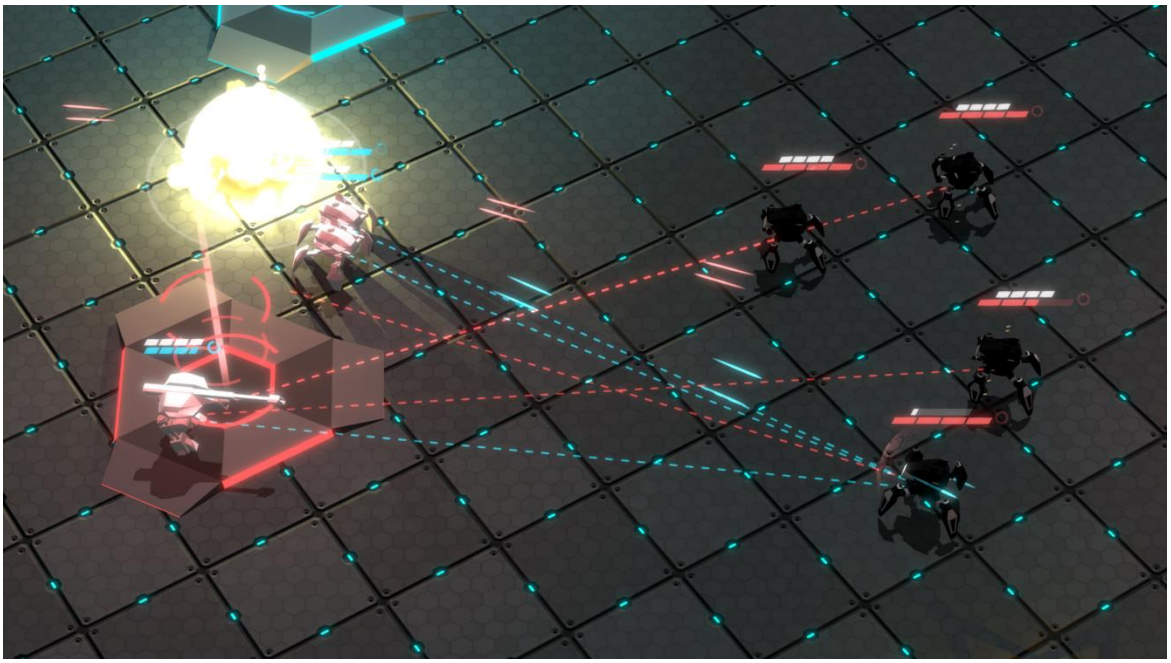


Figura 1.1. Combate de robots en Gladiabots.

3.2. MECÁNICA DEL JUEGO.

CONCEPTOS BÁSICOS.

Gladiabots es un juego de combate de robots entre dos jugadores: El **equipo azul** y el **equipo rojo**. Ambos jugadores configuran sus escuadrones y construyen las IAs de sus robots antes de enviarlos a la arena de combate. Al comenzar la batalla, los robots actúan de forma autónoma obececiendo sus IAs, y combaten entre sí hasta salir vencedores o ser derrotados.

ARENAS DE COMBATE (MAPAS).

Cada **arena de combate** (o **mapa**) contiene varios puntos de aparición para los robots de ambos equipos. Los mapas también incluyen objetos secundarios, como recursos, bases, campos de fuerza, y paquetes de salud. Los mapas se generan aleatoriamente de forma que la arena sea simétricamente idéntica para ambos jugadores, lo que garantiza la igualdad de oportunidades para los dos equipos. Los robots aparecen en el mismo orden en el que se colocaron en la pantalla de configuración del equipo (de la que hablaremos más adelante).

MODOS DE JUEGO.

En los tres modos de juego, el objetivo final es conseguir más puntos que el equipo contrincante. Sin embargo, los distintos modos de juego difieren en la forma en la que podemos conseguir esos puntos, y en las características adicionales que incluyen las arenas de combate.

Colección.

El objetivo de este modo de juego es reunir más recursos que el equipo contrario, dentro del tiempo límite establecido. Nuestro equipo anota puntos recolectando los recursos que aparecen en el mapa, y llevándolos a una de nuestras bases. Cada recurso obtenido y almacenado suma un punto. En este modo de juego, los robots destruidos vuelven a aparecer en la arena de combate. Las características de este modo de juego son las siguientes:

- Tamaño de los equipos: 4 vs 4.
- Tiempo límite: Transcurridos 5 minutos de juego, el equipo con más puntos gana la partida.
- Reaparición de los robots: Retardo de 10 segundos tras ser destruidos.
- Objetos especiales: Bases y recursos.

Dominación.

En este modo de juego el objetivo es capturar y controlar más bases que el equipo contrario, dentro del tiempo límite establecido. Para capturar una base (neutral o enemiga) basta con que un robot aliado consiga acercarse a ella. Aquí obtenemos puntos tanto por mantener nuestras bases bajo control, como por capturar bases neutrales o enemigas. También se consiguen puntos extra por cada robot aliado situado dentro de una de nuestras bases. En este modo de juego, los robots destruidos vuelven a aparecer en la arena de combate. Las características de este modo de juego son las siguientes:

- Tamaño de los equipos: 6 vs 6.
- Tiempo límite: Transcurridos 5 minutos de juego, el equipo con más puntos gana la partida.
- Reaparición de los robots: Retardo de 7,5 segundos tras ser destruidos.
- Objetos especiales: Bases y campos de fuerza.

Eliminación.

El objetivo aquí es destruir el mayor número de bots enemigos, dentro del tiempo límite establecido. Este modo de juego es el que tiene las reglas más sencillas, pero también es el más complicado de programar. Un combate termina en tablas si ningún equipo consigue destruir más enemigos que el otro. En este modo de juego, los robots destruidos *no* vuelven a aparecer en la arena de combate. Las características de este modo de juego son las siguientes:

- Tamaño de los equipos: 8 vs 8.
- Tiempo límite: Transcurridos 5 minutos de juego, el equipo con más puntos gana la partida.
- Reaparición de los robots: Los robots no reaparecen tras ser destruidos.
- Objetos especiales: Campos de fuerza y paquetes de salud.

OBJETOS ESPECIALES.

Dependiendo del modo de juego, los mapas pueden contener distintos tipos de objetos especiales:

Bases. (Modos de juego: Colección y Dominación). Los recursos obtenidos en modo Colección se almacenan en las bases. En el modo Dominación también podemos capturar bases enemigas o neutrales para

convertirlas en bases aliadas. Las bases aliadas que conservemos puntúan a lo largo del tiempo. El hecho de tener robots aliados en nuestras bases nos permite puntuar y capturar otras bases más rápidamente.

Recursos. (Modos de juego: Colección). Los recursos son los objetos que podemos recoger y almacenar en nuestras bases. Cada recurso guardado en una de nuestras bases suma un punto.

Campos de fuerza. (Modos de juego: Dominación y Eliminación). Cuando un robot captura un campo de fuerza, el daño que recibe se ve reducido en un 70% (esto es, 100 puntos de daño se convierten en solo 30 puntos de daño). El campo de fuerza no altera los puntos de salud del robot, que siguen siendo los mismos. Cada campo de fuerza solo puede ser capturado por un único robot. (Ver animación "Campo de fuerza.gif").

Paquetes de salud. (Modos de juego: Eliminación). Cuando un robot recoge un paquete de salud, sus puntos de salud se recuperan hasta su valor máximo. Los paquetes de salud no afectan a los puntos de escudo. Los paquetes de salud aparecen en localizaciones aleatorias a lo largo de la partida.

SALUD, ESCUDOS, Y REGENERACIÓN.

Cada tipo de robot dispone de una cierta cantidad de puntos de escudo y de salud. El escudo absorbe los daños que recibe el robot antes de afectar a sus puntos de salud, y puede regenerarse si el robot no recibe daño alguno durante un lapso de 3 segundos. Sin embargo, si los daños recibidos terminan afectando a nuestros puntos de salud, esta pérdida es permanente. El robot que pierde todos sus puntos de salud es destruido.

CLASES DE ROBOTS.

Gladiabots proporciona cuatro clases de robots con las que formar nuestro equipo (ver figura 1.2). Cada tipo de robot tiene sus ventajas y sus inconvenientes. La selección del tipo de robot que constituirá nuestro escuadrón se realiza en la **pantalla de configuración** de nuestro equipo. Sin embargo, los robots de los tipos amatralladora, escopeta, y francotirador solo pueden constituir el 35% de nuestro escuadrón.

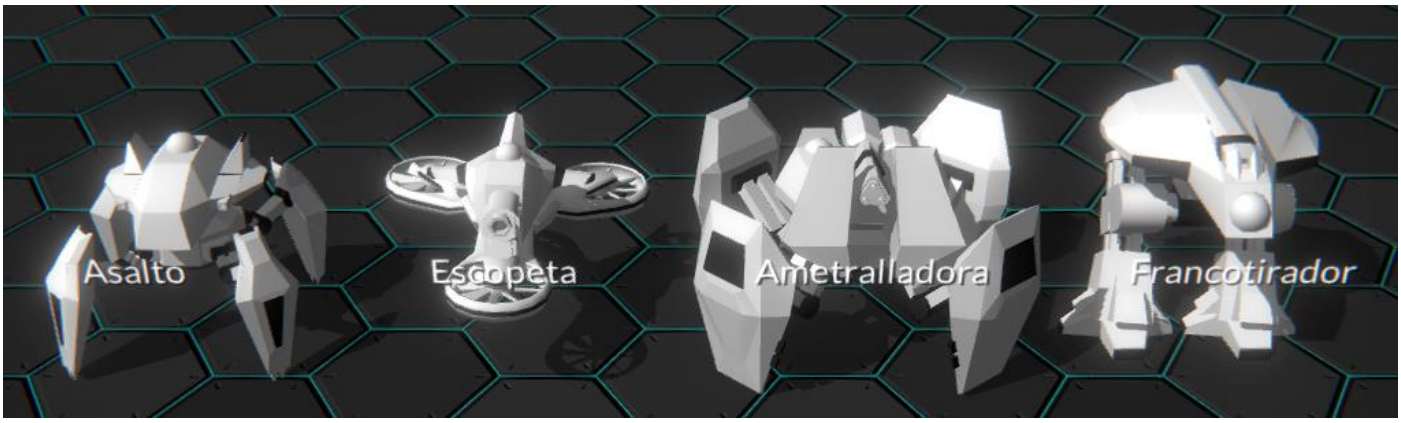


Figura 1.2. Las cuatro clases de robots de Gladiabots: Asalto, escopeta, ametralladora, y francotirador.

Asalto (Assault).

El **robot de asalto** es la clase por defecto de Gladiabots, y ofrece un rendimiento promedio en todos los aspectos:

- Velocidad: Media.
- Salud: Media.
- Tiempo para fijar el objetivo: Medio.
- Potencia de fuego: Media.

Escopeta (Shotgun).

El **robot escopeta** es ideal para el combate a corta distancia, para la obtención de recursos, y para la captura de bases enemigas.

- Velocidad: Rápido (excepto cuando está transportando un recurso).
- Salud: Muy débil.
- Escudo: Muy fuerte.
- Tiempo para fijar el objetivo: Rápido.
- Potencia de fuego: Causa enormes daños a corta distancia, pero no inflinge daños a grandes distancias.

Ametralladora (Machine gun).

El **robot ametralladora** se caracteriza por ser muy resistente y por tener una enorme potencia de fuego. Desafortunadamente, se trata de un robot muy lento a la hora de moverse y de fijar el objetivo:

- Velocidad: Lento (pero no se vuelve más lento cuando está transportando un recurso).
- Salud: Muy resistente.
- Tiempo para fijar el objetivo: Lento
- Potencia de fuego: Muy alta.

Francotirador (Sniper).

El **robot francotirador** es especialmente apropiado para el combate a grandes distancias. Como contrapartida, es muy lento y débil.

- Velocidad: Lento.
- Salud: Débil.
- Escudo: Débil.
- Tiempo para fijar el objetivo: Lento.
- Potencia de fuego: Capaz de causar daños a cualquier distancia (excepto si el objetivo está fuera de alcance).

Las siguientes gráficas muestran el rendimiento de los distintos tipos de robots en cuanto a velocidad de movimientos; puntos de salud y de escudo; velocidad de disparo; y potencia de fuego a corto, medio, y largo alcance.

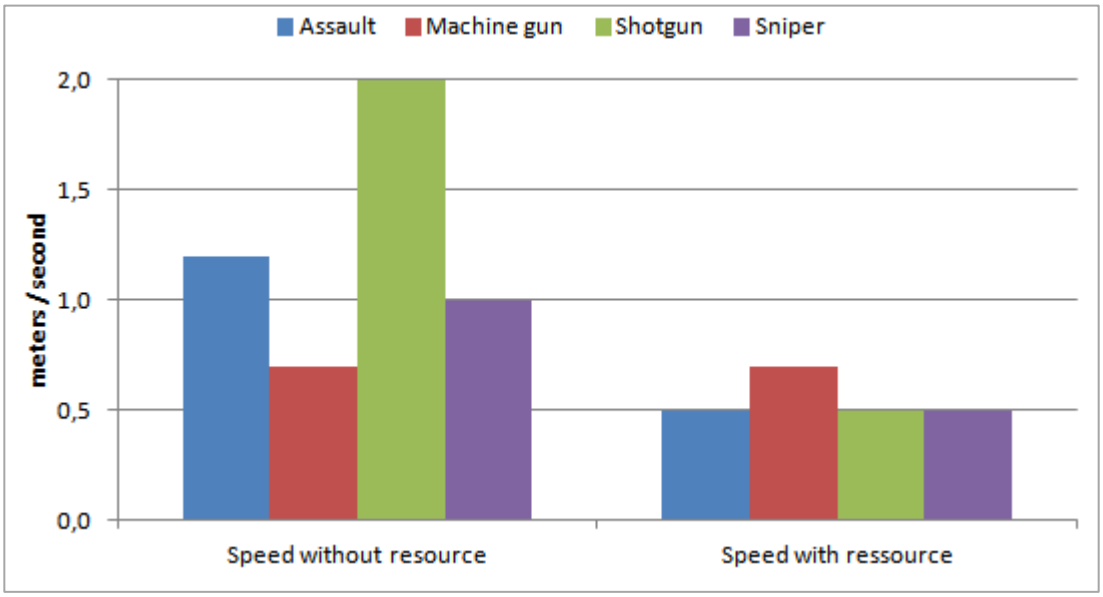


Figura 1.3. Gráfica de la velocidad de movimiento.

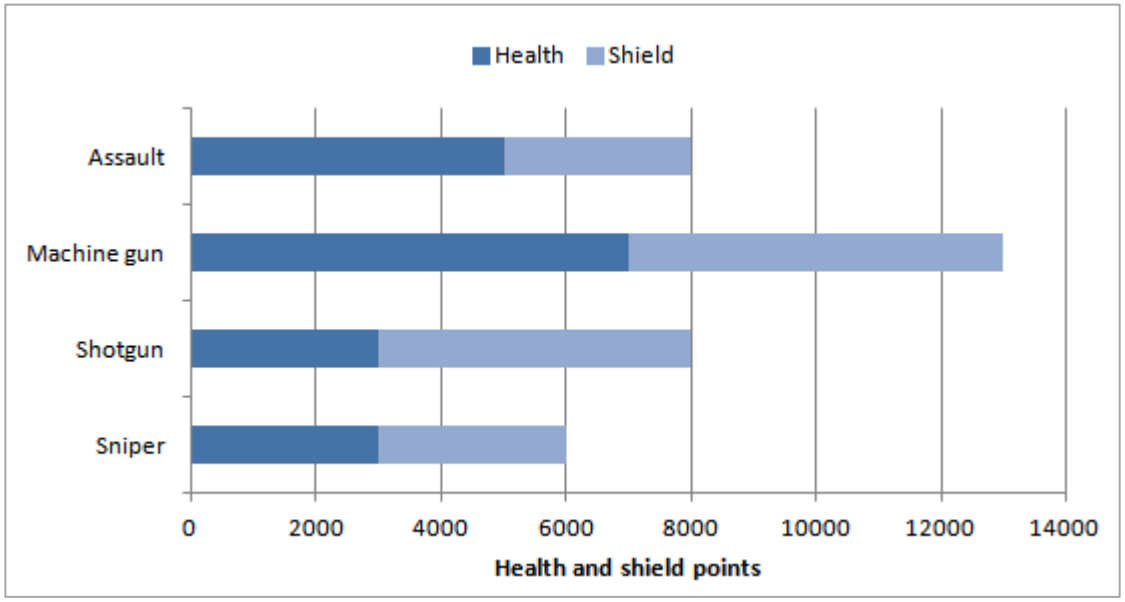


Figura 1.4. Gráfica de los puntos de salud y de escudo.

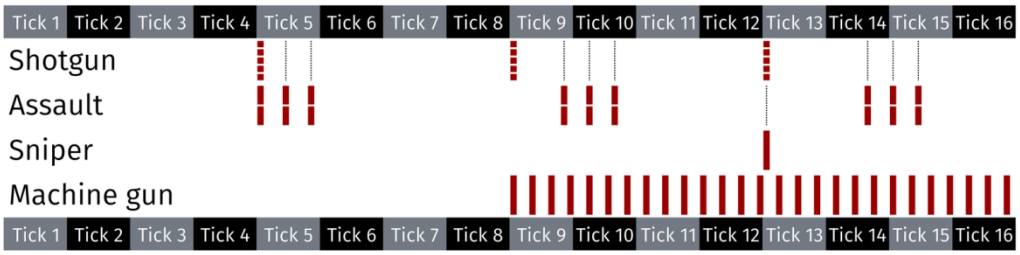


Figura 1.5. Gráfica de la rapidez para fijar el objetivo y de la velocidad de disparo.

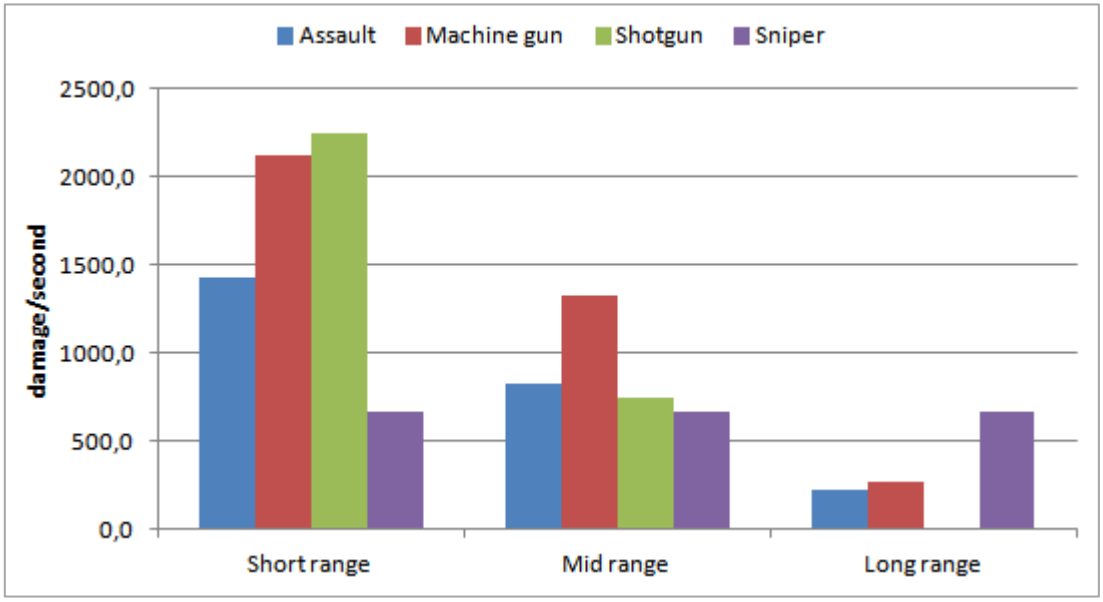


Figura 1.6. Gráfica de la potencia de fuego (daños causados a corto, medio, y largo alcance).

3.3. PROGRAMACIÓN DE LA INTELIGENCIA ARTIFICIAL.

Los robots de Gladiabots son autónomos y no podemos controlarlos directamente. Su funcionamiento obedece una **Inteligencia Artificial** (IA) que nosotros debemos programar. En un instante dado, un robot solo pueden ejecutar una de las acciones de su IA. La tarea de la IA es determinar qué acción debe

ejecutar el robot en la situación actual. Cada robot evalúa su IA de **izquierda a derecha** y de **arriba a abajo** (esto es, en contra del sentido de las agujas del reloj), en busca de la primera **rama válida** que les conduzca a una acción ejecutable (ver animación "Proceso de evaluación de una IA.gif").

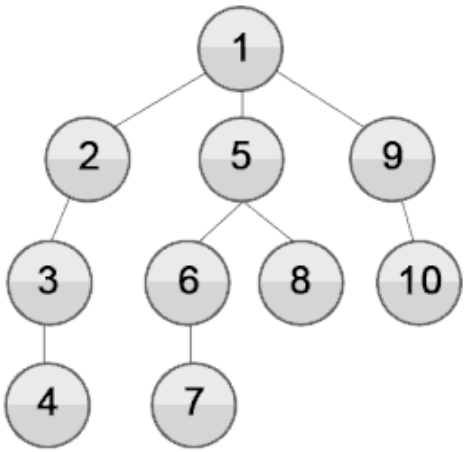


Figura 1.7. Proceso de evaluación de una IA.

La IA de un robot se escribe conectando distintos tipos de nodos mediante un editor gráfico, y adquiere una estructura en forma de árbol puesto del revés.

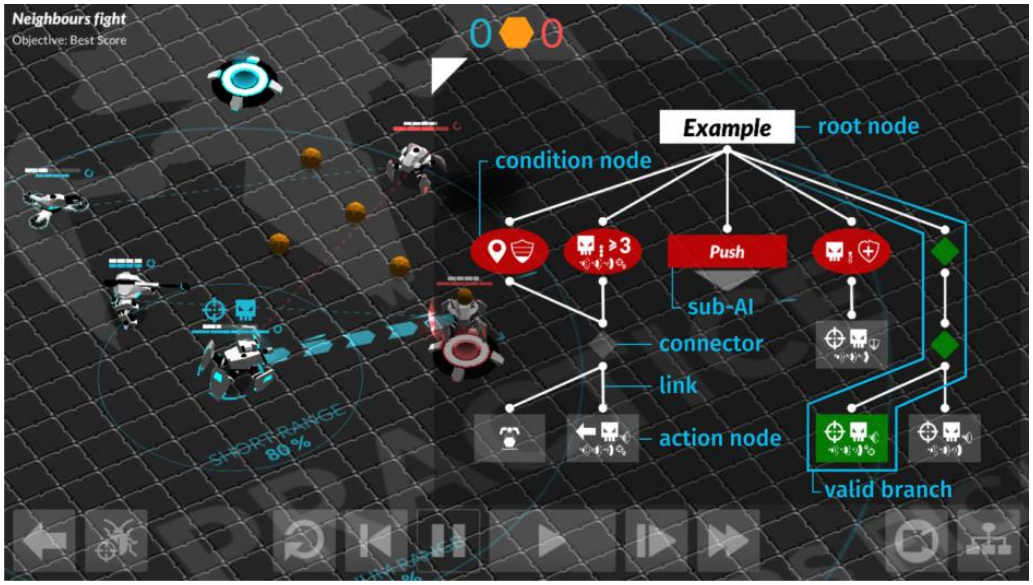


Figura 1.8. Los distintos tipos de nodos de una IA.

Los tipos de nodos que podemos utilizar para construir las IAs de nuestros robots son los siguientes (ver figura):

- **Nodo raíz**. Toda IA contiene exactamente un nodo raíz que define el punto en el que esa IA comienza a evaluarse. El resto de nodos que cuelgan del nodo raíz se evalúan de izquierda a derecha y de arriba abajo (en sentido contrario a las agujas del reloj). El nodo raíz también especifica el nombre de la IA.

- **Nodos de acción**. Si un nodo de acción rectangular se evalúa como válido durante el proceso de comprobación de una IA, el robot ejecuta inmediatamente la acción descrita por ese nodo. Si un robot es incapaz de ejecutar la acción descrita en el instante actual, ese nodo de acción se evalúa como inválido.

- **Nodos de condición**. Si un nodo de condición ovalado se evalúa como válido durante el proceso de comprobación de una IA, se pasan a evaluar todos los nodos que cuelgan bajo él (en sentido contrario a

las agujas del reloj). Un nodo de condición se evalúa como inválido si la condición descrita no se verifica en el instante actual.

- **Nodos conectores**. La única función de estos nodos romboidales es enlazar todos los nodos conectados a la parte superior con todos los nodos conectados a la parte inferior. Estos nodos simplemente sirven para simplificar y asear la apariencia de una IA.

- **Nodos sub–IA**. Estos nodos permiten que una IA puede reutilizar otra IA escrita previamente. Al llegar a uno de estos nodos sub-IA, el programa evalúa toda la IA contenida dentro del nodo, en busca de una rama válida. Cada sub-IA secundaria puede contener en su interior otros nodos sub-IAs.

- **Enlaces**. Los nodos de una IA se interconectan mediante enlaces. Un enlace no es más que una conexión desde la parte inferior de un nodo a la parte superior de otro nodo.

A modo de ejemplo, la figura 1.9 muestra la IA de uno de los robots del equipo azul. En esta sencilla IA, el robot empezaría evaluando la rama de la izquierda: Comprueba si tiene su escudo vacío, y en tal caso, huye del bot enemigo más cercano. Como esta rama no se verifica (porque nada más empezar el escudo está completo), el robot continúa evaluando la segunda rama, que consiste en atacar al bot enemigo que se localice a corta o media distancia. Como al principio no hay ningún bot enemigo dentro de ese rango de distancias esta rama tampoco es válida, y el robot evalúa la rama más a la derecha, que le ordena moverse hacia el bot enemigo más cercano. Ésta es la única rama que se verifica y que el robot ejecuta, de forma que el robot avanza hacia el enemigo más próximo. Mientras el robot avanza hacia sus enemigos la IA sigue evaluándose de izquierda a derecha hasta encontrar la primera rama válida, que tras haberse acercado es atacar al bot enemigo más próximo. En el caso de que el robot fuese atacado y detectase que su escudo ha quedado vacío, se verificaría la primera rama de su IA, y el robot huiría de los bots enemigos.

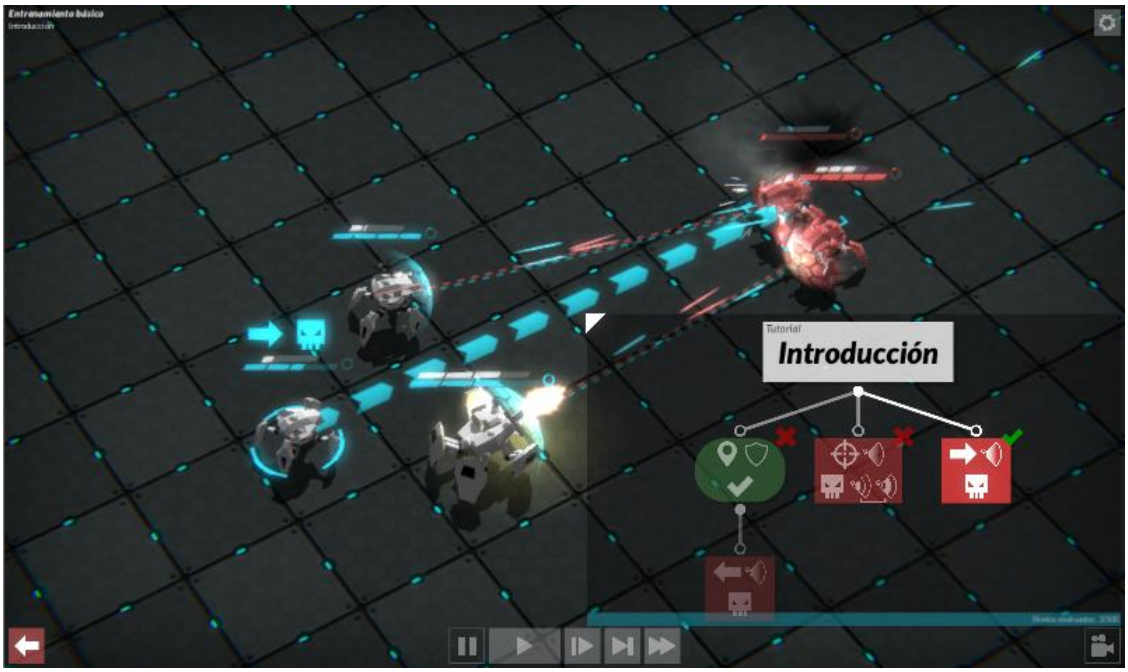


Figura 1.9. Sencillo ejemplo de IA.

3.4. INTERFAZ DE USUARIO.

La figura 1.10 muestra la interfaz de usuario de Gladiabots nada más entrar en el programa. No es una interfaz muy intuitiva, así que vale la pena explicar cada una de sus partes antes de ponernos a trabajar con ella. En la parte superior tenemos un menú de pestañas que nos permite acceder a las distintas zonas de Gladiabots:

- La pestaña **Inicio** sirve para mostrar la pantalla de inicio, donde aparecen los botones de "Continuar tutorial", "Unirse a la comunidad", "Visitar la wiki", y "Optimization pack DLC". Aquí también tenemos el botón de apagado para salir del programa.

Figura 1.10(a). Pantalla de inicio de Gladiabots.

- La pestaña **Solo** permite acceder al tutorial, a los modos de juego para un jugador (Colección, Dominación, y Eliminación), y a la sala de pruebas.

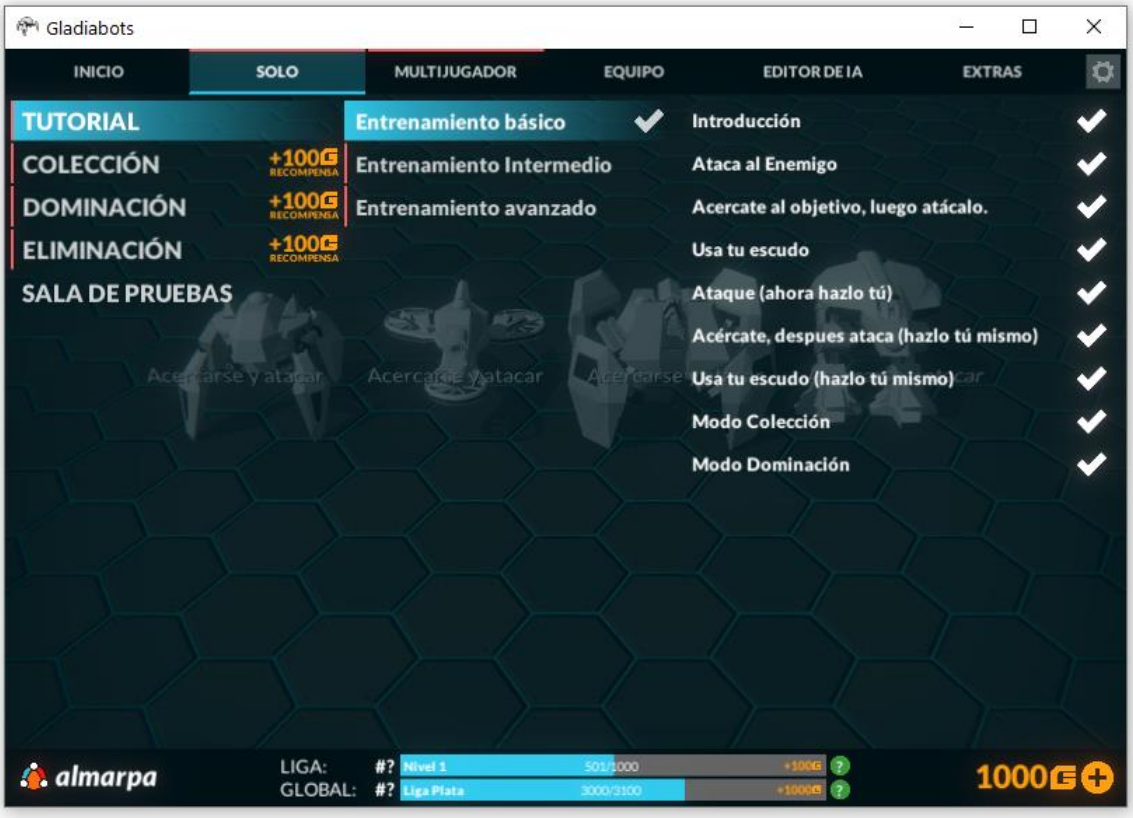


Figura 1.10(b). La pantalla Solo.

- La pestaña **Multijugador** accede a los modos de juego para varios jugadores (Competición, Torneo, Juego libre, y Partidas privadas), así como a nuestras estadísticas en este modo de juego.

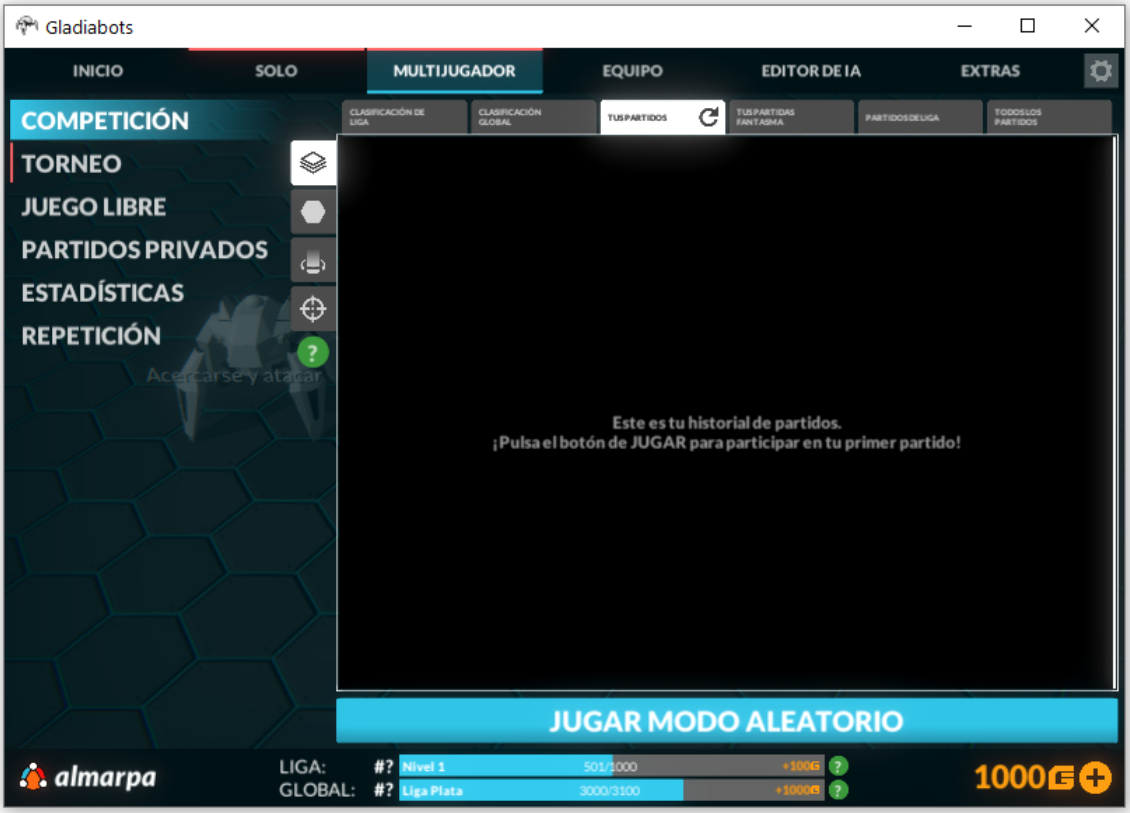


Figura 1.10(c). La pantalla Multijujador.

- La pestaña **Equipo** nos permite acceder a la **ventana de configuración** de los robots que constiyuyen nuestro equipo. En esta ventana podemos seleccionar la clase de robot de cada uno de los miembros de nuestro escuadrón (asalto, escopeta, francotirador, o ametralladora), asignarles una de las IAs que hemos escrito en el editor de IA, e incluso personalizar su aspecto (color, textura, etc.).

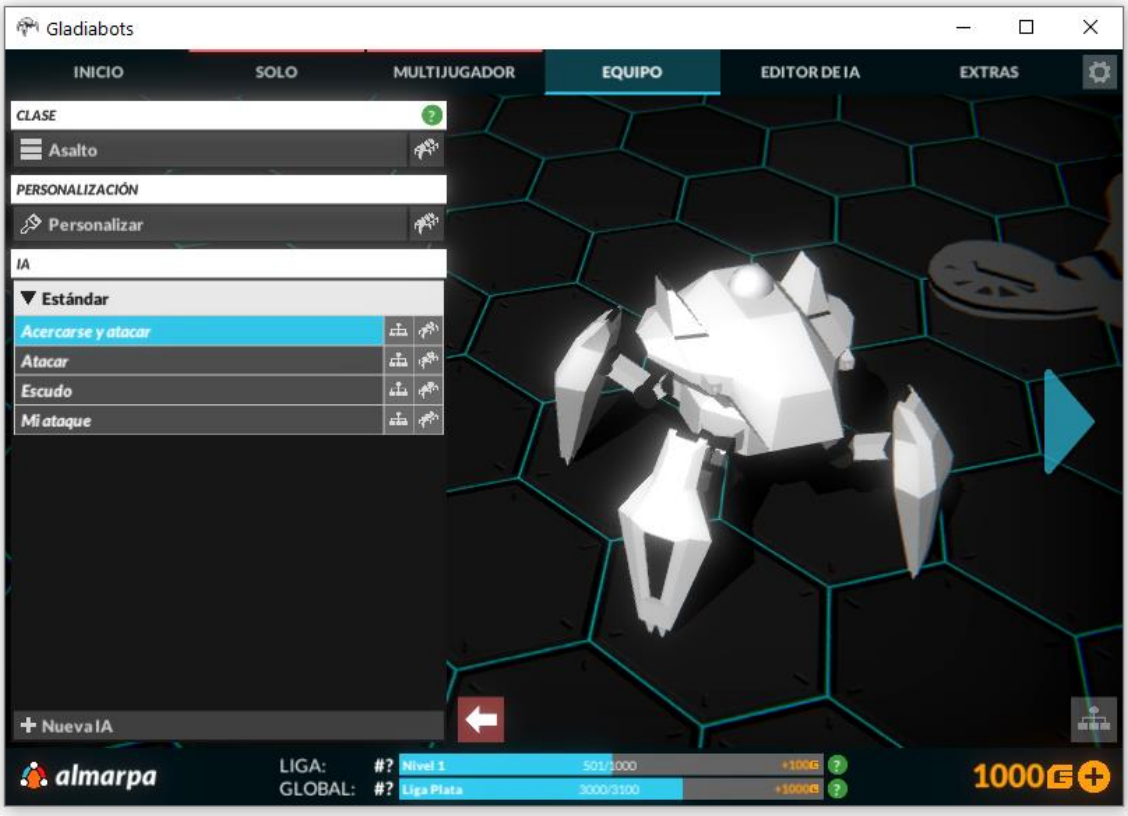


Figura 1.10(d). La pantalla de Equipo.

- La pestaña **Editor de IA** nos permite entrar en la ventana donde escribiremos las IAs de nuestros robots. Este editor nos permite guardar las IAs construidas, crear nuevas IAs, editar IAs ya existentes, importar IAs, etc.

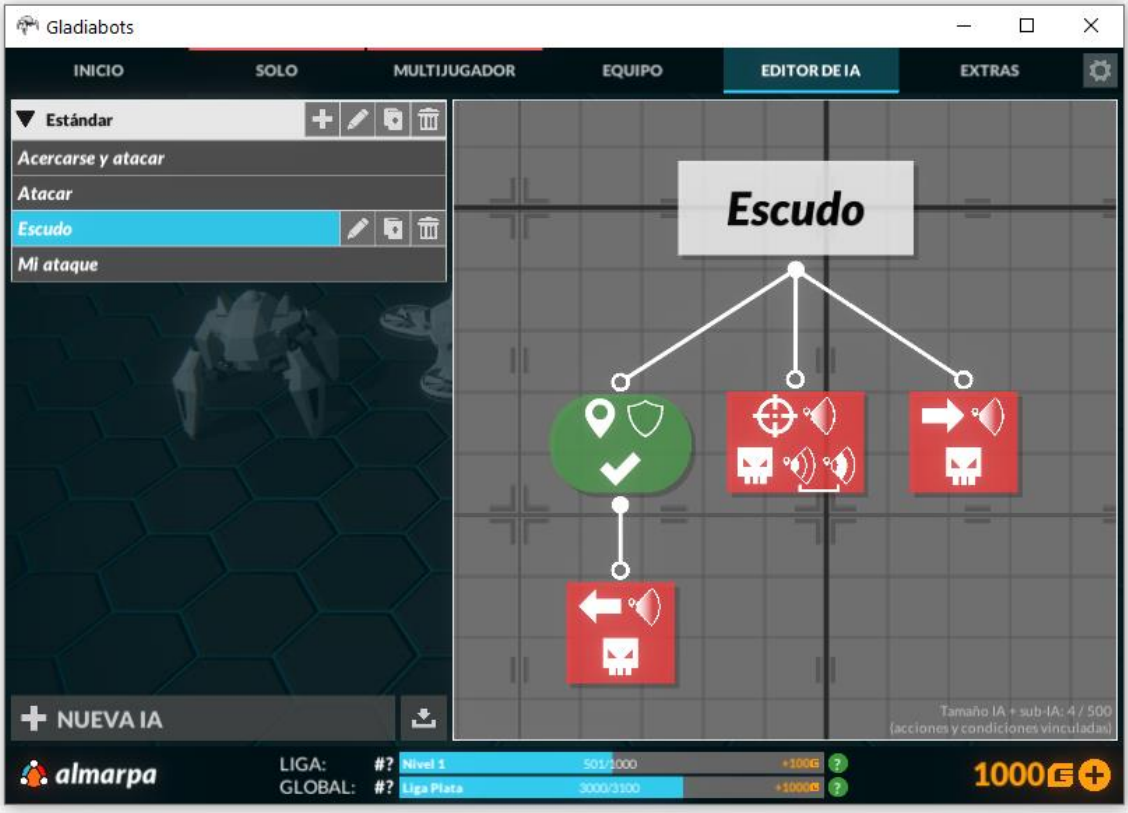


Figura 1.10(e). El editor de IAs.

- Por último, la pestaña **Extras** nos ofrece acceso a otras opciones secundarias, como por ejemplo, los enlaces a las distintas redes sociales en las que participa Gladiabots.

Arriba y a la derecha de la ventana de Gladiabots, el botón del engranaje nos permite acceder a los **ajustes**, como por ejemplo, el idioma, la calidad de los gráficos, la configuración del audio, la gestión de nuestra cuenta de Gladiabots, etc.

Abajo tenemos nuestra puntuación en las ligas de Gladiabots, así como nuestra puntuación global. A la izquierda de esta zona figura el nombre de usuario de nuestra cuenta (en mi caso, "almarpa"), y el botón para salir de Gladiabots.

3.5. EL TUTORIAL DE GLADIABOTS.

En la pestaña Solo, Gladiabots incluye un extenso **tutorial** que nos permite aprender a usar el programa paso a paso. Este tutorial consta de tres bloques: Entrenamiento básico, entrenamiento intermedio, y entrenamiento avanzado. Antes de empezar a jugar con Gladiabots, vamos a trabajar estos entrenamientos.

3.6. ENTRENAMIENTO BÁSICO.

En la ventana principal de Gladiabots pulsamos en la pestaña "Solo", y a continuación, seleccionamos la opción "Tutorial". A continuación, eligimos "Entrenamiento Básico", que consta de 9 lecciones. Vamos a empezar con la lección de "Introducción".

INTRODUCCIÓN.

Esta lección simplemente presenta el juego Gladiabots, los equipos combatientes, la IA que gobierna el comportamiento de los robots, y los botones para controlar la reproducción del combate. Cuando hayas completado la lección, pulsa en "Siguiente" para pasar a la siguiente lección.

ATACA AL ENEMIGO.

En esta lección aprenderemos a programar la IA de nuestro robot para atacar al bot enemigo. Aquí veremos cómo acceder al editor de IA desde la arena de combate, conoceremos el **nodo raíz** y los **nodos de acción**, descubiremos el significado de nuestro primer nodo de acción, y aprenderemos a enlazar nodos para crear ramas. Una vez definida la IA del robot, volvemos a la arena de combate para ejecutar la IA recién escrita y vencer a nuestro enemigo (que en esta lección carece de una IA propia).

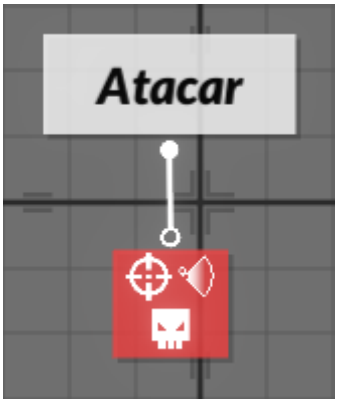


Figura 2.1. Atacar al enemigo más cercano.

ACÉRCATE AL OBJETIVO, Y LUEGO ATÁCALO.

Nada más empezar la lección vemos que nuestro robot está programado para atacar a enemigos localizados a corta o media distancia, ver figura 2.2(a). Pero al ejecutar el programa descubrimos que el robot no hace nada: En efecto, el enemigo se localiza a gran distancia, y la IA no encuentra ninguna rama válida en el programa de control.

Vamos a editar la IA del robot para añadirle otra rama que le permita acercarse al enemigo más cercano. La figura 2.2(b) muestra cómo queda el programa tras añadir esta nueva rama *a la izquierda*. Sin embargo, esta IA también es errónea: Recordemos que aunque los nodos *no* se ejecutan de izquierda a derecha, *sí* que se evalúan de izquierda a derecha, en busca de la primera rama que resulte ser válida. Como en este programa la acción de acercarse al enemigo está en la rama de la izquierda, siempre es la primera en evaluarse, y siempre es válida, por lo que el robot solo se acerca al enemigo pero nunca lo ataca.

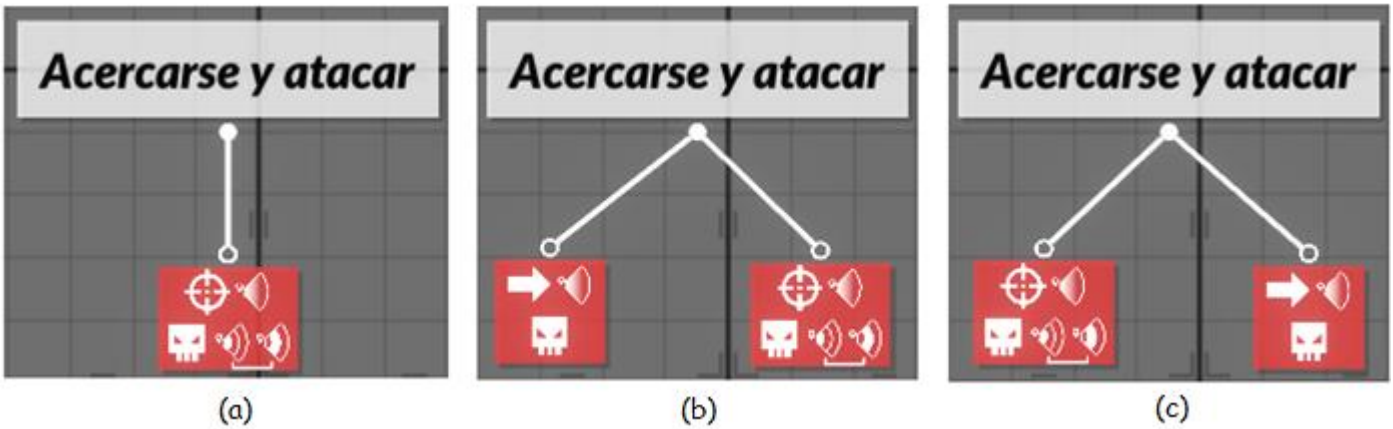


Figura 2.2. Moverse al enemigo más carcano, y atacar al enemigo a corta o media distancia.

Para corregir esta IA debemos cambiar de lugar la acción "Moverse hacia el bot enemigo más cercano" para ubicarla a la derecha, dejando la acción "Atacar al bot enemigo a corta o media distancia" a la izquierda, ver figura 2.2(c). Al ejecutar esta IA, veremos que nuestro robot primero se acerca al enemigo (porque la única rama válida del programa es la de la derecha). Pero en cuanto se acerca a media distancia, se verifica la rama de la izquierda, y nuestro robot comenzará a atacar al enemigo hasta conseguir destruirlo.

USA TU ESCUDO.

Recordemos que nuestros robots van equipados con un escudo (representado con una barra blanca). El escudo absorbe el daño antes de que la salud (representada con una barra azul) se vea afectada. Si el escudo no absorbe daño durante un tiempo, se regenera automáticamente.

En esta lección, nuestro robot comienza con la IA que escribimos en la lección previa (acercarse y atacar). Pero en este caso debe combatir contra dos bots enemigos que han sido programados para contraatacar. Si no hacemos nada más, nuestro robot pierde su escudo rápidamente, y con dos bots atacándolo simultáneamente, cae derrotado tras perder toda su salud.

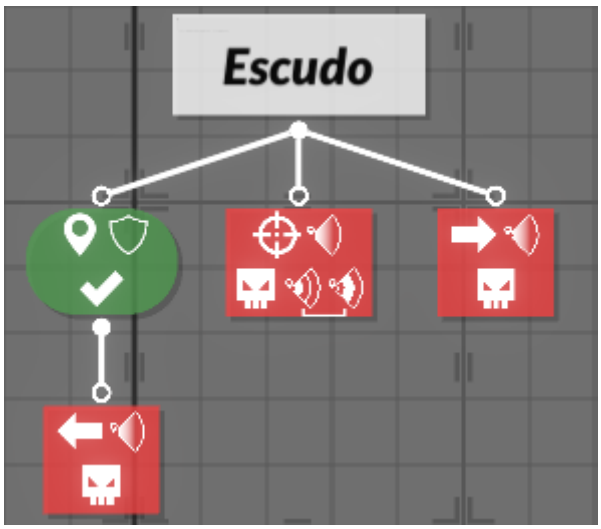


Figura 2.3. Escudo.

Vamos a modificar la IA para huir del enemigo si nuestro escudo se queda vacío. Para ello, usaremos un **nodo de condición**. Bajo los nodos de condición podemos conectar otros nodos, que solo se comprobarán si la condición es válida.

La figura 2.3 muestra la nueva IA de nuestro robot. Al comenzar el combate, la primera rama válida es la de la derecha, y nuestro robot se aproxima al bot enemigo más cercano. Al acercarse a media distancia del enemigo se verifica la rama del centro, y nuestro robot comienza a atacar. Pero en este caso los enemigos se defienden, y también nos atacan. Sin embargo, cuando el robot detecta que su escudo está vacío se verifica el nodo condicional de la izquierda, la IA ejecuta el nodo de acción que cuelga bajo él, y el robot huye de los bots enemigos para regenerar su escudo y volver a la carga.

ATAQUE (AHORA HAZLO TÚ).

En esta lección aprenderemos a crear nuestras propios IAs. Para ello, vamos a programar a nuestro robot para que ataque a un bot enemigo que carece de IA.

Para crear un nuevo nodo que cuelque del nodo raíz, basta con arrastrar un enlace desde la parte inferior del nodo raíz (ver figura 2.4). Al soltar el ratón, Gladiabots nos permite elegir el tipo de nodo a conectar (acción, condición, conector, o sub-IA). En esta lección, Gladiabots solo nos permite añadir un nodo de acción.

Figura 2.4. Crear un nuevo nodo.

Al crear un nuevo nodo de acción, se abre una ventana que nos permite configurar todos los detalles de este nodo, ver las figuras de la serie 2.5.

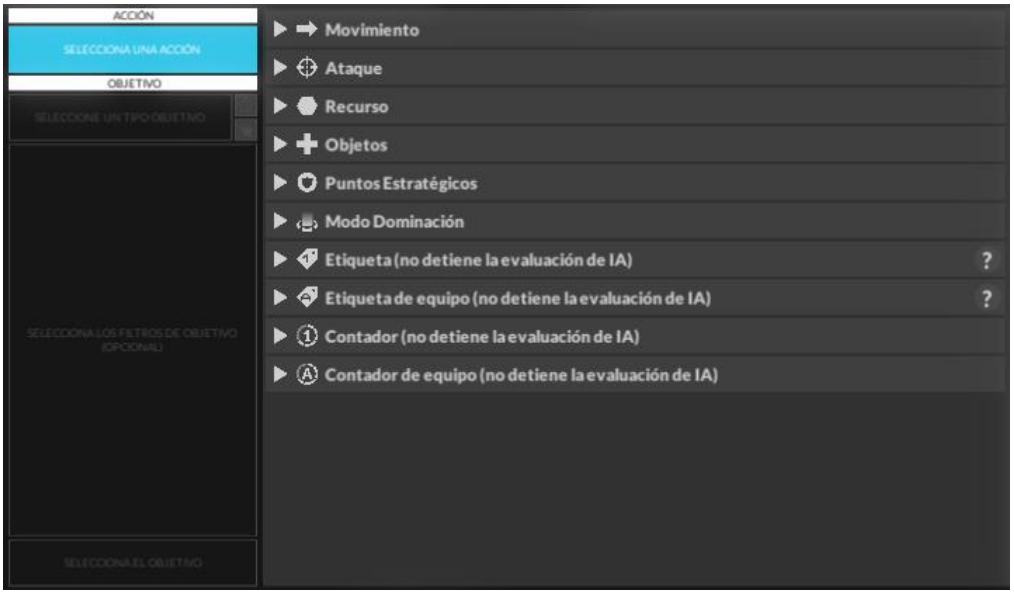


Figura 2.5(a). Crear un nuevo nodo de acción (1).

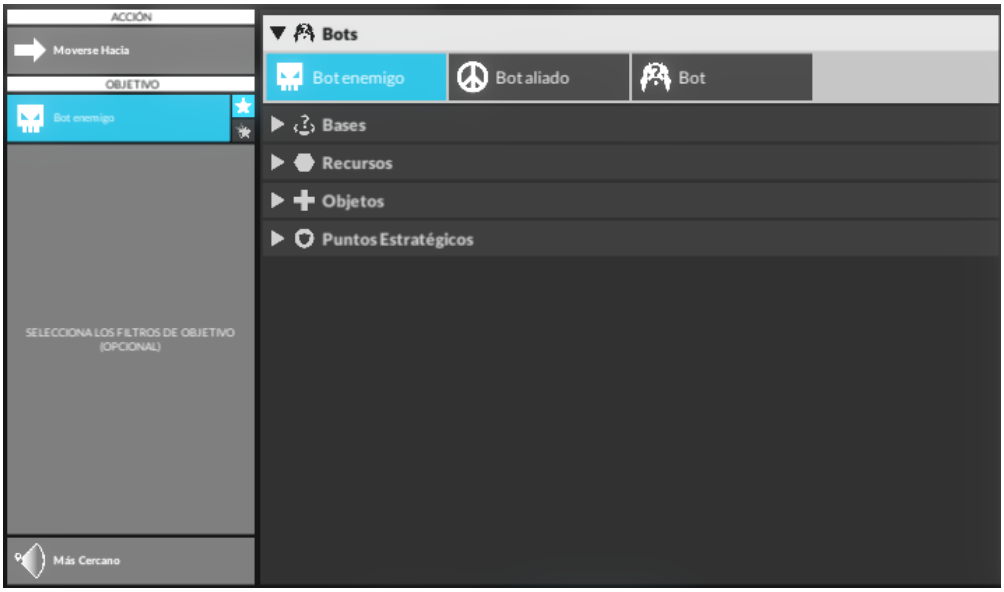


Figura 2.5(b). Crear un nuevo nodo de acción (2).

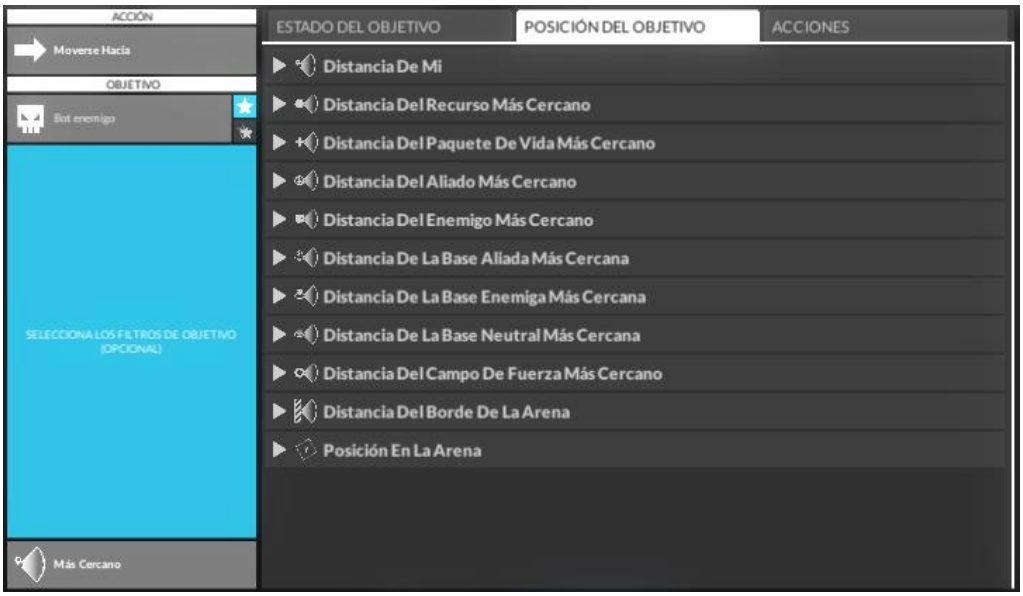


Figura 2.5(c). Crear un nuevo nodo de acción (3).

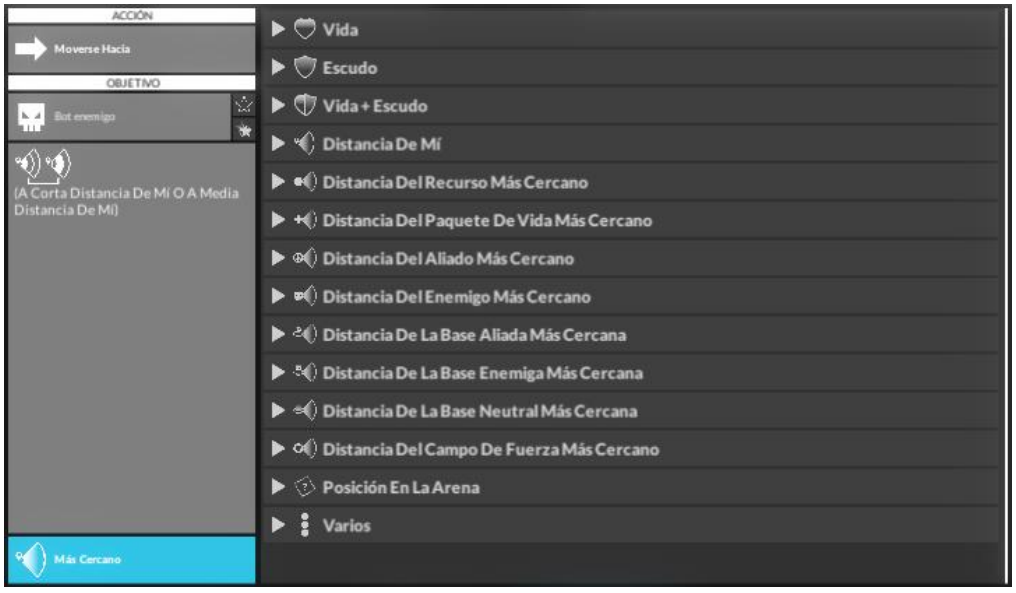


Figura 2.5(d). Crear un nuevo nodo de acción (4).

Primero elegimos el **tipo de acción**: Movimiento, ataque, recursos, objetos, puntos estratégicos, etc. Luego seleccionamos la entidad objetivo para esta acción (**tipo de objetivo**), que puede ser un bot enemigo, un bot aliado, un bot genérico, una base (neutral, enemiga, aliada, o genérica), un recurso, un objeto, etc. A continuación podemos elegir (opcionalmente) una serie de filtros para mejorar la selección del objetivo (**filtros de objetivo**), en función del estado del objetivo (clase de bot, nivel de vida, nivel de escudo) o de la posición del objetivo (distancia de mí, distancia a la base más cercana, distancia al borde de la arena, etc.). Finalmente, seleccionamos algunas características más para el objetivo de esa acción (**objetivo**), como por ejemplo, el objetivo de vida más débil, el del escudo más débil, el más cercano de mí, el más lejano de mí, el más lejano a una base, el más cercano a un recurso, etc. En nuestro caso, creamos un nodo de acción para atacar al bot enemigo más cercano de mí a corta o media distancia, ver figura 2.6.

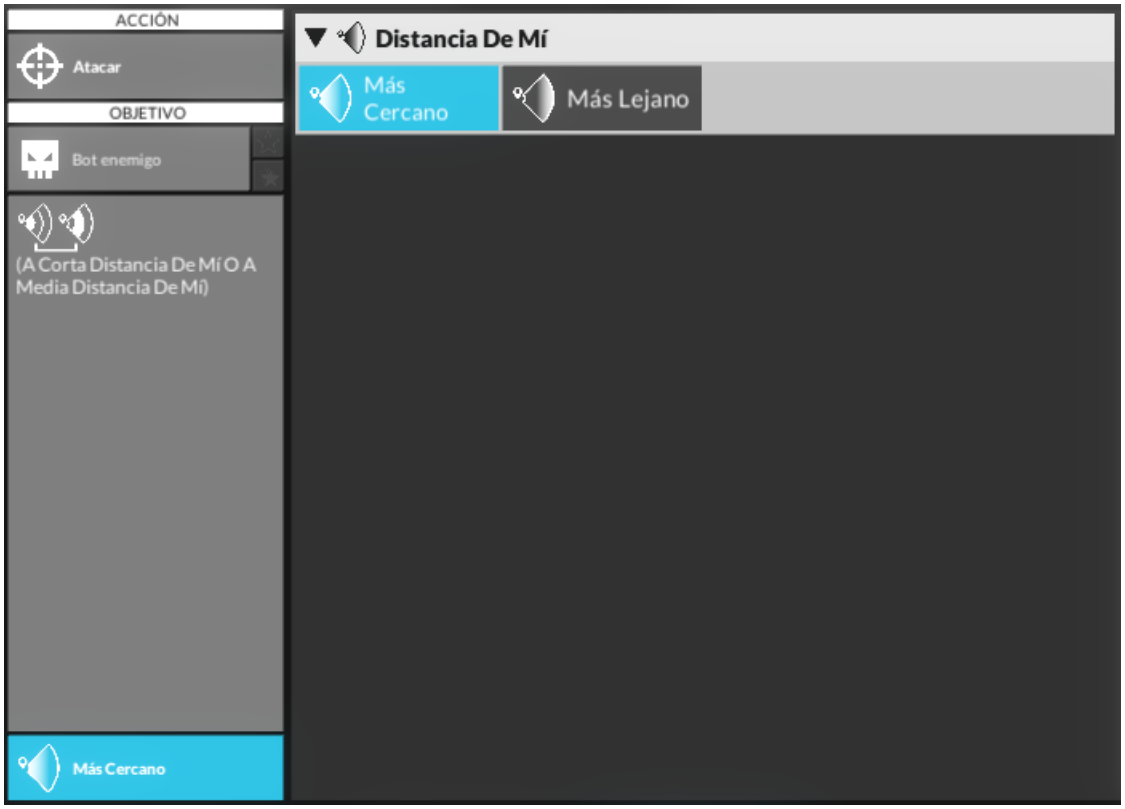


Figura 2.6. Atacar al bot enemigo más cercano de mí a corta o media distancia.

Al ejecutar esta IA veremos que nuestro robot comienza a atacar al bot enemigo, porque se localiza media distancia de nosotros.

Figura 2.7. Mi IA para atacar al bot enemigo más cercano de mí a corta o media distancia.

ACÉRCATE, Y DESPUÉS ATACA (HAZLO TÚ MISMO).

En esta lección vamos a programar la IA de nuestro robot para hacer que se acerque al enemigo más cercano, y para que pase a atacarlo cuando quede a corta o media distancia de nosotros. Cuidado: Debemos

recordar dónde colocar las acciones de moverse y atacar para que la IA las evalúe en el orden correcto, de forma que el programa funcione adecuadamente.

USA TU ESCUDO (HAZLO TÚ MISMO).

En esta ocasión vamos a programar al robot para que huya cuando su escudo quede vacío. Esta lección comienza con la IA de la lección previa ya programada, y nuestra misión es completarla. De nuevo, arrastrando y soltando un enlace desde la parte baja del nodo raíz, se abre el menú para crear un nuevo nodo. En este caso necesitamos un nodo de condición. A continuación, se abre la ventana para configurar este nodo de condición. Aquí seleccionamos el objetivo para esa condición (elegimos la opción "Yo mismo"), especificamos los filtros para establecer restricciones a ese objetivo ("Escudo del objetivo al 0%"), y para terminar, indicamos el tipo de condición (seleccionamos "Existe" ☑ para indicar que queremos que se cumpla esa condición. Si quisiéramos que no se cumpla esa condición, seleccionaríamos "No existe" ☒).

Una vez creado el nodo de condición, arrastramos un enlace desde su parte inferior para conectarlo a un nodo de acción para huir de los enemigos. Para configurar este nodo de acción, seleccionamos la acción de movimiento "Huir". Como objetivo de esta acción, especificamos que queremos huir de un bot enemigo. Dejamos los filtros sin especificar, y finalmente indicamos que queremos huir del bot enemigo más cercano a nosotros.

MODO COLECCIÓN.

En modo Colección nuestro objetivo es recoger los recursos y llevarlos a nuestra base para anotar puntos.

En esta lección la arena de combate incluye dos recursos y dos bases (una enemiga y otra aliada). Nuestro robot comienza con la IA mostrada en la figura 2.8. De izquierda a derecha, las ramas especifican lo siguiente: (1) Atacar al bot enemigo más cercano que esté a corta o media distancia. (2) Si nuestro robot transporta un recurso, anotar el recurso en la base aliada más cercana. (3) Recoger el recurso más cercano. (4) Moverse hacia el bot enemigo más cercano.

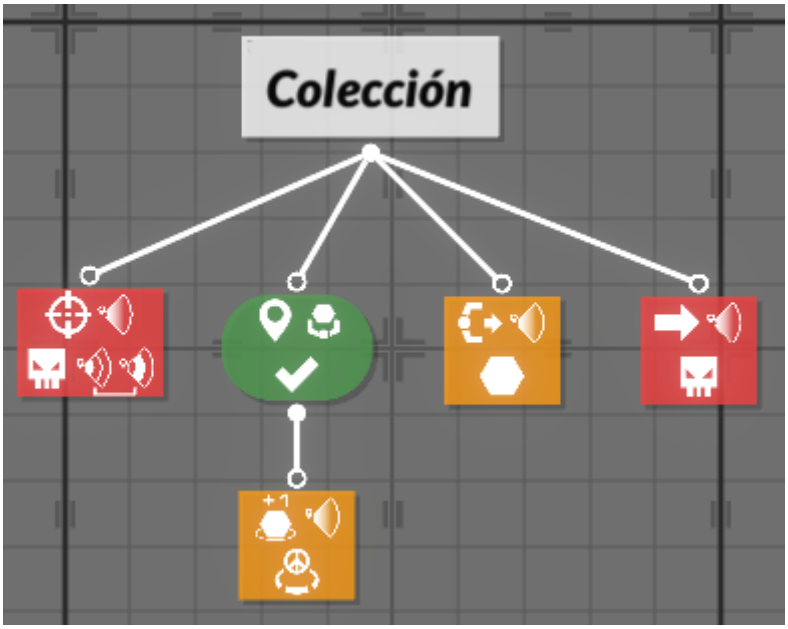


Figura 2.8. Ejemplo de IA para el modo Colección.

Al comenzar la partida, la primera rama que se verifica es la tercera, y el robot avanza hacia el recurso más cercano. El bot enemigo hace lo mismo, y en el momento en el que ambos bots se aproximan el uno al otro, pasa a verificarse la primera rama, y los robots comienzan a atacarse mutuamente. Cuando nuestro robot consigue acabar con el robot enemigo (algo que siempre ocurre, porque el bot enemigo carece de

escudo), vuelve a verificarse la tercera rama, y se dirige hacia el recurso más próximo. Al recogerlo se verifica la segunda rama, y el robot lo lleva a nuestra base para anotar un punto. Cuando el robot recoge el segundo recurso y lo anota en nuestra base, la partida termina con victoria.

Un par de comentarios importantes: Notar que nuestro robot se vuelve más lento cuando transporta un recurso. Además, tampoco es capaz de atacar cuando se encuentra en este estado. Debemos tener este hecho muy en cuenta cuando diseñemos las IAs de los robots en este modo.

MODO DOMINACIÓN.

Recordemos que en el modo Dominación nuestra misión es capturar y retener bases. Una vez que una base es capturada, pasará a anotar puntos con el tiempo.

En esta lección, la arena de combate contiene dos bases neutrales, y nuestro robot se enfrenta a otro bot enemigo. La figura muestra la IA con la que comienza nuestro robot. Esta IA consta de cuatro ramas, que de izquierda a derecha, especifican lo siguiente: (1) Atacar al bot enemigo más cercano que esté a corta o media distancia. (2) Capturar la base neutral más cercana. (3) Capturar la base enemiga más cercana. (4) Moverse hacia el bot enemigo más cercano.

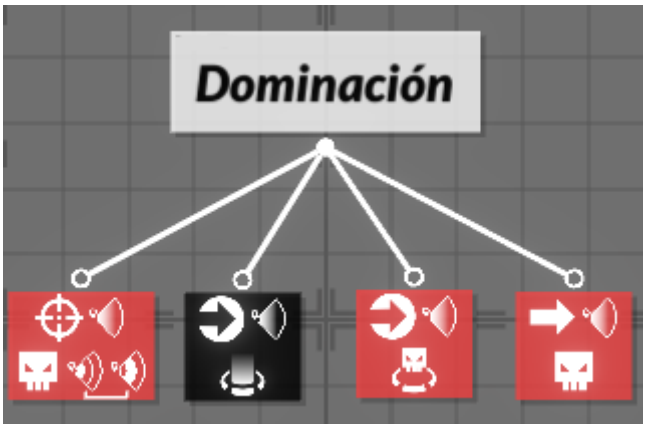


Figura 2.9. Ejemplo de IA para el modo Dominación.

Al comenzar la partida, la primera rama que se verifica es la segunda, y nuestro robot se acerca a la base neutral más cercana. El bot enemigo hace lo mismo. Nuestro robot captura una base cuando entra en la zona de corto alcance. Cuantos más bots lleguen a la base, más rápida será la captura. Una vez capturada una base, comenzará a anotar puntos. Además, cuantos más bots permanezcan en la base, más rápidamente anotará puntos esa base. Después de que ambos robots (el nuestro y el enemigo) hayan capturado sendas bases, se verifica la tercera rama de la IA, y nuestro robot avanzará hacia la base enemiga para capturarla. Pero en el momento que se acerca a media distancia del bot enemigo ubicado en la base enemiga, nuestro robot empieza a atacarlo. Como el bot enemigo no dispone de escudo, nuestro robot consigue derrotarlo. A continuación se verifica la tercera rama, y el robot avanza hacia la base enemiga para capturarla. Para capturar una base enemiga, primero debemos neutralizarla, y luego capturarla, lo cual es un proceso más largo que el de capturar una base neutral. Una vez capturada, la base empieza a anotar puntos hasta que se cumple el tiempo límite, y ganamos la partida.

PROBAMOS LO QUE HEMOS APRENDIDO (1).

Con esta última lección, hemos completado el entrenamiento básico. Antes de pasar al entrenamiento intermedio, vamos a probar lo que hemos aprendido hasta ahora. Para ello, acudimos a la pestaña "Solo", seleccionamos la opción "Colección", y eligimos el Nivel 1 del Capítulo 1. Vamos a jugar nuestra primera partida a Gladiabots en modo Colección.

En este primer nivel del capítulo 1 dos de nuestros robots (por defecto, dos robots de asalto) se enfrentan a otros dos bots enemigos por el control de tres recursos. La arena de combate dispone de dos bases: Una base enemiga y una base aliada.

Aljugar en modo Individual (Solo), nuestras tareas son básicamente dos: (1) Escribir la(s) IA(s) que controlarán el comportamiento de nuestros robots, y (2) configurar nuestro equipo de robots (esto es, elegir la clase de cada uno de los robots de nuestro escuadrón, y asignarles una de las IAs que hayamos construido).

Para empezar, acudimos al editor de IAs de Gladiaboots. Como vemos, Gladiabots incluye por instalación un pack con algunas IAs preconstruidas. Primero pulsamos el botón + para crear una nueva IA vacía, le ponemos un nombre, y la dejamos vacía. A continuación, borramos todas las demás IAs (esto es, las IAs que Gladiabots trae por defecto), ya que son bastente ineficientes, y porque nosotros vamos a construir nuestras propias IAs partiendo de cero. Una vez hecho esto, en la lista de IAs disponibles solo quedará la IA vacía que nosotros creamos previamente. Vamos a darle contenido, añadiendo los bloques de acción y de consdición que creamos necesarios, y en el orden de prioridad (de iquierda a derecha) que consideremos conveniente. Por ser nuestra primera partida en Gladiabots, empezaremos construyendo una sola IA para controlar a nuestros dos robots. Esto no tiene por qué ser así, y podríamos escribir dos IAs distintas, para controlar de forma diferente a cada uno de los robots. Por el momento, nos conformaremos con controlar a ambos con la misma IA.

A continuación, configuramos nuestro equipo de robots. Notar que, como en esta misión sólo disponemos de dos robots, ambos están obligados a ser de la clase Asalto. Si intentamos reconfigurar nuestro equipo para que uno de los robots sea de otra clase, el programa no nos lo permitirá. (Recordar que el número total de robots de las clases Escopeta, Ametralladora, y Francotirador solo pueden constituir el 35% de nuestro escuadrón). Por último, asignamos la IA que hemos constrido antes a los dos robots.

Vamos a probar la IA que hemos construido. Volvemos a la arena de combate, y pulsamos en el botón "Listo" para comenzar el combate. Si perdemos la partida es porque nuestra IA no es adecuada para el escenario actual, y deberemos refinarla hasta conseguir ganar la batalla. Cuando consigamos vencer, hacemos una captura de pantalla de la IA, y la guardamos como imagen con el nombre "C11.jpg" (**C**olección Capítulo **1** Nivel **1**). Avisamos al profesor para que tome nota del resultado, del tanteo, del número de robots que hemos perdido en la batalla, etc.

Una vez ganada la primera batalla, continuamos con el Nivel 2. Para empezar, probamos con la misma IA que utilizamos para superar el Nivel 1. Probablemente fracasaremos. Intentamos refinar esa IA para completar el Nivel 2, hasta donde nuestros actuales conocimientos nos lo permitan. No os pangáis nerviosos si no conseguís pasar de este segundo nivel, porque todavía no hemos completado el tutorial y nos quedan muchas cosas por aprender. De hecho, vamos dejar el modo "Solo" por el momento, y a continuar con el entrenamiento intermedio.

3.7. ENTRENAMIENTO INTERMEDIO.

El "Entrenamiento Intermedio" también consta de 9 lecciones, y aquí comenzaremos con la lección "Si A o B".

SI A O B.

En esta leccón necesitamos comprobar si el bot enemigo se sitúa a corta o a media distancia. El aspecto de una IA con una comprobación del tipo "Si A o B" es el mostrado por la figura 2.10. Las IAs de este tipo comprueban la condición A, y si no se verifica, a continuación comprueban la condición B. Si se verifica

cualquiera de las dos condiciones, ejecutan la acción que cuelga bajo ellas (el rectángulo vacío de la figura 2.10).

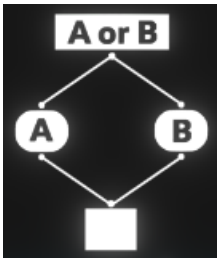


Figura 2.10. Estructura del tipo "Si A o B".

En esta lección, nuestro robot comienza con la IA de la figura 2.11, con la que comprueba si el enemigo está a corta distancia (nodo de la izquierda) o a media distancia (nodo de la derecha), y si se cumple cualquiera de esas dos condiciones, ataca a ese enemigo.

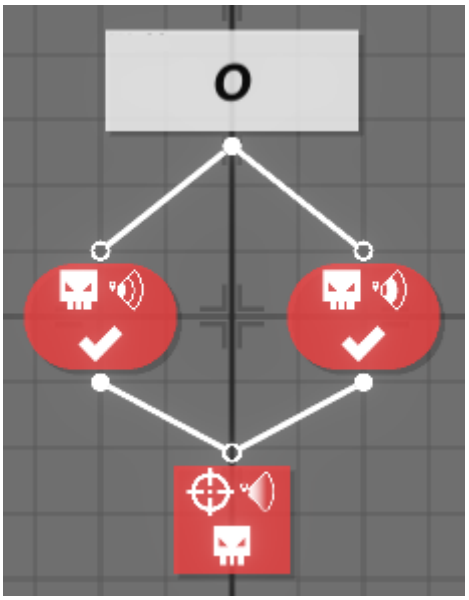


Figura 2.11. Ejemplo de IA con una estructura del tipo "Si A o B".

En este caso, el combate dura poco: El bot enemigo comienza avanzando hacia nosotros, y en cuanto entra dentro del rango de media distancia, nuestro robot comienza a atacarlo. Como el bot enemigo no dispone de escudo, y está programado para atacar únicamente cuando se encuentra a corta distancia de nosotros, nuestro robot consigue derrotarlo con facilidad. (De hecho, la IA del bot enemigo es un ejemplo de IA mal programada, porque es altamente inefectiva. De ser la IA de nuestro robot, habría que corregirla y refinarla).

SI A Y B.

Para cumplir con la misión de esta lección tenemos un límite de tiempo de 25 segundos. En este caso, vamos a enfrentarnos a dos bots enemigos con un único robot. En situaciones como ésta, no siempre es necesario huir cuando nuestro escudo está bajo; también podemos consultar el número de bots enemigos restantes. Para ello, vamos a emplear una estructura "Si A y B".

La figura 2.12 muestra la apariencia general de una estructura "Si A y B". En este tipo de estructuras exigimos que se verifiquen las condiciones A y B simultáneamente para que pueda ejecutarse la acción que cuelga bajo ellas.

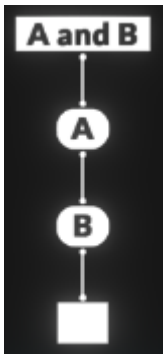


Figura 2.12. Estructura del tipo "Si A y B".

La figura 2.13 muestra un ejemplo de IA que utiliza una estructura "Si A y B". Esta IA contiene tres ramas, que de izquierda a derecha, son las siguientes: (1) Comprobamos si nuestro escudo está vacío y si el número de bots enemigos es mayor o igual que dos, y si se cumplen ambas condiciones, hacemos que el robot huya del bot enemigo más cercano. (2) Atacamos al bot enemigo más cercano a corta o media distancia. (3) Nos movemos hacia el bot enemigo más cercano.

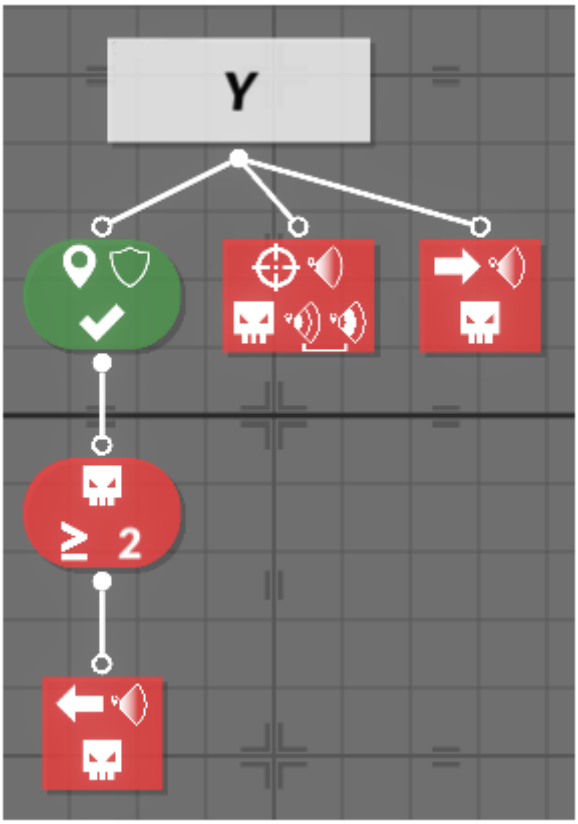


Figura 2.13. Ejemplo de IA con una estructura del tipo "Si A y B".

Con esta IA nuestro robot comienza combatiendo contra los dos bots enemigos (que en esta lección carecen de escudo). Inicialmente, y como el número de bots enemigos es dos, nuestro robot se aleja cada vez que se queda sin escudo. Pero tras regenerar el escudo, vuelve a la carga y continúa luchando hasta derrotar a uno de los bots. En ese momento, si su escudo se queda vacío ya no huye, sino que sigue atacando al bot que queda hasta conseguir acabar con él y ganar la batalla.

TRABAJO EN EQUIPO.

En Gladiabots, cada robot de nuestro escuadrón puede usar su propia IA personalizada, lo que nos permite utilizar estrategias de equipo. En esta lección jugaremos una partida en modo Colector en una arena de combate con una base neutral y dos recursos. Nuestro equipo consta de dos robots y el equipo rival de tres bots enemigos (sin escudo).

En el centro de la parte inferior de la pantalla tenemos el botón para acceder a la ventana de **configuración del equipo**, donde podemos probar diferentes combinaciones de IAs para completar exitosamente nuestra misión. En esta lección, nuestros robots tienen a su disposición dos IAs preconstruidas, llamadas Atacante y Recolector (ver figura 2.14), cada una de las cuales vamos a asignar a uno de los robots de nuestro equipo. De izquierda a derecha, la IA Atacante sirve para atacar al bot enemigo más cercano a corta o media distancia, y para acercarse al bot enemigo más cercano. La IA Recolector nos permite anotar el recurso transportado en la base aliada más cercana, y recoger el recurso más cercano.

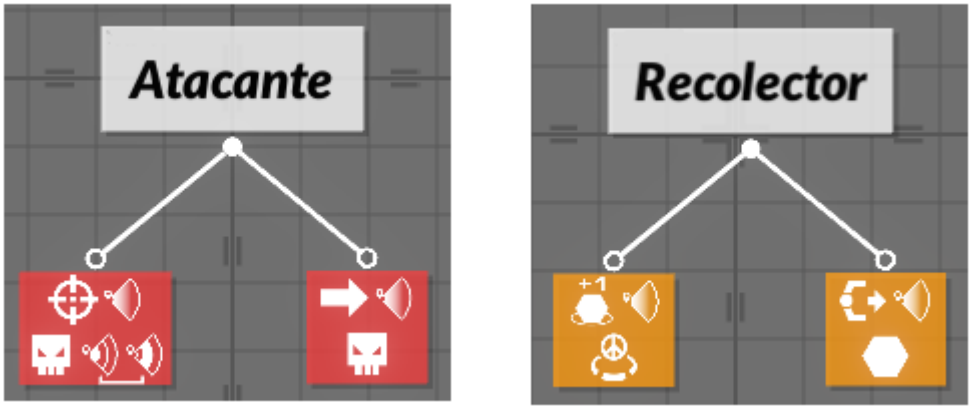


Figura 2.14. Las IAs Atacante y Recolector.

Al lanzar la partida, el robot con la IA Recolector se dirige hacia el recurso más cercano, y el robot con la IA Atacante hacia el bot enemigo más próximo. Mientras uno de nuestros robots se ocupa de recoger y anotar recursos, el otro se encarga de destruir a los enemigos. Esta estrategia en equipo nos permite ganar la partida.

ESCOPETA.

Gladiabots cuenta con varias clases de bots, cada una con sus fortalezas y debilidaades. La clase **escopeta** tiene poca vida, pero se mueve muy rápido (excepto cuando transporta un recurso), y causa un daño devastador a corto alcance. Como la escopeta es más eficaz a corta distancia, vamos a escribir una IA que le haga esperar a que el enemigo esté lo suficientemente cerca para atacarlo.

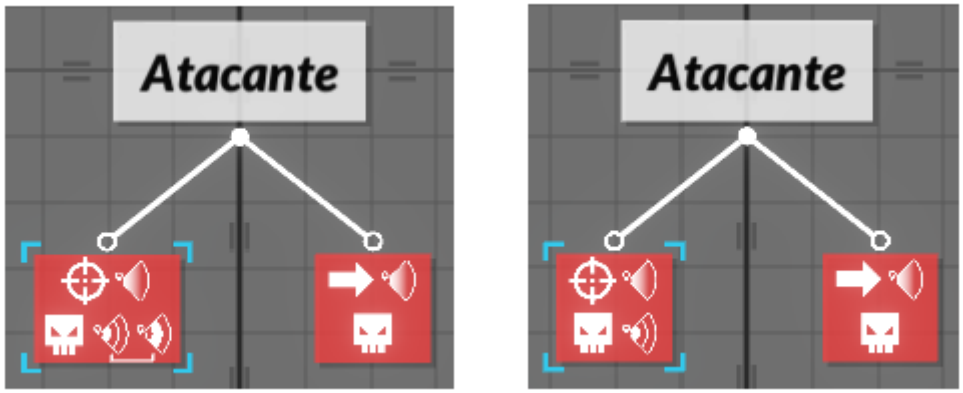


Figura 2.15. IA para un robot de la clase escopeta.

En esta lección comenzamos con la IA mostrada a la izquierda de la figura 2.15. Vamos a editar el bloque de la izquierda para hacer que la escopeta solo ataque a corta distancia. Para ello, seleccionamos el bloque de la izquierda, y pulsamos en el botón de "Editar el nodo seleccionado" (o simplemente, volvemos a pinchar sobre el bloque seleccionado) para acceder a la ventana de edición del nodo. Aquí vamos a combar los filtros de objetivo para que el robot sólo ataque a los bots enemigos a corta distancia. Tras modificar el nodo, la IA debería quedar como vemos en la parta derecha de la figura.

Con esta IA, nuestro robot no comienza a atacar hasta situarse a corta distancia del enemigo. Debido a su mayor potencia de fuego, nuestro robot termina derrotando al bot enemigo y se hace con la victoria.

FRANCOTIRADOR.

La clase **francotirador** se mueve lentamente, y tiene poco escudo y poca vida. Sin embargo, nunca falla sus disparos a cualquier distancia (excepto fuera de alcance), e inflinge mucho daño a su objetivo. Como el francotirador es más eficaz a largo alcance, vamos a escribir una IA que le haga atacar antes de que el enemigo se acerque.

En esta lección comenzamos con la IA mostrada a la izquierda de la figura 2.16. De nuevo, vamos a editar el bloque de la izquierda para hacer que el francotirador también ataque a gran distancia. La IA debería quedar como muestra la parte derecha de la figura:

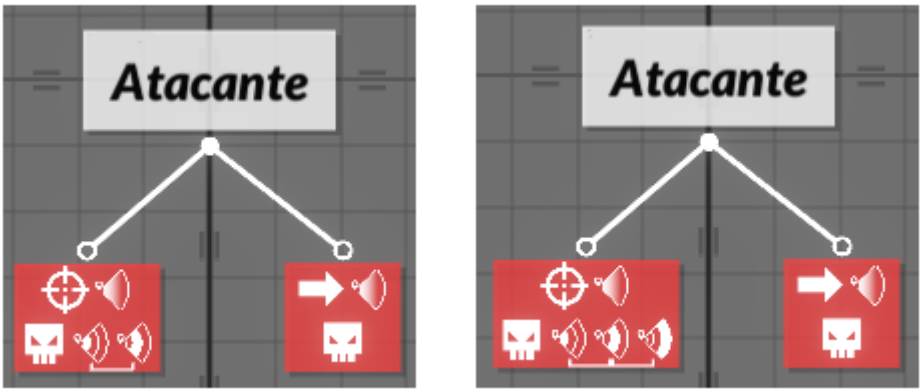


Figura 2.16. IA para un robot de clase francotirador.

AMETRALLADORA.

La clase **ametralladora** se mueve lentamente, y tarda mucho tiempo en apuntar al objetivo. Sin embargo, es muy resistente, tiene una enorme potencia de fuego, y no se mueve más lentamente mientras transporta recursos. Como la ametralladora destaca por su enrme potencia de fuego, pero necesita mucho tiempo para apuntar, vamos a construir su IA para que siga atacando al mismo objetivo hasta que salga de su rango de ataque.

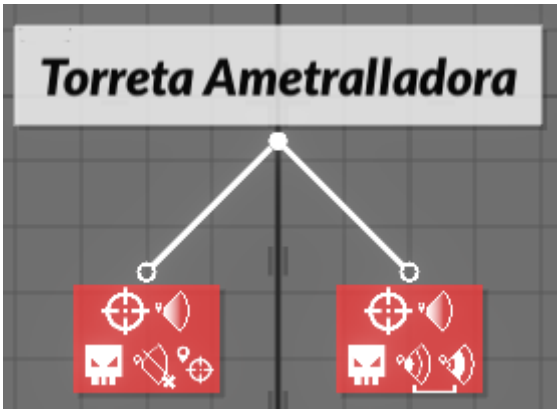


Figura 2.17. IA para un robot de clase ametralladora.

La IA de la figura consta de dos ramas: La rama de la izquierda (la primera en evaluarse) sirve para atacar al bot enemigo más cercano que no esté fuera de nuestro alcance y que ya esté siendo atacado por nosotros, y la rama de la derecha nos permite atacar al bot enemigo más cercano a corta o media distancia. Con esta IA nuestro robot permanece inmóvil a la espera de que el bot enemigo se aproxime a media distancia, momento en el que empezará a atacarlo. Nuestro robot seguirá disparando ráfagas al bot enemigo hasta que éste huya para recargar su escudo y salga de su rango de alcance.

COMPOSICIÓN DEL EQUIPO.

La composición de un equipo puede ser crucial para el resultado de un combate. En esta lección, tres de nuestros robots van a enfrentarse a otros tres bots enemigos. Vamos a acceder a la ventana de configuración del equipo para probar diferentes combinaciones de clases para vencer al equipo contrario. Como veremos, una de las combinaciones que nos permite ganar el combate es elegir un bot de asalto, un francotirador, y una ametralladora (ver figura 2.18).

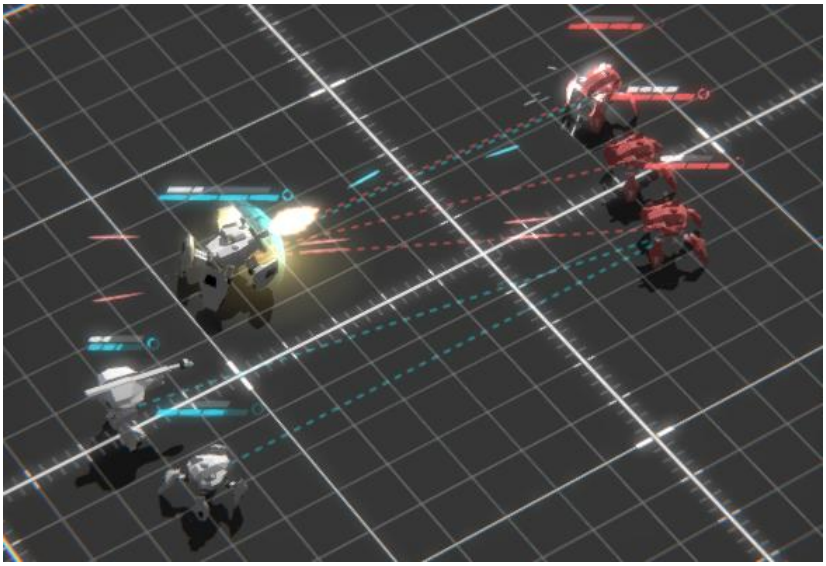


Figura 2.18. El combate entre nuestros tres robots y los tres bots enemigos.

La figura 2.19 muestra la IA que incorpora esta lección para definir el comportamiento de los tres robots, sean de la clase que sean.

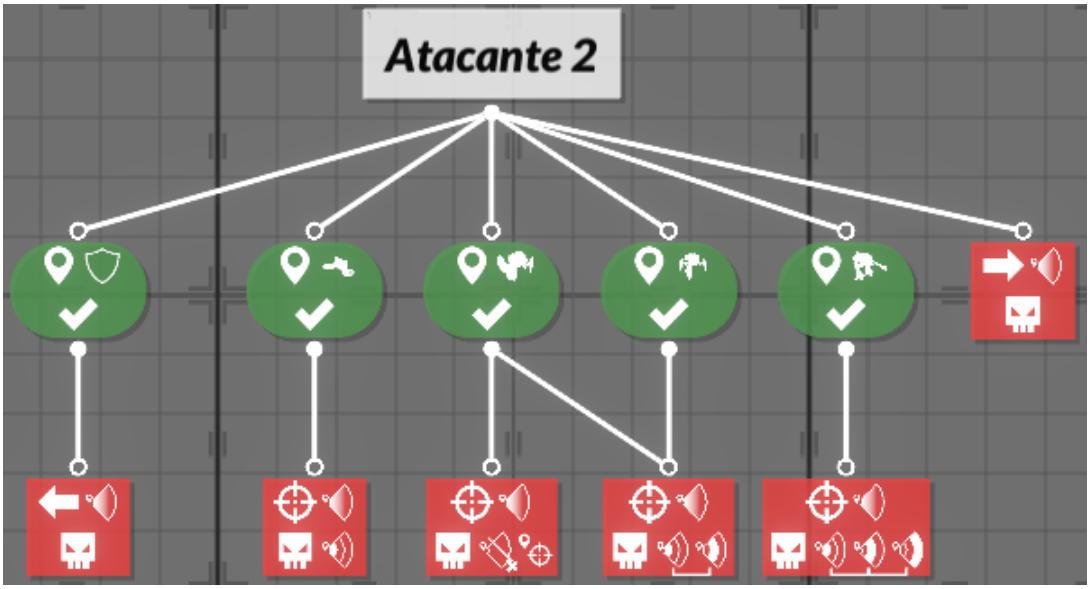


Figura 2.19. IA para controlar el ataque y la retirada de un robot, sea cual sea su clase.

De izquierda a derecha, incluye un total de seis ramas: (1) Si el escudo está vacío, e independientemente de su clase, el robot huye del bot enemigo más cercano. (2) Si el robot es de la clase escopeta, ataca al bot enemigo más cercano que se sitúe a corta distancia. (3) Si el robot es de la clase ametralladora, primero intenta atacar al bot enemigo más cercano al que ya esté atacando y que no esté fuera de alcance. De no verificarse esa rama, trata de atacar al bot enemigo más cercano a corta o media distancia. (4) Si el robot es de la clase asalto, intenta atacar al bot enemgo más cercano a corta o media distancia. (5) Si el robot es de la clase francotirador, ataca la bot enemigo más cercano a corta, media, o larga distancia. (6) Por último, e independientemente de su clase, el robot se aproxima al bot enemigo más cercano.

PAQUETES DE VIDA.

Los paquetes de vida recargan nuestros puntos de salud al máximo cada vez que los capturamos. Sin embargo, una vez recogidos desaparecen, así que debemos usarlos sabiamente.

Figura 2.20. Paquetes de vida.

En esta misión, nuestro robot se enfrenta a dos bots enemigos. La IA con la que controlaremos a nuestro robot es la mostrada en la figura 2.21. La única rama novedosa es la de la izquierda: Si el nivel de salud de nuestro robot está entre el 0% y el 25% o entre el 25% y el 50%, enviamos al robot a recoger el paquete de vida más cercano.

Figura 2.21. IA para recoger los paquetes de vida.

Al lanzar el combate, nuestro robot consigue derrotar el primer bot enemigo con facilidad. Pero como ha combatido con dos bots simultáneamente, su nivel de salud se ve afectado. En el instante en el que la vida de nuestro robot disminuye del 50% se verifica la primera rama, y el robot acude a recoger el paquete de vida para restaurar sus puntos de salud. Con su salud restaurada, el robot puede volver a combatir al segundo bot enemigo hasta ganar la partida.

CAMPOS DE FUERZA.

Los **campos de fuerza** reducen el daño que recibe el robot que lo ha capturado. El efecto desaparece cuando el robot abandona al campo de fuerza en el que estaba. Un campo de fuerza solo puede ser capturado por un robot, así que no debemos permitir que nuestros oponentes se hagan con los campos de fuerza antes que nosotros.

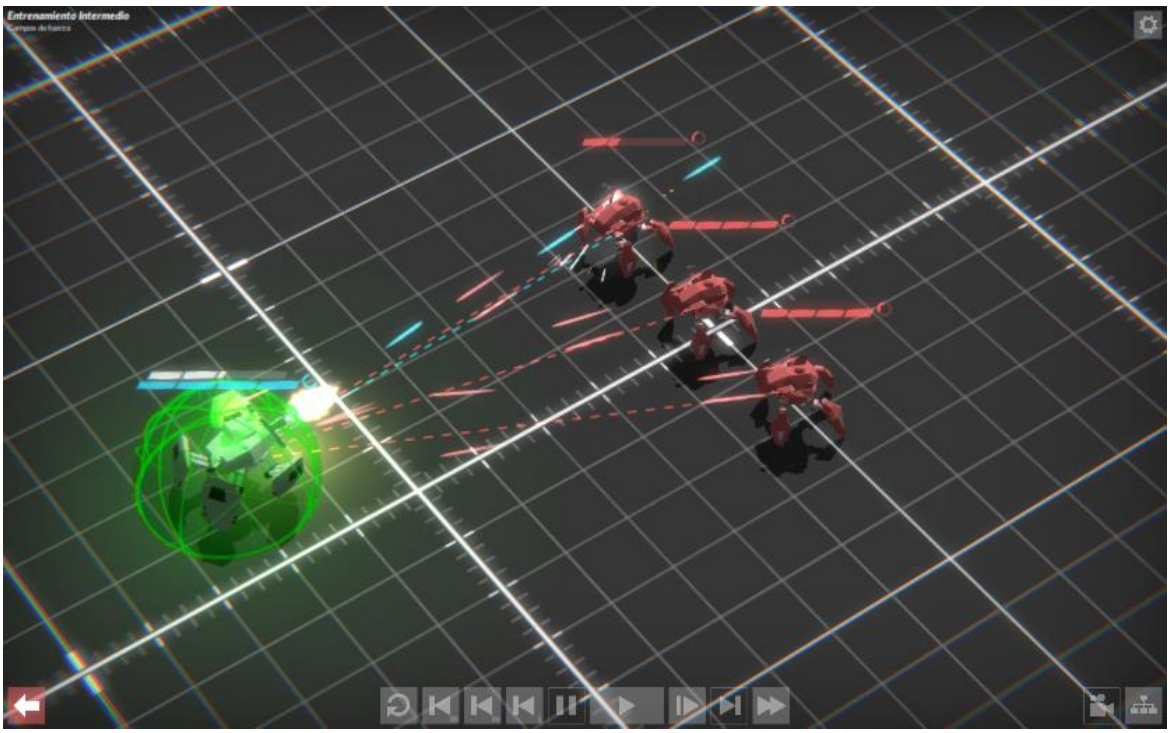


Figura 2.22. Campos de fuerza.

En esta lección nuestro robot (clase ametralladora) se enfrenta a tres bots enemigos. Afortunadamente, la arena de combate incluye un campo de fuerza en las proximidades de nuestro robot. La IA que gobernará el comportamiento de nuestro robot es la mostrada en la figura. La rama de la izquierda le indica al robot que capture el campo de fuerza más cercano.

Figura 2.23. IA para capturar un campo de fuerza y atacar desde él.

Al comenzar el combate, lo primero que hace nuestro robot es acudir al campo de fuerza más cercano para capturarlo. Tras capturar el campo de fuerza, nuestro robot puede combatir a los tres bots enemigos uno tras otro hasta conseguir derrotarlos.

PROBAMOS LO QUE HEMOS APRENDIDO (2).

Ahora que ya hemos aprendido algunas cosas importantes sobre Gladiabots, vuelve al Nivel 2 del Capítulo 1 del modo Colección, para modificar tus IAs y comprobar si eres capaz de superarlo. Prueba varias cosas: (1) Revisa el orden de prioridad de las distintas ramas de tu IA (recuerda que una rama a la izquierda siempre se evalúa antes que todas las ramas a su derecha). (2) Recuerda gestionar adecuadamente tu escudo. (3) Prueba a escribir distintas IAs para controlar a cada uno de tus robots. (4) Observa que los filtros de objetivo de las acciones y de las condiciones son muy variados, y nos permiten un gran control de la acción y de la condición que estamos utilizando, así que úsalos sabiamente; etc.

Cuando hayas conseguido superar el Nivel 2 de este primer capítulo del modo Coelcción, guarda la IA con el nombre "C12.jpg" (**C**olección Capítulo **1** Nivel **2**). No sigas probando con más niveles; vamos a completar el tutorial con el entrenamiento avanzado.

3.8. ENTRENAMIENTO AVANZADO.

PERSEGUIR AL OBJETIVO ACTUAL.

En esta lección el bot enemigo aparece muy cerca de un recurso. Nuestra misión es perseguirlo y destruirlo antes de que pueda anotar el recurso conseguido en su base.

La figura 2.24 muestra la IA de control de nuestro robot. Recordemos que al definir un nodo de acción, podemos aplicar filtros a nuestro objetivo en función de la acción actual sobre ese objetivo. En esta IA vamos a utilizar la acción de la rama de la izquierda para continuar atacando al enemigo al que ya estemos atacando actualmente.

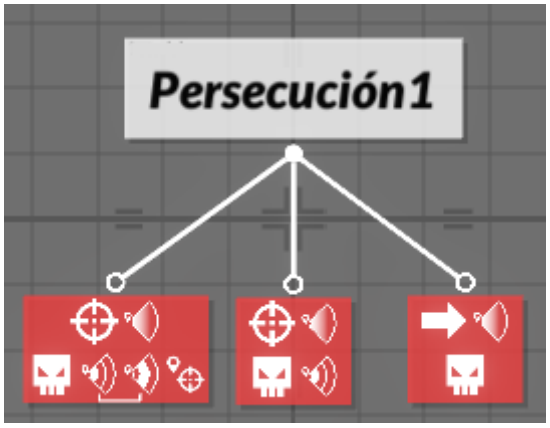


Figura 2.24. IA para perseguir y destruir a un bot antes de que anoote un recurso.

Como nuestro robot es de la clase escopeta, lo hemos programado para acercarse al bot enemigo rapidamente, atacarlo a corta distancia, y seguir atacándolo hasta haberlo destruido.

PERSEGUIR AL OBJETIVO ETIQUETADO.

De nuevo, en esta lección nuestra misión es destruir a enemigo antes de que consiga anotar un recurso en su base.

En Gladiabots podemos **etiquetar** a ciertas entidades para fijarlas como objetivos de nuestras acciones. Sin embargo, debemos tener en cuenta que nuestro robot es el único que podrá ver las etiquetas que pone. La IA de la figura 2.25 nos permiten etiquetar a nuestro enemigo y atacarlo. La acción de la izquierda sirve para etiquetar el bot enemigo más cercano que esté a corta distancia; la acción central permite atacar a corta distancia al bot enemigo más cercano que haya sido etiquetado; y por último, la acción de la derecha hace que nuestro robot se aproxime al bot enemigo más cercano.

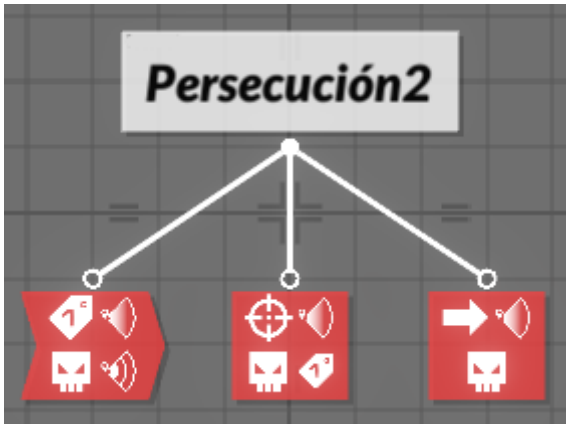


Figura 2.25. IA para etiquetar y atacar a un bot enemigo.

Al ejecutar esta IA en la arena de combate, nuestro robot comienza acercándose al bot enemigo más cercano, que en esta lección, consigue capturar un recurso e intenta anotarlo en su base. Al acercarse a corta distancia nuestro robot etiqueta al bot enemigo, y una vez etiquetado empieza a atacarlo. Esta estrategia nos permite derrotar al bot enemigo antes de que llegue a anotar el recurso capturado.

FUEGO CONCENTRADO.

En esta nueva misión tres de nuestros robots se enfrentan a otros tres bots enemigos. Para conseguir derrotarlos, vamos a utilizar las **etiquetas de equipo**. Las etiquetas de equipo son un poco diferentes de las etiquetas normales, ya que son visibles para todos los miembros de un equipo. Las etiquetas de equipo pueden utilizarse para transmitir información y órdenes entre los robots de un mismo equipo.

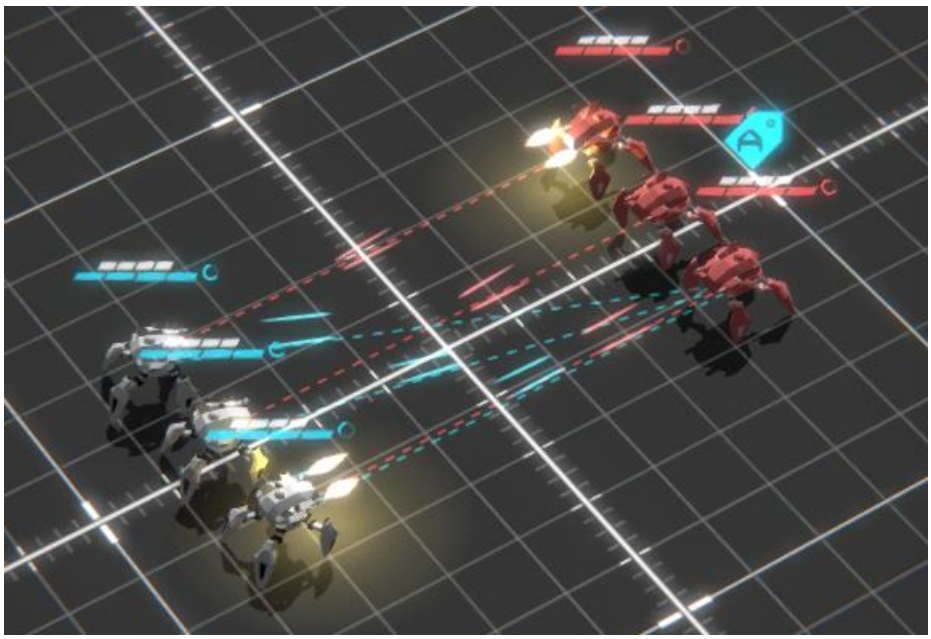


Figura 2.26. Los tres robots de nuestro equipo concentrando el fuego sobre el bot enemigo etiquetado.

Accedemos a la configuración del equipo, donde vemos que nuestros robots tienen disponibles las IAs llamadas "Líder de fuego concentrado" y "Fuego concentrado". Aquí vamos a asignar la IA "Líder de fuego

concentrado" a uno de nuestros robots, lo que le permitirá etiquetar un objetivo para concentrar el fuego del equipo sobre él. Los otros dos robots vendrán controlados por la IA "Fuego concentrado", para atacar al bot enemigo etiquetado por el líder.

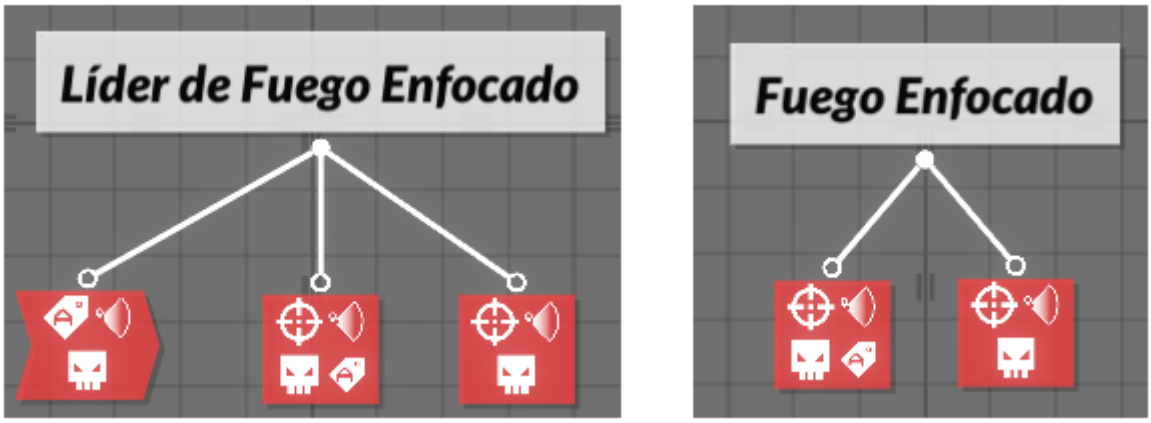


Figura 2.27. Las IAs "Líder de fuego concentrado" y "Fuego concentrado".

Al comenzar el combate, el robot líder etiqueta a uno de los bots enemigos para que todos los miembros del equipo concentren el fuego sobre él (ver figura 2.26). Con esta estrategia derrotamos uno a uno a todos los miembros del equipo rival, y conseguimos la victoria con facilidad.

NODOS CONECTORES.

En esta lección nuestro equipo de dos robots se enfrenta a un equipo enemigo de tres bots en una partida en modo Colector. En este caso, la IA que necesitaremos usar para ganar la partida será más compleja. Cuando se combinan varias condiciones y acciones, las IAs pueden desordenarse fácilmente. Para evitar este problema y simplificar el diseño de las IAs, podemos usar **nodos conectores**. En esta lección vamos a usar un nodo conector para enlazar las dos condiciones y las tres acciones de la IA mostrada en la figura 2.28(a). Recordemos que los conectores siempre se consideran válidos durante la ejecución de una IA.

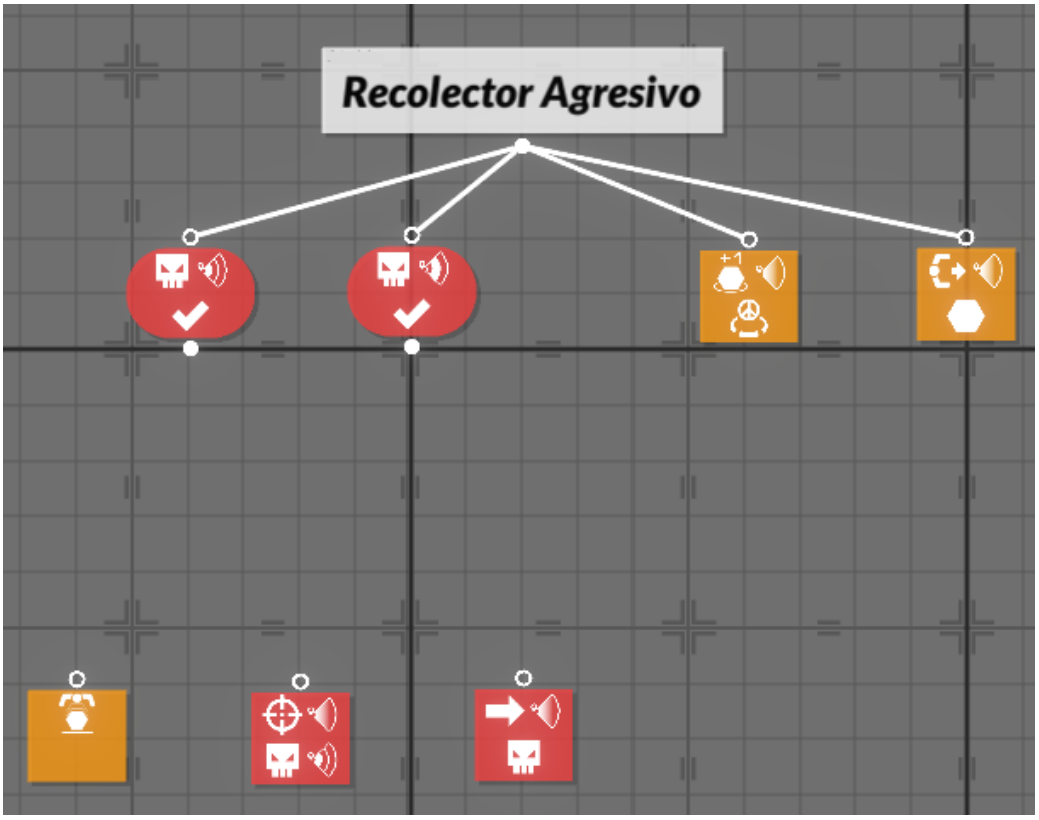


Figura 2.28(a). IA sin terminar.

La figura 2.28(b) muestra la IA de esta lección una vez completada con el nodo conector. En el caso de que el bot enemigo esté a corta o a media distancia de uno de nuestros robots, estos procederán a evaluar el árbol de acciones bajo el nodo conector, a saber, soltarán el recurso que estén transportando, atacarán al enemigo más cercano a corta distancia, o se aproximarán al bot enemigo más cercano. En el caso de no cumplirse ninguna de estas dos condiciones, nuestros robots tratarán de ejecutar alguna de las dos últimas ramas, para anotar el recurso transportado en la base aliada más cercana, o para acudir a recoger el recurso más cercano.

Al iniciar la partida, nuestros dos robots acuden directamente a recoger los dos recursos para transportarlos a las bases y anotar puntos. Pero en cuanto un bot enemigo queda a corta o media distancia de nuestros robots, sueltan los recursos transportados y empiezan a atacarlo hasta destruirlo. Una vez destruido, los robots vuelven a recoger los recursos y a llevarlos hacia la base. Y de nuevo, si un bot enemigo entra en el rango de cortas o medias distancias, sueltan los recursos y lo atacan hasta destruirlo. Una vez destruidos todos los bots enemigos, nuestros robots pueden finalmente anotar los recursos en la base aliada para ganar la partida.

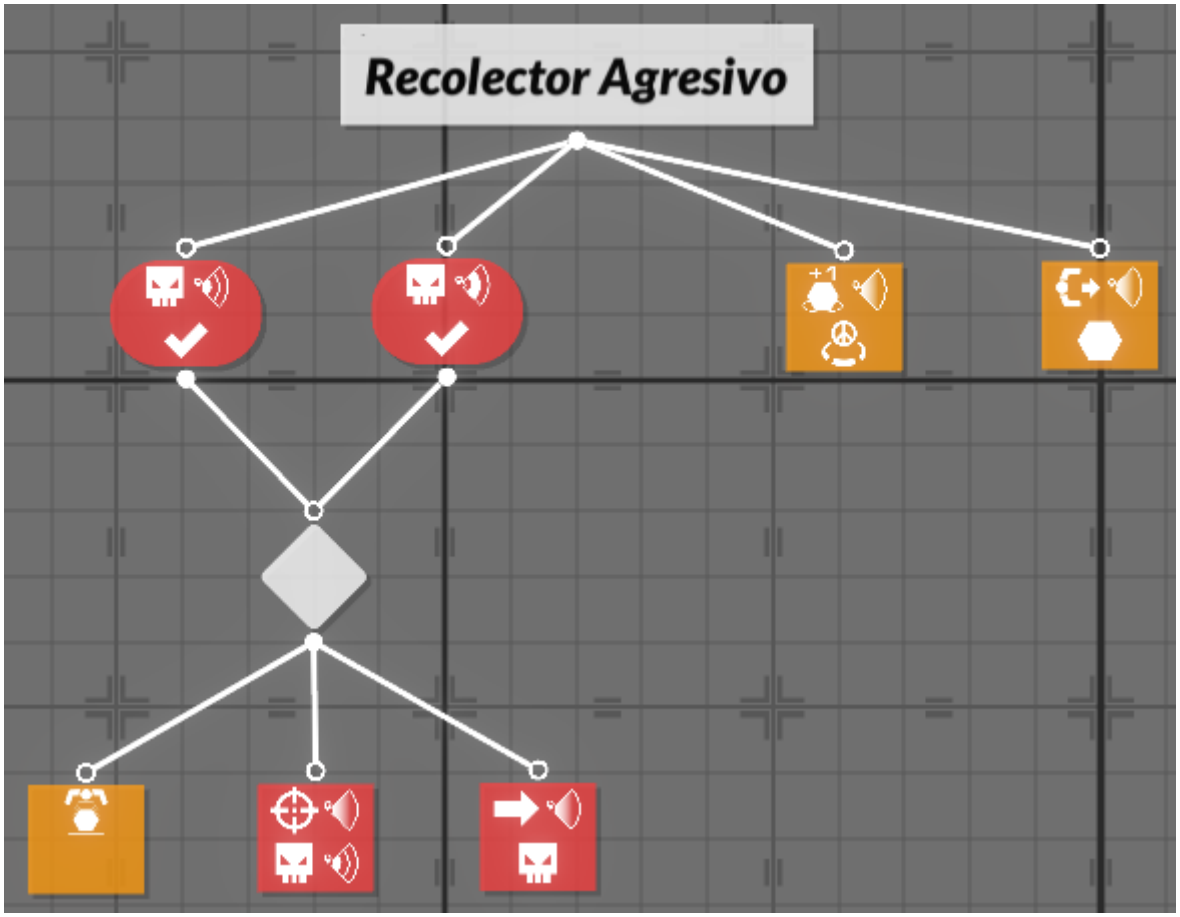


Figura 2.28(b). IA terminada.

SUB-IAs.

En Gladiabots podemos reutilizar una IA creada previamente dentro de otra IA. Para ello, debemos usar los nodos llamados **Sub-IAs**. En esta lección comenzamos construyendo una IA vacía llamada "IA Maestra". Tras arrastrar un enlace desde el nodo raíz, creamos un nuevo nodo de tipo Sub-IA. A continuación, seleccionamos una de las IAs disponibles (a saber, Atacar y Escudo). Una vez agregada la Sub-IA, pulsamos en el botón con forma de flecha hacia abajo para abrirla, y en el botón de la flecha hacia arriba para cerrarla y volver a la IA que la integra. También podemos pulsar en el botón "Lista de IAs" para acceder a la lista de IAs disponibles.

En nuestro caso, la IA llamada "IA Maestra" estará compuesta por dos Sub-IAs: Las IAs "Escudo" y "Atacar", ver figura 2.29. La figura 2.30 muestra la estructura interna de estas dos Sub-IAs.

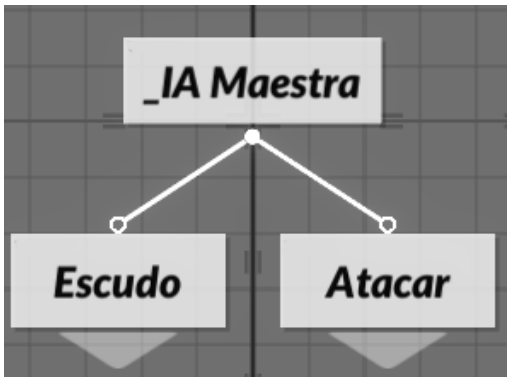


Figura 2.29. Una IA compuesta por dos Sub-IAs.

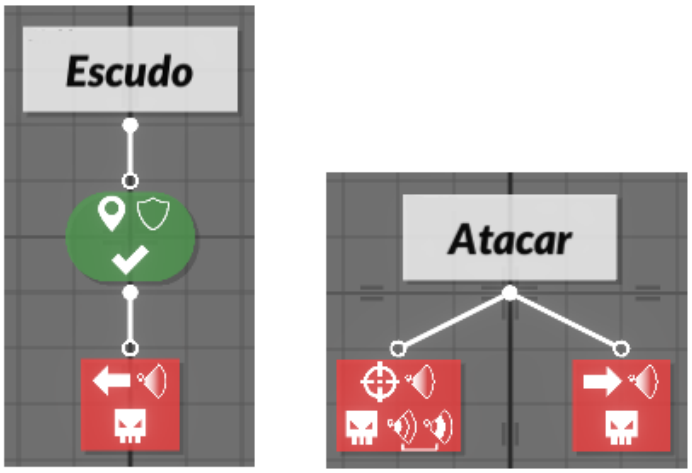


Figura 2.30. Las Sub-IAs "Escudo" y "Atacar".

Con esta IA, nuestro robot es capaz de enfrentarse y vencer a dos bots enemigos, como vemos en la figura 2.31.

Figura 2.31. La IA "IA Maestra" ejecutándose en la arena de combate.

3.9. ESTRATEGIAS.

La wiki de Gladiabots incluye una sección sobre estrategias, a la que podemos acceder a través del enlace https://wiki.gladiabots.com/index.php?title=Strategies. Una vez estudiadas, disponemos de total libertad de usar estas estrategias en nuestras IAs para los modos Colección, Dominación, y Eliminación, ya que podrían resultarnos útiles para hecernos con la victoria.

RETIRADA.

Huir cuando el nivel de escudo es superior al 0%. Cuando estamos siendo atacados por varias unidades a media distancia, vale la pena retirarse con antelación. No debemos esperar a que nuestro escudo caiga a un nivel del 0%, porque los enemigos que nos están atacando a medias distancias seguirán haciéndolo cuando entremos en el rango de grandes distancias, y eso es algo que querremos evitar si nuestro escudo está tan bajo (ver IA "Retirada 1"). En concreto, si estamos siendo atacados por una ametralladora o un francotirador, probablemente querremos retirarnos con cualquier nivel de salud o de escudo (ver IA "Retirada 2").

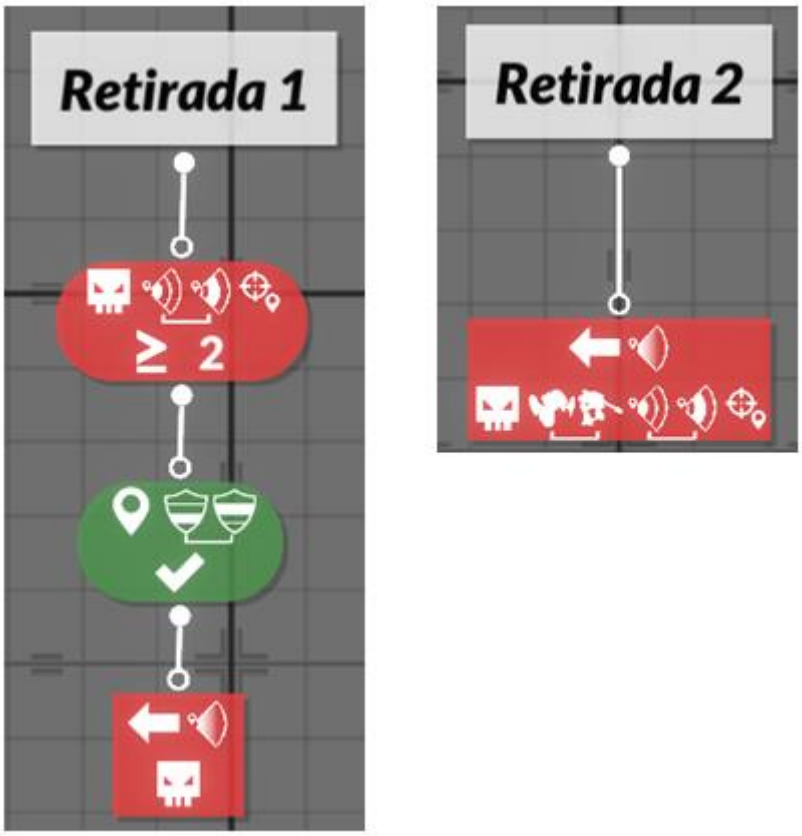
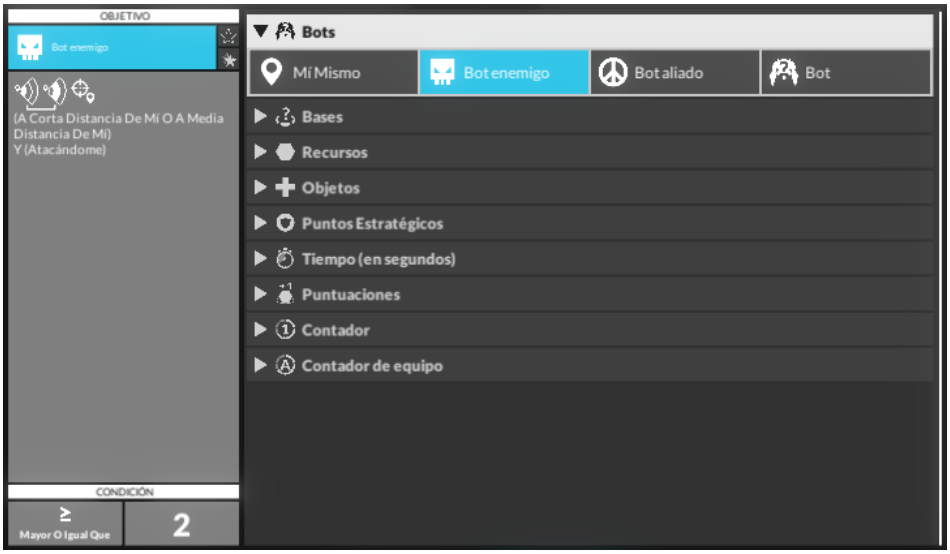
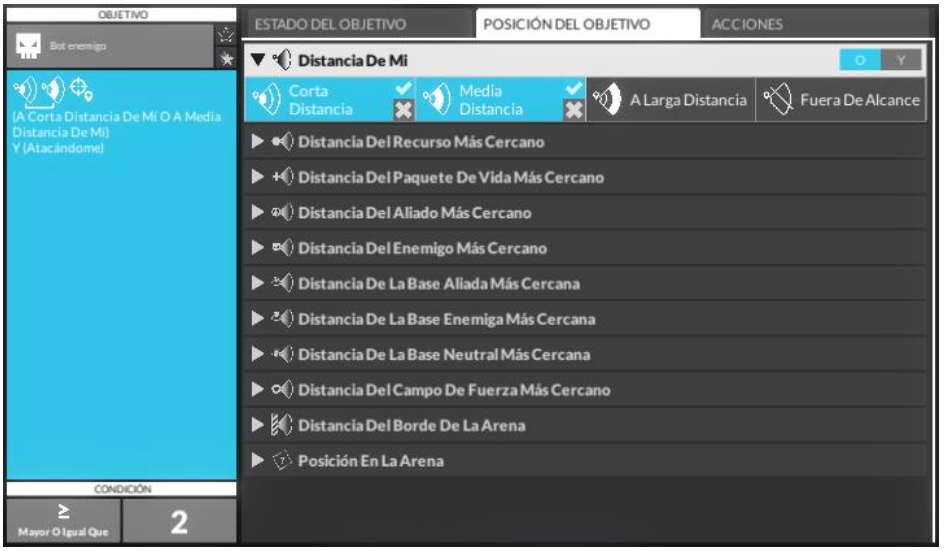


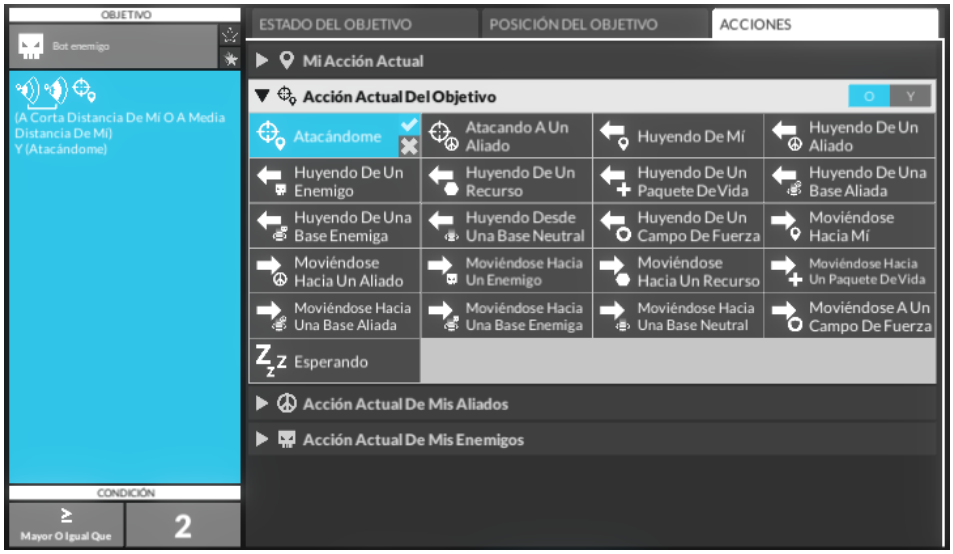
Figura 2.32. A la izquierda, una IA para que el robot huya cuando el número de enemigos que le están atacando a corta y media distancia es mayor o igual que dos, y si su escudo está entre el 75% y el 25% de su capacidad. A la derecha, una IA para que el robot huya cuando está siendo atacado por un robot de tipo ametralladora o francotirador a cortas o medias distancias.

¿Cómo hemos construido estas IAs? Comencemos con la IA "Retirada 1". Para el primer condicional hemos seleccionado como objetivo los bots enemigos; en los filtros de objetivos hemos especificado enemigos a corta o media distancia de mí que estén atacándome; y en la condición hemos seleccionado que su número sea mayor o igual que dos (ver figuras de la serie 2.33). Por cierto, notar que en los distintos filtros de objetivo podemos seleccionar el tick (✓) o la cruz (✗) para indicar si queremos que se cumpla ese filtro (por ejemplo, que el bot enemigo esté atacándomo) o que no se cumpla ese filtro (por ejemplo, que el bot enemigo no esté atacándome). Para el segundo condicional nos seleccinamos como objetivo a nosotros mismos (esto es, al propio robot); en los filtros de objetivo indicamos que el escudo esté entre el 75% y el 50%, y entre el 50% y el 25%; y en la condición especificamos que queremos que se cumpa esa condición,

seleccionando la opción "existe". Finalmente, el nodo de acción específica que huyamos del bot enemigo más cercano cuando se cumplan los dos nodos de condición de los que cuelga. Puede que nos parezca muy complicado interpretar una IA ya escrita, como la IA "Retirada 1". Sin embargo, recordemos que al pasar el ratón sobre cada uno de los nodos que constituyen la IA, aparece un cuadro de texto que indica en palabras llanas el significado de esa IA, ver figura 2.34.







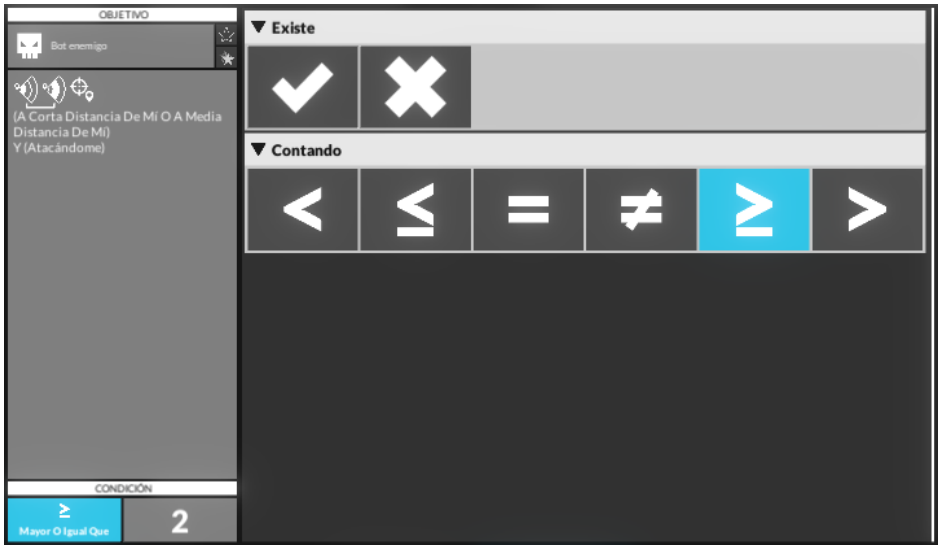


Figura 2.33. Proceso de construcción del primer nodo condicional de la IA "Retirada 1".

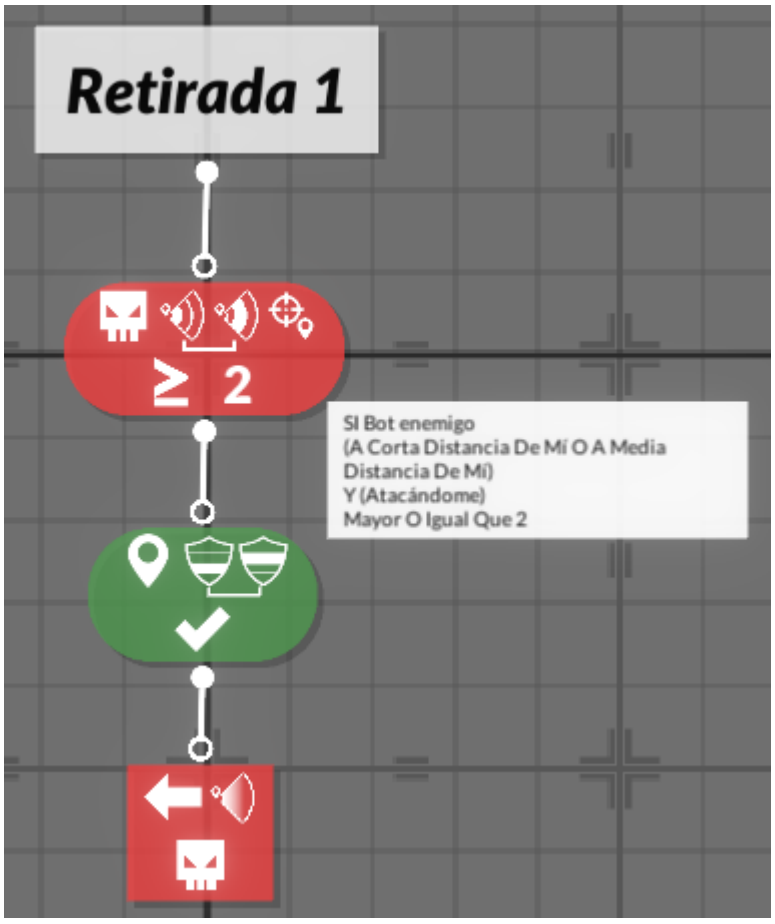
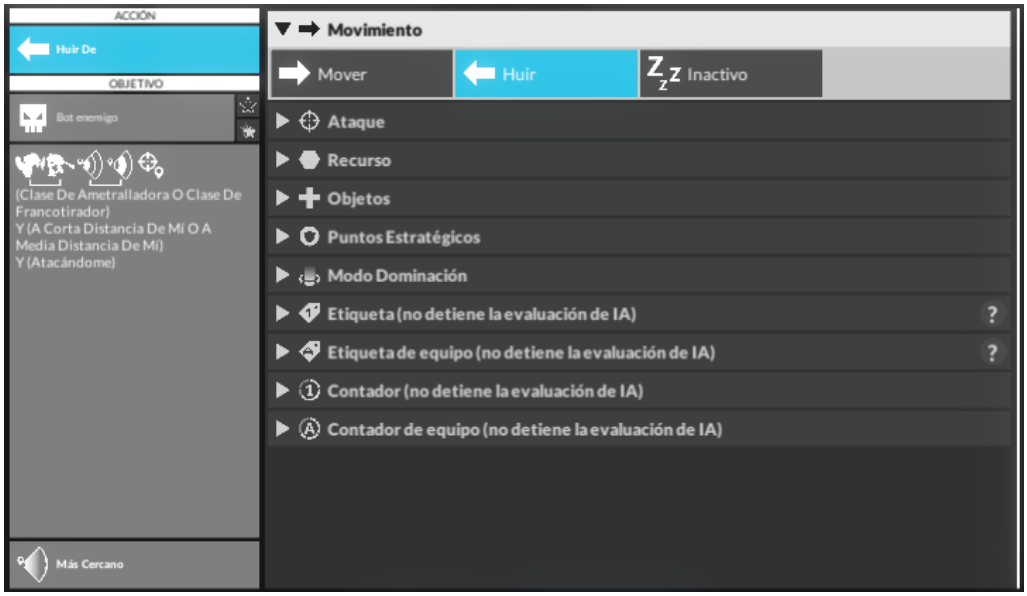
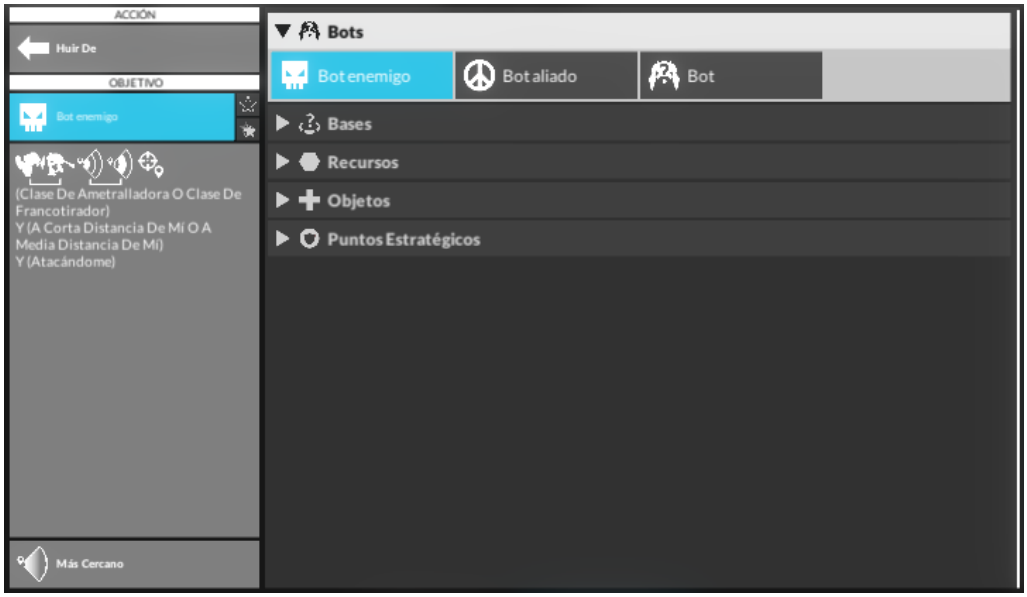


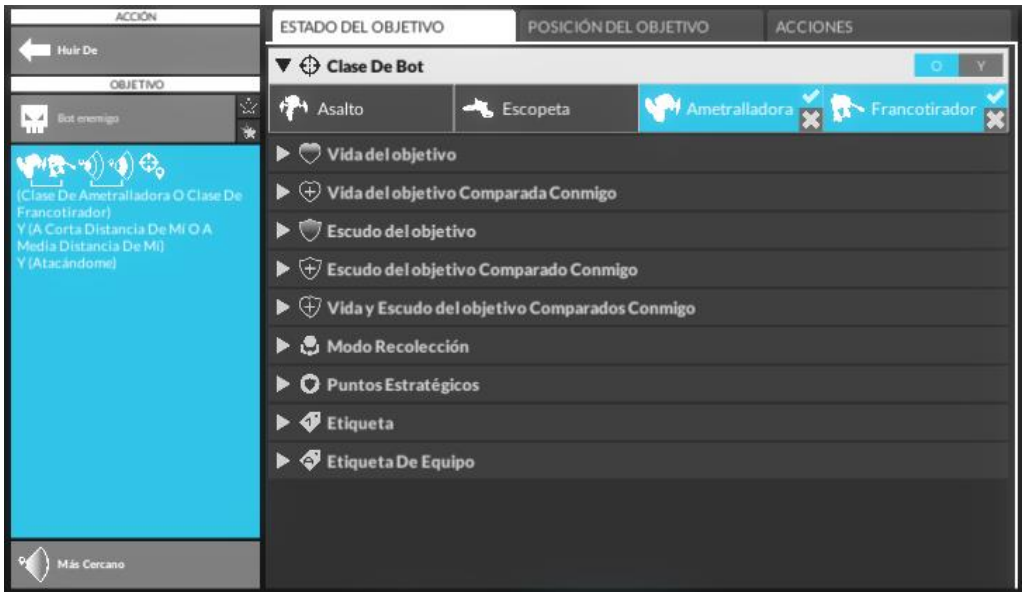
Figura 2.34. Caja de texto con el contenido del primer condicional de la IA "Retirada 1".

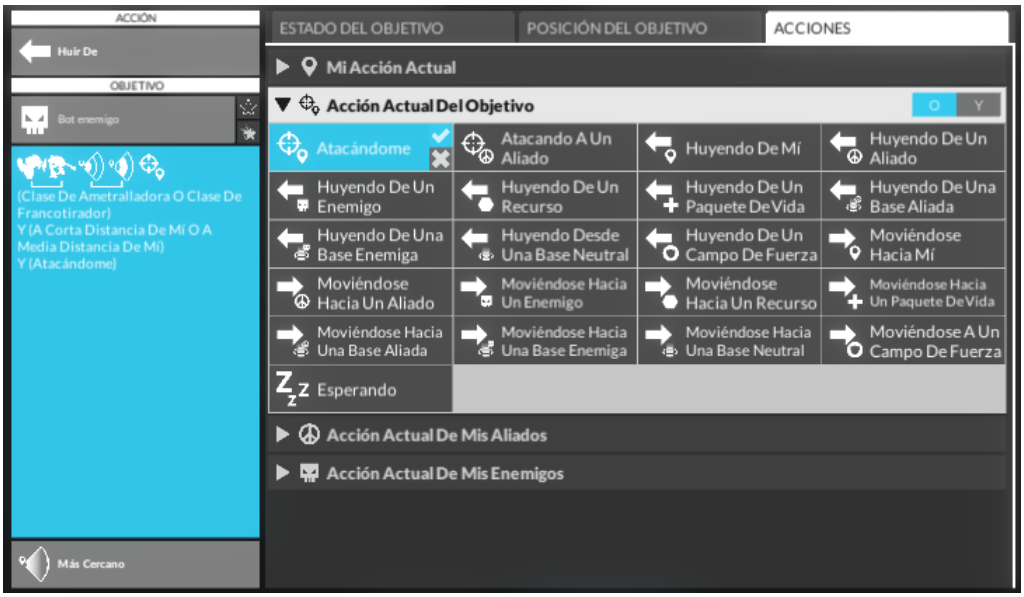
Para construir la IA "Retirada 2" hemos usado un único nodo de acción, que combina todos los requerimientos de esa acción. Las figuras de la serie 2.35 muestran el proceso de construcción de ese nodo.

En primer lugar, elegimos la acción de huir. A continuación, en Objetivo, especificamos que queremos huir de un bot enemigo. En los filtros de objetivo indicamos la clase de bot de la que queremos huir (ametralladora o francotirador), la distancia a la que deben estar estos bots (corta y media distancia), y que estén atacándome.


ACCIÓN
Huir De
OBJETIVO
Bot enemigo
(Clase De Ametralladora O Clase De Francotirador)
Y (A Corta Distancia De Mí O A Media Distancia De Mí)
Y (Atacándome)
Más Cercano
Movimiento
Mover
Huir
Inactivo
Ataque
Recurso
Objetos
Puntos Estratégicos
Modo Dominación
Etiqueta (no detiene la evaluación de IA)
Etiqueta de equipo (no detiene la evaluación de IA)
Contador (no detiene la evaluación de IA)
Contador de equipo (no detiene la evaluación de IA)


ACCIÓN
Huir De
OBJETIVO
Bot enemigo
(Clase De Ametralladora O Clase De Francotirador)
Y (A Corta Distancia De Mí O A Media Distancia De Mí)
Y (Atacándome)
Más Cercano
Bots
Bot enemigo
Bot aliado
Bot
Bases
Recursos
Objetos
Puntos Estratégicos


ACCIÓN
Huir De
OBJETIVO
Bot enemigo
(Clase De Ametralladora O Clase De Francotirador)
Y (A Corta Distancia De Mí O A Media Distancia De Mí)
Y (Atacándome)
Más Cercano
ESTADO DEL OBJETIVO
POSICIÓN DEL OBJETIVO
ACCIONES
Clase De Bot
O
Y
Asalto
Escopeta
Ametralladora
Francotirador
Vida del objetivo
Vida del objetivo Comparada Conmigo
Escudo del objetivo
Escudo del objetivo Comparado Conmigo
Vida y Escudo del objetivo Comparados Conmigo
Modo Recolección
Puntos Estratégicos
Etiqueta
Etiqueta De Equipo



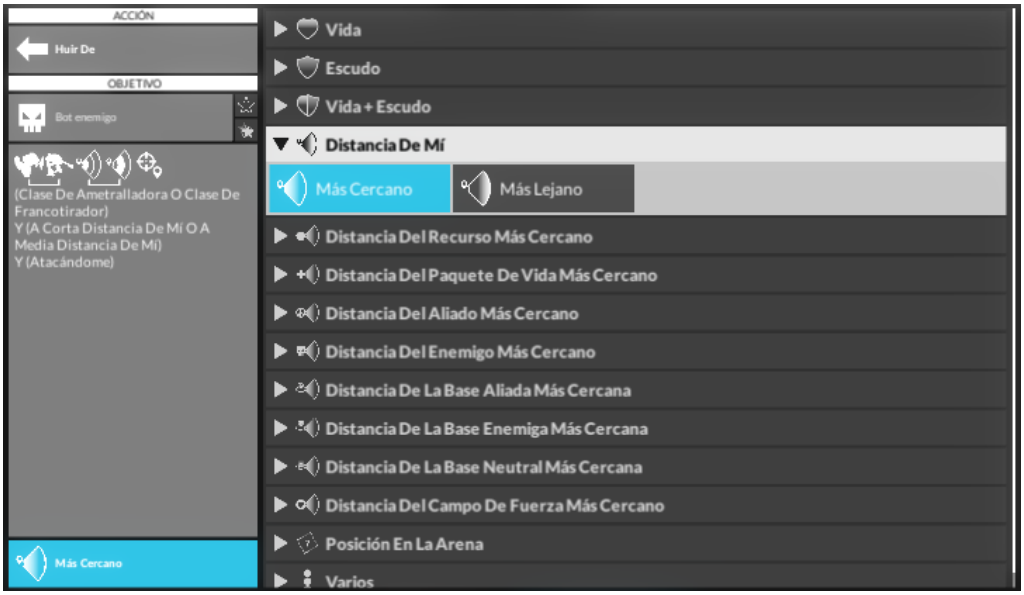


Figura 2.35. Proceso de construcción del nodo de acción de la IA "Retirada 2".

Huir sólo mientras nos estén atacando. No vale la pena huir simplemente porque nuestro escude esté bajo, sino porque además nos estén atacando. Si dejan de atacarnos, deberíamos detener el movimiento de huída.

ATAQUE.

Minimizar el tiempo empleado en cambiar el objetivo de nuestro ataque. Normalmente nuestros robots deberían continuar atacando al bot enemigo al que comenzaron a atacar, para evitar cambiar a un objetivo más cercano. En el caso de los robots ametralladora y francotirador, en los que el tiempo para fijar un nuevo objetivo es muy prolongado, esto es particularmente importante. En estos casos, nuestra IA debería incluir unas reglas estrictas que indiquen cuándo deberían cambiar de objetivo.

Fuego concentrado. Debemos priorizar nuestros ataques sobre aquellos enemigos que ya estén siendo atacados por nuestros aliados. También deberíamos priorizar nuestros ataques sobre enemigos cuyo escudo esté a un nivel entre el 25% y el 0%.

Centrar la atención en el francotirador. Siempre debemos prestar especial atención al francotirador enemigo. Si nuestro francotirador consigue alcanzar al francotirador enemigo, es recomendable que nuestras unidades más próximas pasen a atacarle, incluso en el rango de medias distancias (siempre y cuando valga la pena hacerlo y/o puedan hacerlo con seguridad).

PRESIONAR.

Atacar a media distancia. Los ataques a grandes distancias causan muy poco daño, pero los ataques a medias distancias son más eficaces y nos permiten presionar a nuestros enemigos en retirada, especialmente cuando tenemos varias unidades presionando simultáneamente. Pero cuidado: Una presión mal ejecutada puede ser una táctica muy arriesgada que puede llegar a convertirse en contraproducente.

Presionar en el momento preciso. Nunca deberíamos presionar cuando ya nos están atacando. En ciertas ocasiones, especialmente cuando tengamos muchos enemigos cerca, deberíamos abortar el proceso de presión.

Flanquear y rodear al enemigo. Los robots que han sido flanqueados o rodeados son fáciles de destruir. Para flanquear a los bots enemigos podemos usar una combinación de movimientos de aproximación y huída de los bots enemigos, de las bases, y de los recursos.

RECURSOS.

Puntuar de forma inteligente. Las acciones de anotar puntos tienen ventajas e inconvenientes: Si invertimos demasiado tiempo en puntuar, quedaremos indefensos ante la presión de los bots enemigos. Pero no invertir tiempo en puntuar probablemente implicará que perdamos la partida por 0 a 1. Siempre debemos estar alerta al nivel de salud de nuestros robots, y vigilar si el enemigo está puntuando.

Otras estrategias. Al principio de la partida deberíamos apresurarnos a recoger los recursos más cercanos. En fases más avanzadas de la partida podremos hacernos con esos recursos menos accesibles. En ciertas ocasiones, una buena estrategia podría consistir en recoger 2 o 3 recursos al mismo tiempo.

ESCOPETA.

El robot escopeta siempre debería *empezar* atacando al primer bot enemigo que detecte a media distancia. Lo mejor que podemos hacer es aproximarnos lo máximo posible a las unidades a media distancia, especialmente si están huyendo o si ya están siendo atacadas por un aliado. Esta estrategia le ofrecerá al

robot escopeta más disparos a media distancia mientras las unidades enemigas intentan escapar del rango de medias distancias. Un ejemplo sería dejar de avanzar hacia las unidades a media distancia únicamente cuando ya tengamos al menos dos unidades visibles a media distancia.

3.10. MODO COLECCIÓN.

Tras completar el tutorial y presentar algunas estrategias básicas, deberíamos estar preparados para empezar con los modos de juego individuales. Pero antes de empezar, recuerda que debes limpiar Gladiabots de las IAs que incluye por defecto: Para ello, acude a la pestaña "Editor de IA", borra todas las IAs que no hayas tú, y avisa al profesor para que verifique que ya no quedan IAs preconstruidas. A partir de ahora, tendrás que escribir tus propias IAs sin más ayuda que lo que hayas aprendido. ¡Vamos a ello!

En la pestaña "Solo", y bajo el Tutorial, podemos acceder al modo de juego Colección. Recordemos que en este modo de juego individual, el objetivo es recoger y puntuar el mayor número de recursos. El modo Colección se compone de un total de 20 capítulos, cada uno de ellos con varios niveles y etapas finales. Conforme vayamos completando cada capítulo, el juego nos permitirá acceder al siguiente.

Aquí vamos a comenzar con el capítulo 1 (con 5 niveles y una etapa final), e iremos avanzando capítulo a capítulo hasta completar unos cuantos. Más adelante probaremos el resto de modos de juego, a saber, el modo Dominación y el modo Eliminación.

COLECCIÓN: CAPÍTULO 1, NIVEL 1.

En el primer nivel del capítulo 1, dos de nuestros robots se enfrentan a otros dos bots enemigos por el control de tres recursos. Recordemos que tras terminar el entrenamiento básico, ya escribimos una IA para superar este nivel. Sin embargo, y con todo lo aprendido hasta ahora, deberíamos ser capaces de construir una nueva IA que nos permita superar esta misión de forma mucho más eficiente.

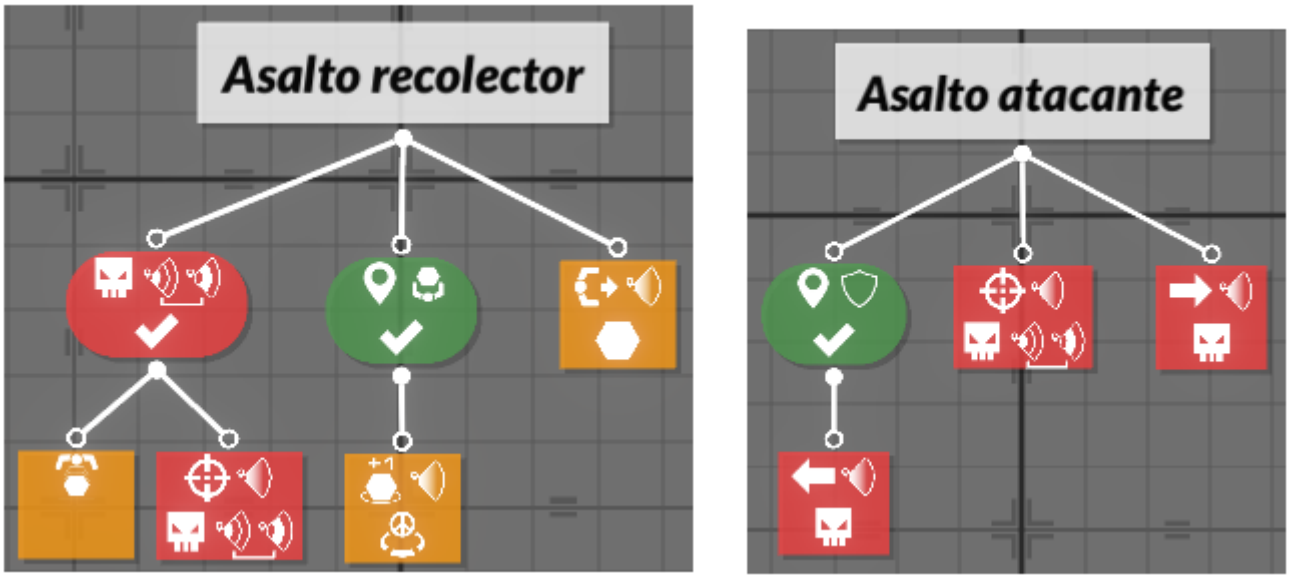


Figura 3.1. Las IAs "Asalto recolector" y "Asalto atacante".

En mi caso, he optado por escribir dos IAs distintas, llamadas "Asalto recolector" y "Asalto atacante", que he asignado a cada uno de mis dos robots de asalto. La figura 3.1 muestra estas dos IAs. La IA "Asalto recolector" consta de tres ramas: (1) Si hay un bot enemigo a corta o media distancia del robot recolector, éste suelta el recurso que esté transportando y ataca a ese bot enemigo. (2) Si nuestro robot transporta un recurso, anota el recurso en la base aliada más cercana. (3) En otro caso, el robot se dirige hacia el recurso más cercano para recogerlo. La IA "Asalto atacante" consta de otras tres ramas: (1) Si su escudo

está vacío, el robot huye del bot enemigo. (2) El robot ataca al bot enemigo más cercano a corta o media distancia. (3) El robot se dirige al bot enemigo más cercano.

Con estas dos IAs, nuestros robots consiguen ganar la partida con un marcador de 2 a 1 a nuestro favor. Es posible que se te ocurran otras formas de programar a los robots para conseguir una vistoria más eficiente; no dudes en hacerlo. Por proporner algunas ideas, podríamos probar a asignar a ambos robots la IA "Asolto recolector", para ver qué pasa. Otra idea sería modificar la IA "Asalto atacante" para que persiga y ataque prioritariamente a aquellos bots enemigos que estén transportando un recurso, etc. Por supuesto, se valorará al alza a aquellos alumnos que consigan ganar la partida con un tanteo mejor que el obtenido con estas IAs. Por ejemplo, unas IAs que consigan un marcador de 2 a 0 obtendrán mayor puntuación, y a su vez, unas IAs que consigan un marcador de 3 a 0 una mayor puntuación todavía.

COLECCIÓN: CAPÍTULO 1, NIVEL 2.

En esta misión, dos de nuestros robots de asalto se enfrentan a otros dos bots de asalto enemigos. En este caso hay dos bases aliadas y dos bases enemigas, y los equipos compiten por el control de 5 recursos. De nuevo, nuestra misión es programar a los robots para conseguir ganar la partida.

Por supuesto, podemos volver a usar las IAs escritas en el ejemplo previo. Pero de nuevo, recordar que cuanto más favorable sea el marcador para nuestro equipo, mejor será la nota que obtendremos en esta misión.

COLECCIÓN: CAPÍTULO 1, NIVEL 3.

En este tercer escenario nuestros robots se enfrentan a otros tres bots enemigos. En la arena de cobate hay tres bases enemigas, y otras tres bases aliadas. Los robots competirán por el control de 7 recursos.

En este caso, y como disponemos de tres robots, podemos configurar uno de ellos para que sea de otra clase distinta a la clase Asalto. Disponemos de total libertad para configurar el equipo a nuestro gusto, y únicamente debemos recordar que los robots de las clases Escopeta, Amatralladora, y Francotirador solo pueden constituir el 35% de nuestro escuadrón.

En ocasiones, puede que lo único que necesitemos sea redistribuir las IAs que ya teníamos escritas antes entre un mayor número de robots disponibles. Otras veces necesitaremos reescribir las IAs para conseguir superar el nivel. Sea como sea, reasigna o resscribe las IAs de los tres robots para conseguir ganar la partida de la manera más eficiente, con el menor número de bajas, y con el mejor marcador posible.

COLECCIÓN: RESTO DE CAPÍTULOS Y NIVELES.

Como comprobaremos, los sucesivos niveles y capítulos del modo Colección se vuelven más y más complejos con bastante rapidez. Las IAs que hayamos constuído en los primeros niveles sin duda fracasarán en los niveles sucesivos, y no nos permitirán progresar en la campaña. Es por ello que os propongo que uséis la IA de la figura 3.2 ("Ultimate AI for beginners"). Esta IA está sacada directamente de la sección de estrategia de de la wiki oficial de Gladiabots, y nos permite superar todos los niveles de los dos primeros capítulos del modo Colección sin tener que hacer el más mínimo cambio. Además, y con unos pocos retoques para las clases francotirador y ametralladora, podremos avanzar en el capítulo 3 y escalar en la lista de clasificados de la liga del modo Colección.

Como vemos, esta IA consta de 6 ramas, que de izquierda a derecha especifican lo siguiente: (1) Si el escudo del robot está por debajo del 50%, y si está siendo atacado por un enemigo a corta, media, o larga distancia, el robot primero suelta el recurso que esté transportando y luego huye del bot enemigo más cercano que le esté atacando. (2) Si el robot está transportando un recurso, lo anota en la base aliada más

cercana. (3) Esta rama permite definir tres tipos de ataque: Primero, el robot intenta atacar al bot enemigo más cercano a corta distancia; luego trata de atacar al bot enemigo con la vida más débil cuyo escudo esté por debajo del 25% y que se localice a corta, media, o larga distancia; y finalmente intenta atacar al bot enemigo con el escudo más débil a corta o media distancia. (4) Si el nivel de su escudo está por encima del 75% y ningún robot enemigo le está atacando, el robot primero intenta atrapar el recurso más cercano, y en otro caso trata de acercarse al bot enemigo más cercano. (5) El robot ataca al enemigo con el escudo más débil a corta, media, o larga distancia. (6) Finalmente, el robot se aproxima al bot enemigo más cercano.

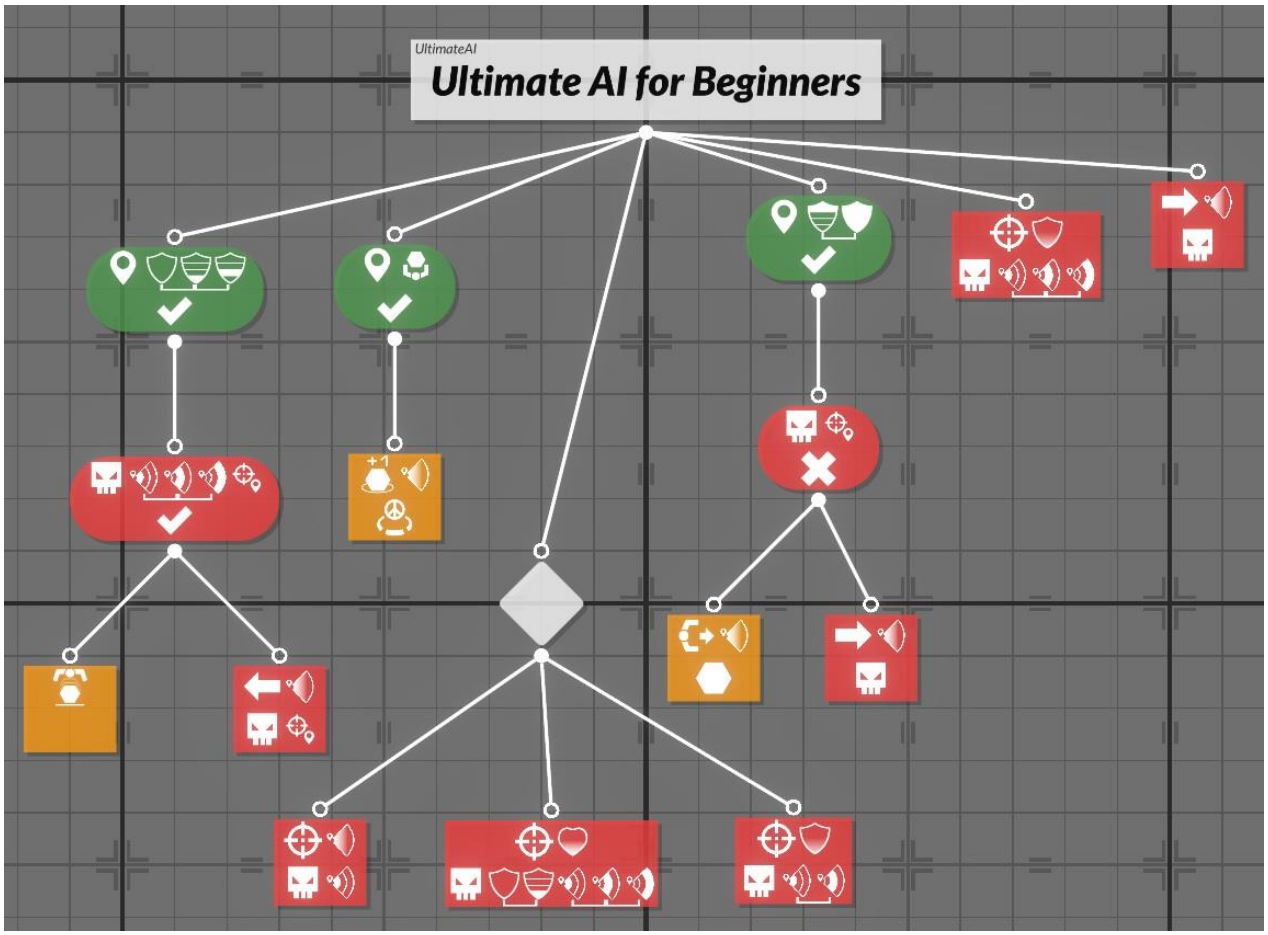


Figura 3.2. "Ultimate AI for beginners", una IA optimizada para los primeros niveles del modo Colección.

¿Por qué os doy una solución? Muy sencillo: La mejor forma de aprender a desarrollar mejores IAs de las que empezamos sabiendo escribir es revisando otras IAs más eficientes, y entendiendo la metodología para construirlas. De esta forma, aprenderemos un montón de cosas que realmente no llegamos a comprender en el tutorial. Para construir esta IA basta con observar los iconos que incluyen los nodos, y a continuación, buscar esos mismos iconos en el editor. Y para entender cuál es la función de los distintos nodos de una IA, solo necesitamos leer el cuadro de texto que aparece al pasar el ratón sobre ellos. Este proceso os ayudará a sentiros más cómodos con el desarrollo de IAs en Gladiabots, y para escribir vuestras propias IAs a partir de la que aquí usamos como ejemplo.

La figura 3.3 muestra otro ejemplo de IA inicial con la que podríamos comenzar en el modo Colección. Esta IA, llamada "Babby - Collection", se ha diseñado para un escuadrón donde todos los robots son de la clase Asalto. Como vemos en la figura, la IA consta de 3 ramas, que a su vez, se dividen en varias subramas. De izquierda a derecha, la IA especifica lo siguiente: (1) En primer lugar, el robot anota el recurso más cercano en una base aliada a corta distancia. En otro caso, si su escudo está por debajo del 50% y está siendo atacado por un bot enemigo, el robot huye de cualquier bot enemigo que le esté atacando y que no esté fuera de alcance. (2) La segunda rama es la rama de las acciones de ataque: Primero, el robot ataca al enemigo de escudo más débil que esté transportando un recurso y que se localice a corta o media distancia.

Luego, el robot trata de atacar al enemigo más cercano que no esté fuera de alcance y al que ya estuviese atacando antes. Por úmtimo, el robot intenta atacar al enemigo de escudo más débil a corta o media distancia. (3) La última rama también define múltiples acciones: Primero, el robot tratará de anotar el recurso más cercano en una base aliada. En segundo lugar, el robot trata de coger el recurso más cercano al que no se esté aproximando ya un aliado. En tercer lugar, el robot se aproxima al bot enemigo más cercano que esté transportando un recurso (para atacarlo). Por último, el robot simplemente se aproxima al bot enemigo más cercano.

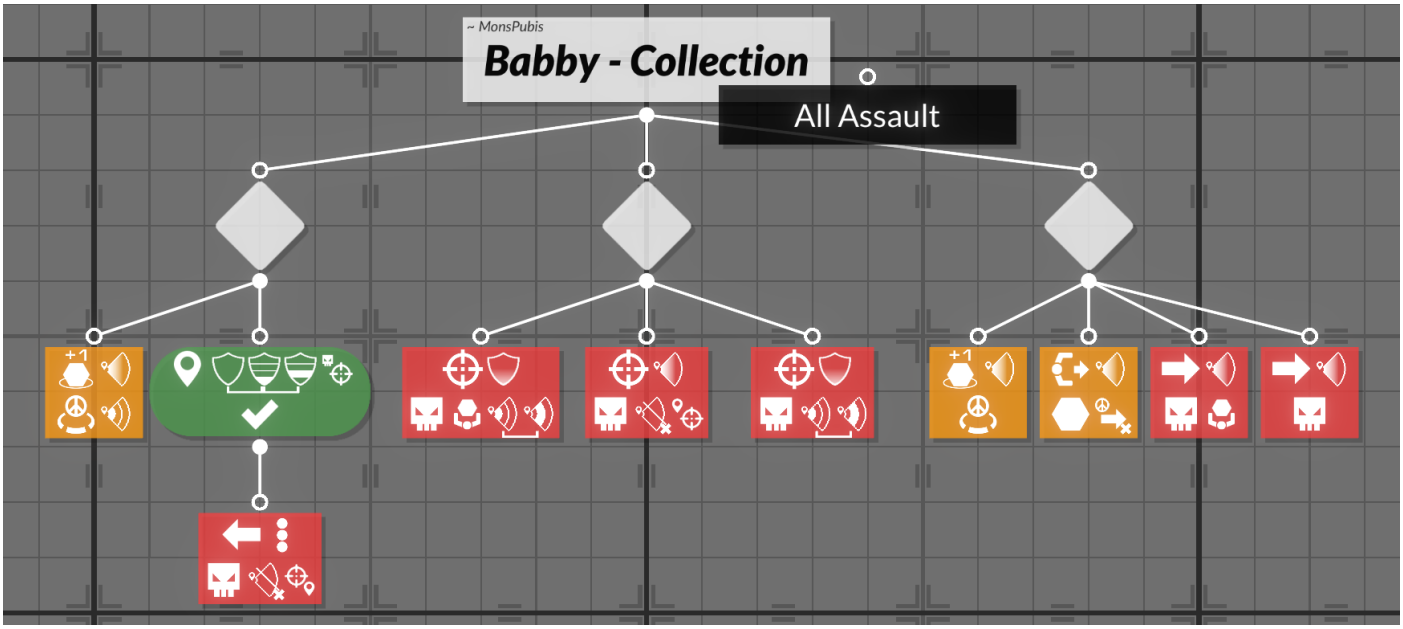


Figura 3.3. "Babby - Collection", otra IA adaptada a los primeros niveles del modo Colección.

Notar que, llegados a este punto, nuestras IAs empiezan a ser más y más extensas, por lo que será muy importante ser organizados y limpios a la hora de escribirlas. Para ello, lo mejor es comenzar construyendo una sola IA para todos nuestros robots. Más adelante, podremos refinar esta IA añadiendo condicionales que especifiquen diferentes comportamientos para cada clase de robot de nuestro equipo. Otro aspecto importante es hacer nuestras IAs los más sencillas posible. Si nuestras IAs son muy enrevesadas será muy fácil perderse a la hora de buscar malfuncionamientos, y añadir más nodos para corregir errores solo conseguirá que el funcionamiento sea todavía peor.

Con estos consejos, vamos a intentar ir avanzando en los distintos niveles y capítulos del modo Colección, hasta conseguir dominar el proceso de construcción de IAs y haber aprendido a programar IAs cada vez más eficientes. Cuando ya nos sintamos cómodos en el modo Colección, estaremos preparados para probar el modo Dominación.

3.11. MODO DOMINACIÓN.

En este modo de juego, el objetivo es capturar y mantener bases para conseguir el mayor número de puntos posible. El modo Dominación consta de 13 capítulos, cada uno de ellos con varios niveles y etapas finales. Conforme vayamos completando cada capítulo, el programa nos dará acceso al siguiente. Empecemos.

DOMINACIÓN: CAPÍTULO 1, NIVEL 1.

En este primer nivel del capítulo 1, dos de nuestros bots (asalto) se enfrentan a otros dos bots enemigos (asalto) por el control de una base neutral. La arena también incluye dos campos de fuerza.

De nuevo, he optado por escribir dos IAs distintas, llamadas "Asalto atacante y líder" y "Asalto conquistador". La idea es que el robot atacante se aproxime a un campo de fuerza para atacar a los bots enemigos desde allí y hacer de cebo. En cuanto los bots enemigos concentren su ataque en el robot atacante, el robot conquistador acudirá a la base para tomarla y anotar puntos.

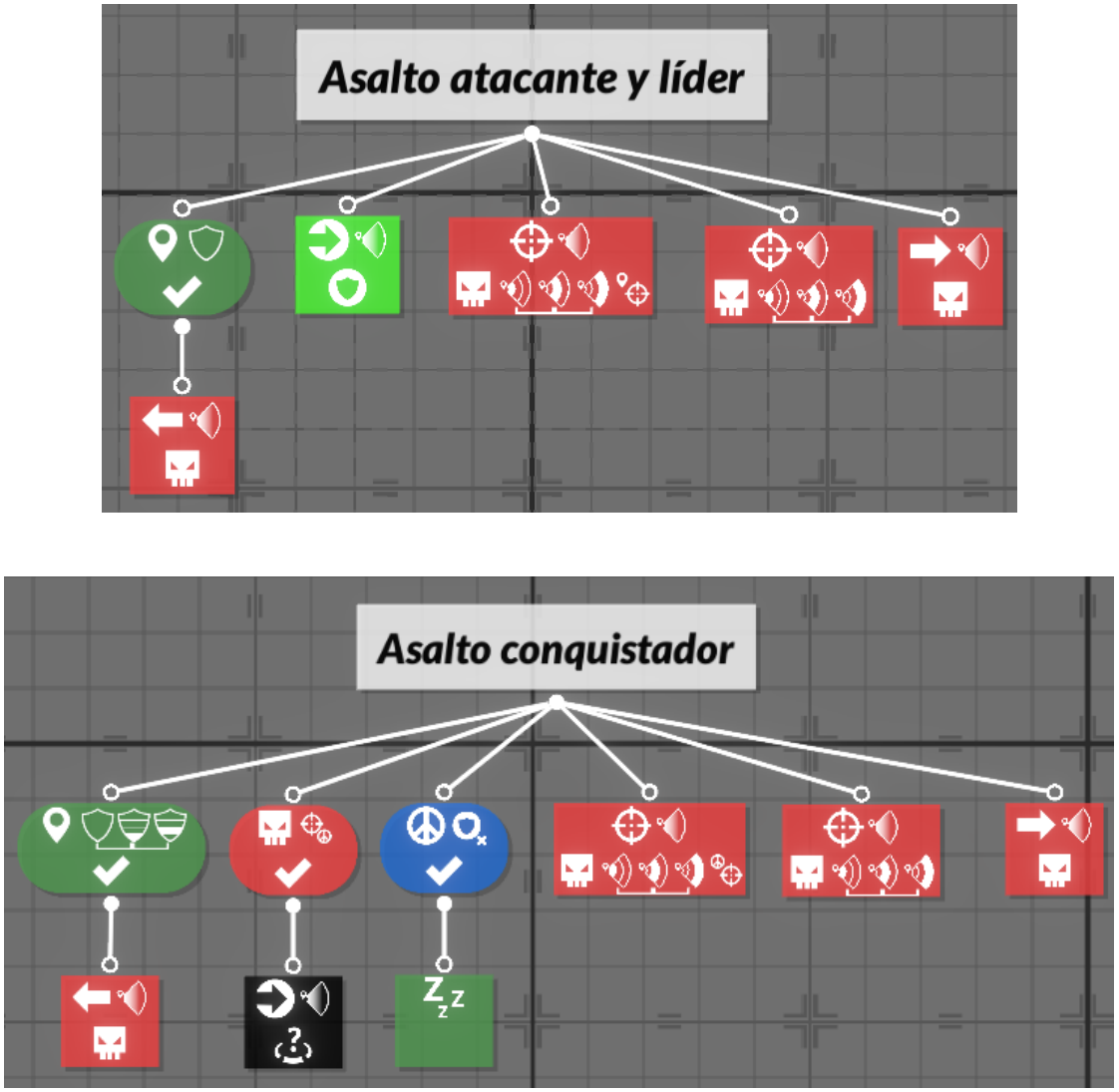


Figura 3.3. Las IAs "Asalto atacante y líder" y "Asalto conquistador".

La figura 3.3 muestra las IAs de los robots atacante y conquistador. Como vemos, la IA del atacante consta de 5 ramas: (1) Si el escudo queda vacío, el atacante huye del bot enemigo más cercano. (2) El robot acude a capurar el campo de fuerza más cercano. (3) El robot ataca al bot enemigo a corta, media, o larga distancia que ya estuviese atacando. (4) El robot ataca al bot enemigo a corta, media, o larga distancia más próximo a él. (5) Por último, el robot se aproxima al bot enemigo más cercano. Por su parte, la IA del conquistador es más compleja, y consta de 6 ramas: (1) Si su escudo está por debajo del 50%, el robot huye del enemigo má cercano. (2) Si los bots enemigos están atacando a un aliado, el robot acude a capturar la base más cercana. (3) Si el robot aliado todavía no ha capturado un campo de fuerza, el robot permanece inactivo. (4) El robot ataca al bot enemigo a corta, media, o larga distancia que esté siendo atacado por el robot aliado. (5) El robot ataca al bot enemigo a corta, media, o larga distancia más próximo a él. (6) Por último, el robot se aproxima al bot enemigo más cercano.

Con estas IAs conseguimos que nuestros robotd ganen la partida con un tanteo de 34 a 0, y sin que resulten destruidos en ninguna ocasión. (Recordemos que, en modo Dominación, los robots destruidos vuelven a aparecer al cabo de unos pocos segundos). Intenta escribir tus prpias IAs sin usar las IAs de este ejemplo, para conseguir la victoria más eficiente posible (esto es, con mejor tanteo y menor númerode bots aliados destruidos). De nuevo, se valorará al alza a aquellos alumnos que consigan ganar la partida con un mejor tanteo y un menor número de bajas aliadas.

DOMINACIÓN: CAPÍTULO 1, NIVEL 2.

En este segundo nivel, tres de nuestros robots se enfrentarán a otros tres bots enemigos (todos ellos de la clase asalto) por el control de tres bases neutrales. La arena de combate también dispone de tres campos de fuerza.

Por supuesto, podríamos seguir con nuestros tres robots de asalto y usar las mismas IAs del nivel previo, pero nos daríamos cuenta que en este nuevo nivel esas IAs fracasan estrepitosamente, puesto que estaban muy adaptadas a los circunstancias del primer nivel.

Con un escuadrán de tres robots no es obligatorio que sean todos de la clase asalto; podemos elegir que uno de los tres sea un robot de otra clase (escopeta, ametralladora, o francotirador). Por ello os propongo que uséis una nueva clase de robot y construyáis las nuevas IAs que os permitan ganar la partida de la forma más eficiente (mayor tanteo posible, menor número de bajas aliadas posible).

DOMINACIÓN: RESTO DE CAPÍTULOS Y NIVELES.

De nuevo, y como ocurría en el modo Colección, en los niveles subsiguientes las cosas se ponen más y más complicadas con bastante rapidez. De nuevo, os propongo que uséis la IA de la figura 3.4 ("Babby - Domination"), también extraída de la sección de estrategia de la wiki de Gladiabots. Con esta IA podremos superar varios de los primeros niveles del modo Dominación. Esta IA nos servirá de muestra para aprender a programar las IAs del modo Dominación, y será el esqueleto sobre el que construiremos nuestras IAs mejoradas para superar el resto de niveles.

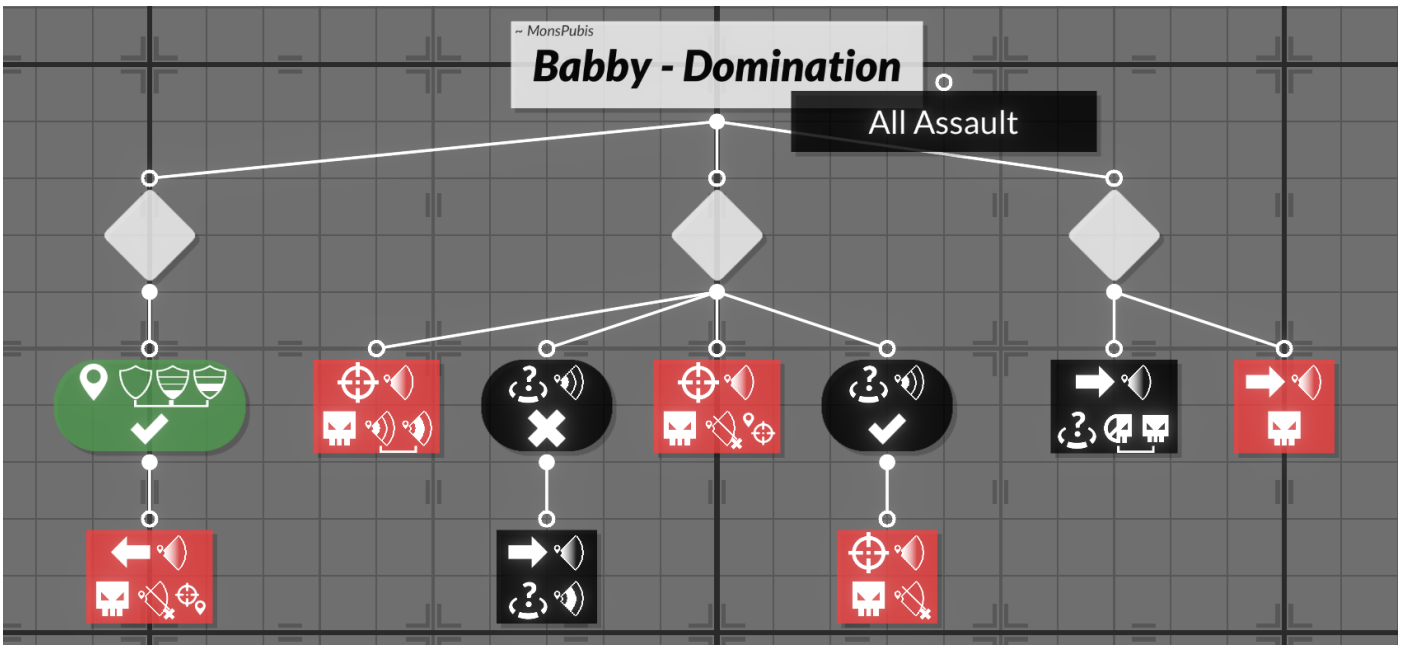


Figura 3.4. "Babby - Domination", una IA optimizada para los primeros niveles del modo Dominación.

Como vemos en la figura, esta IA funciona para un escuadrán de robots de tipo Asalto. La IA constra de 3 ramas, que de izquierda a derecha, especifican lo siguiente: (1) Si el nivel de su escudo está por debajo del 50%, el robot huye del bot enemigo más cercano que no esté fuera de su alcance y que le esté atacando. (2) Esta compleja rama consta de cuatro subramas, que se evalúan de izquierda a derecha como sigue: (2a) Primero, el robot intenta atacar al bot enemigo más cercano a corta o media distancia. (2b) A continuación, y si no hay una base a corta distancia, trata de aproximarse a una base a media distancia. (2c) Después intenta atacar al bot enemigo más cercano que no esté fuera de alcance y al que ya estuviese atacando previamente. (2d) Por último, y si hay una base a corta distancia, el robot ataca al bot enemigo más cercano que no esté fuera de alcance. (3) En la tercera y última rama de la IA, el robot empieza tratando de

acercarse a la base neutral o enemiga más cercana, y en caso de no ser posible, trata de aproximarse al bot enemigo más cercano.

A partir de esta muestra de IA, trataremos de avanzar en los distintos niveles y capítulos del modo dominación, hasta haber aprendido a programar IAs cada vez más eficientes. Cuando ya nos sintamos cómodos en el modo Dominación, podemos probar el modo de juego más complicado de todos: Eliminación.

3.12. MODO ELIMINACIÓN.

El objetivo final del modo Eliminación es simplemente destruir el mayor número de bots enemigos, dentro del tiempo límite establecido. Recordar que, en este moo de juego, los robots destruidos no reaparecen en la arena de combate (en otras palabras, su muerte es permanente).

El modo Eliminación consta de 12 capítulos, cada un de ellos con varios niveles y etapas finales, alos que podremos ir accedeiendo conforme vayamos completando los niveles y capítulos previos. Comencemos con el nivel 1 del capítulo 1.

ELIMINACIÓN: CAPÍTULO 1, NIVEL 1.

En este primer nivel dos de nuestros robots de asalto se enfrentan a otros dos bots de asalto enemigos. La rena de combate incuye tres campos de fuerza en ubicaciones simétricas respecto a los puntos de aparición de ambos escuadrones.

Yo he optado por programar a mis dos robots con la misma IA, a la que he llamado "Asalto Eliminación". Como vemos en la figura 3.5, mi IA que consta de tres ramas: La primera (izquierda) es la rama para huir, la segunda (centro) es la rama de ataque, y la tercera (derecha) es la rama de aproximación. De secciones previas, ya deberíamostener claro el significado de cada una de las ramas: (1) Si el nivel de su escudo está por debajo del 50% y está siendo atacado, el robot huye del bot enemigo más cercano que le esté atacando. (2) La rama de ataque procede como se indica: Primero tratamos de atacar al bot enemigo más cercano a corta distancia; a continuación atacamos al bot enemigo más cercano a corta, media, o larga distancia al cual ya estuviese atacando; después atacamos al bot enemigo de vida más débil a corta, media, o larga distancia, y cuto escude esté por debajo del 25%; finalmente, atacamos al bot enemigo de escudo más débil a corta, media, o larga distancia. (3) La IA termina intentando acercarse al bot enemigo más cercano que no haya capturado un campo de fuerza, o simplemente, a bot enemigo más cercano.

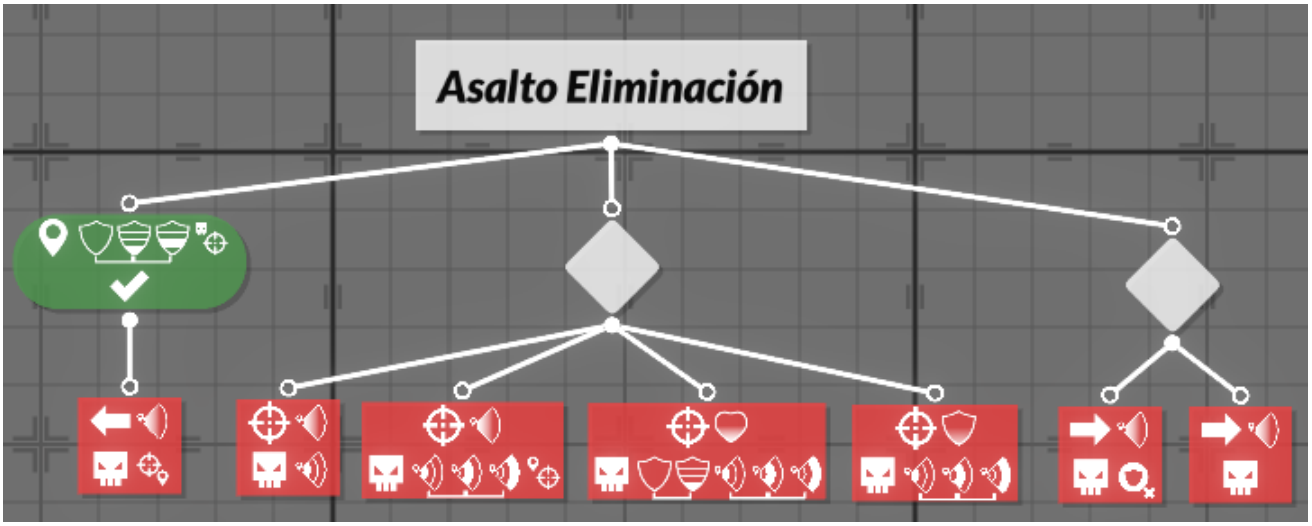


Figura 3.5. La IA "Asalto Eliminación".

Con esta IA conseguimos derrotar a los dos bots enemigos sin sufrir ninguna baja en nuestro escuadrón, y en un tiempo aproximado de 45 segundo. Como punto negativo, con esta IA nuestros robots se ven muy presionados por los enemigos, que tiende a alejarlos de los campos de fuerza. Talvez se os ocurra una forma más eficiente de ganar la batalla. De nuevo, recordad que se puntuará al alza aquellas IAs que consigan resultados más ventajosos (mínimo número de bajas, mínimo tiempo hasta la victoria, etc.).

ELIMINACIÓN: CAPÍTULO 2, NIVEL 2.

En este nuevo nivel, tres de nuestros robots se enfrentan a tres bots de asalto enemigos. De nuevo, la arena de combate incluye tres campos de fuerza en posiciones simétricas.

Tener en cuenta que, aunque el escuadrán enemigo está formado por tres bots de asalto, nuestro escuadrón no tiene por qué hacerlo. En este caso, podemos elegir que uno de nuestros bots sea de las clases escopeta, ametralladora, o francotirador.

Con la IA previa conseguimos ganar la batalla en unos 4 minutos, sin ninguna baja en uestro escuadrón, y presionando al conjunto enemigo, que queda acorralad contra una pared. Sin embargo, y de nuevo, os animamos para que uséis vuestras propias IAs, para que las optimicéis, y para que uséis otro tipo de robots.

ELIMINACIÓN: RESTO DE CAPÍTULOS Y NIVELES.

Como en los modos de juego previos, la figura 3.6 muestra una IA inicial optimizada para operar en el modo Eliminación. Como siempre, esta IA será nuestra guía para aprender a manejarnos en el modo Eliminación, y constituirá la base sobre la que construiremos nuestras futuras IAs opimizadas y refinadas para superar los subsiguientes niveles y capítulos.

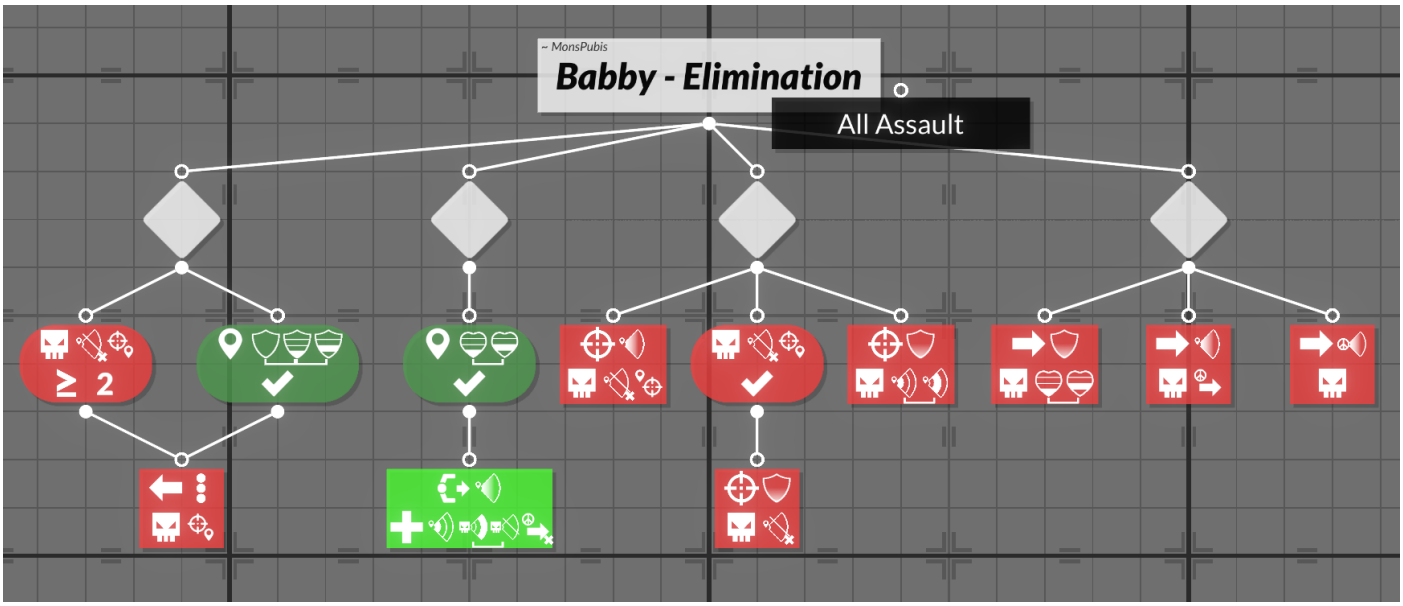


Figura 3.6. "Babby - Elimination", una IA optimizada para los primeros niveles del modo Eliminación.

La IA "Babby - Elimination" está diseñada para controlar un escuadrón de robots de tipo Asalto, y consta de 4 ramas. De izquierda a derecha, las ramas especifican lo siguiente: (1) Si el número de bots enemigos que me estén atacando y que no estén fuera de alcance es mayor o igual que dos, o si mi escudo ha caído por debajo del 50%, el robot huye de todos los bots enemigos que le estén atacado (los tres puntos del nodo especifica "todos"). (2) Si su nivel de vida está por debajo del 50%, el robot tratará de recoger el paquete de salud más cercano que se localice a corta distancia de él, a larga distancia o fuera del alcance de los enemigos, y al que no se esté aproximando ya uno de los robots aliados. (3) Esta tercera rama es la de ataque, que de izquierda a derecha, especifica lo siguiente: Primero, el robot tratará de atacar al bot

enemigo más cercano que no esté fuera de alcance y al que ya estuviese atacando antes. A continuación, y si hay un bot enemigo que le esté atacando y que no esté fuera de alcance, el robot ataca al bot enemigo con el escudo más débil y que no esté fuera de alcance. Por último, elrobot ataca al bot enemigo de escudo más débil que se encuantre a corta o media distancia. (4) Para terminar, el robot procede como sigue: Primero intenta acercarse al bot enemigo con ele scudo más débil y con un nivel de vida por debajo del 50%. Después trata de acercarse al bot enemigo más cercano al que se estén acercando el resto de aliados. En otro caso, el robot tratará simplemente de acercarse al bot más cercano.

3.13. EL MODO MULTIJUGADOR.

Para acceder al modo Multijugador acudimos a la ventana principal de Galdiabots, y pulsamos en la pestaña "Multijugador". La figura 4.1 muestra una vista de la ventana Multijugador.

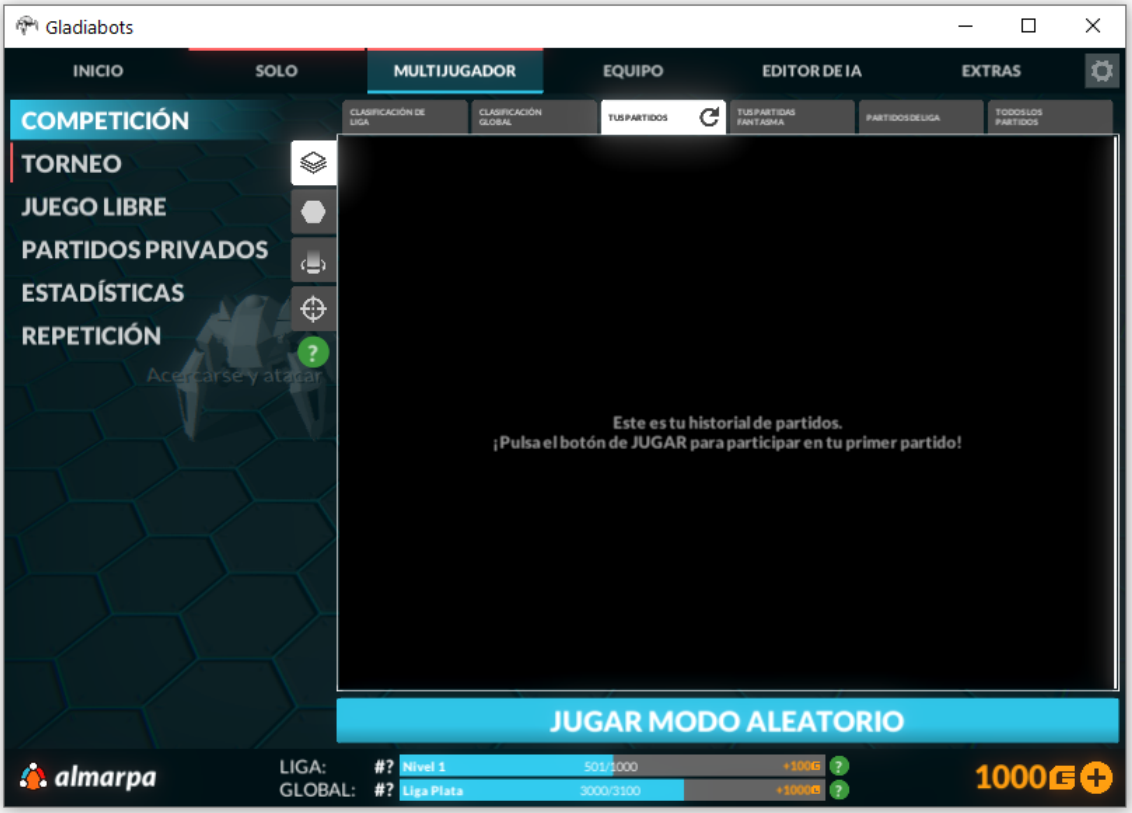


Figura 4.1. La ventana Multijugador.

Existen varias formas de jugar a Gladiaboots en modo Multijugador, que pasamos a describir brevemente:

Competición. Esta opción nos permite competir con otros usuarios de Gladiabots. Los resultados de las distintas partidas nos permitirán ganar puntos de experiencia (XP) que determinarán nuestra posición en la clasificación global de Gladiabots y en las distintas ligas.

Torneo. El modo torneo nos permite participar en torneos oficiales y de la comunidad de jugadores de Gladiabots.

Juego libre. En el modo de juego libre podemos organizar partidas sin puntuación con otros jugadores. Estas partidas no clasificatorias nos darán puntos de XP que no tendrán efecto en nuestra clasificación global.

PARTIDOS PRIVADOS. Esta opción nos permite jugar partidas privadas con amigos u otros jugadores de la comunidad. Éste es el modo de juego Multijugador que utilizaremos para enfretarnos entre nosotros y comprobar qué alumn@ ha construido la IA más eficiente.

Estadísticas. Esta opción establece una conexión con la web https://stats.gladiabots.com donde se almacenan las estadísticas del jugador.

Repetir. Opción que nos permite repetir partidas ya jugadas. Para seleccionar una partida, necesitamos consutar la ID de esa partida en el **historial de partidas** (la lista de las partidas en las que está participando o ha participado el jugador), o en la **web de estadísticas**.

3.14. ORGANIZAR UNA PARTIDA PRIVADA.

CREAR UNA CUENTA DE USUARIO.

Para organizar una partida privada con otro usuario de gladiabots, primero necesitamos que ambos usuarios dispongan de una cuenta que los identifique. Al instalar Gladiabots, el programa arranca sin estar asignado a ninguna cuenta; es por ello que en la esquina inferior derecha aparece un símbolo de interrogación. Al crear una cuenta, el símbolo de interrogación se sustituye por el nombre de usuario de esa cuenta ()en mi caso, almarpa). Para crear una cuenta de Gladiabots, acudimos al menú de configuración (botón del engranaje), expandimos la lista de opciones de Cuenta, y seleccionamos la opción Crear cuenta. Para crear una cuenta, elegimos un nombre de usuario (en mi caso, almarpa), y opcinalmente, proporcionaamos una dirección de correo electrónico. Vuestros ordenadores ya deberían tener creadas las cuentas de Galdiabots, con nombres de usuario PC1, PC2, PC3, etc. De no ser así, avisad al profesor para crear una cuenta y asignarle un no,bre de usuario correcto.

CREAR UNA NUEVA PARTIDA PRIVADA.

Para que nuestro escuadrón de robots se enfrente al escuadrón de robots de un compañero de clase, debemos crear una partida privada para esos dos usuarios. Para crear una partida privada, acudimos al modo Multigugador y seleccionamos la opción "Partidos privados". Se abrirá una ventana como la mostrada en la figura 4.2.

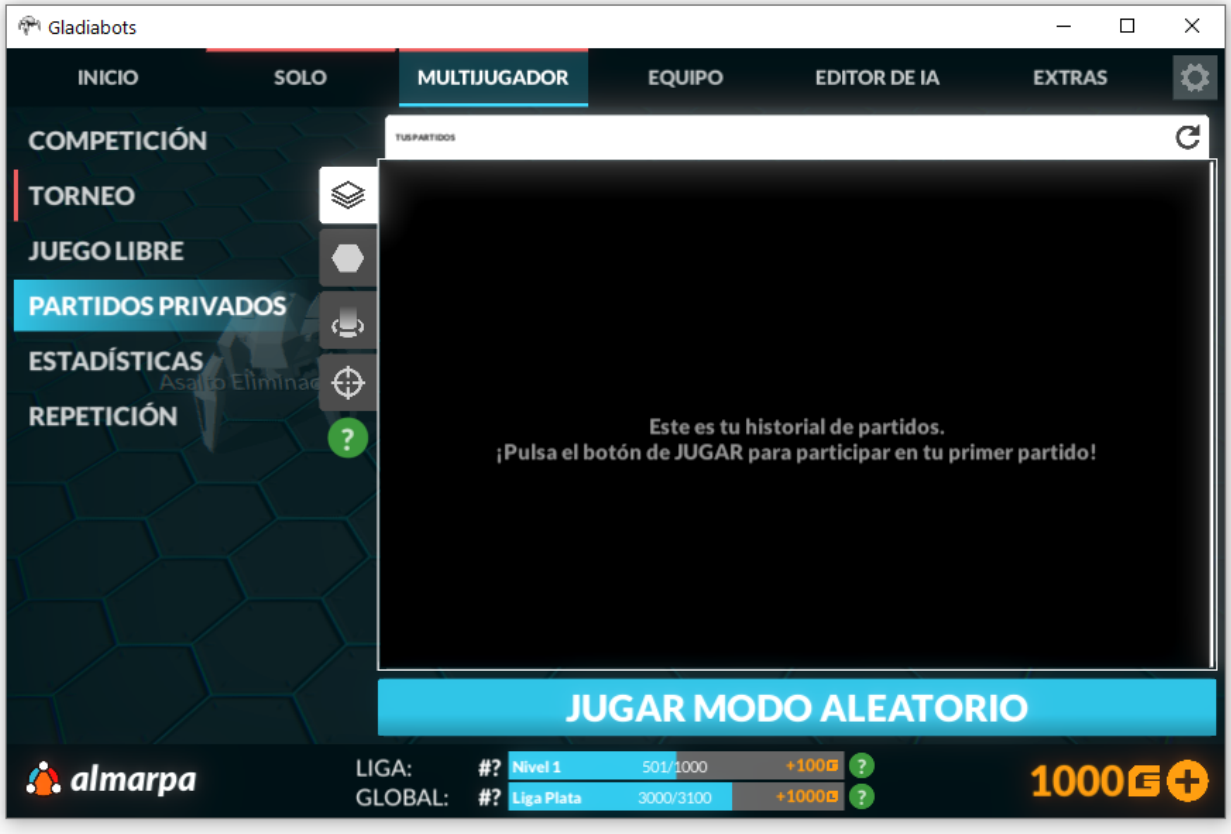


Figura 4.2. Partidas privadas.

La ventana de partidas rivadas disponde de 4 pestañas que nos permiten acceder a una partida privada en modo aleatorio, en modo Colección, en modo Dominación, o en modo Eliminación. El botón de la interrogación se conecta a la wiki de Gladiabots (https://wiki.gladiabots.com/index.php?title=Game_modes) para recordarnos en qué consiste cada modo de juego. Una vez elegido un modo de juego, la ventana central muestra una lista con nuestras partidas privadas en curso, y el botón azul alargado nos permite empezar una nueva partida en el modo seleccionado (aleatorio, colección, dominación, o eliminación).

A modo de ejemplo, voy a crear una partida privada en modo Dominación con el usuario PC2 (ver la serie de figuras a continuación). Para ello pulso en la pestaña del modo Dominación. La ventana central aparece vacía porque, hasta la fecha, no había creado ninguna partida privada en modo dominación. Pulso en el botón azul alargado de "Jugar en modo Dominacion", tras lo cual aparece una ventana que me solicita el nombre del usuario con el que quiero jugar esta partida Multigugador: Escribo PC2 y pulso OK.

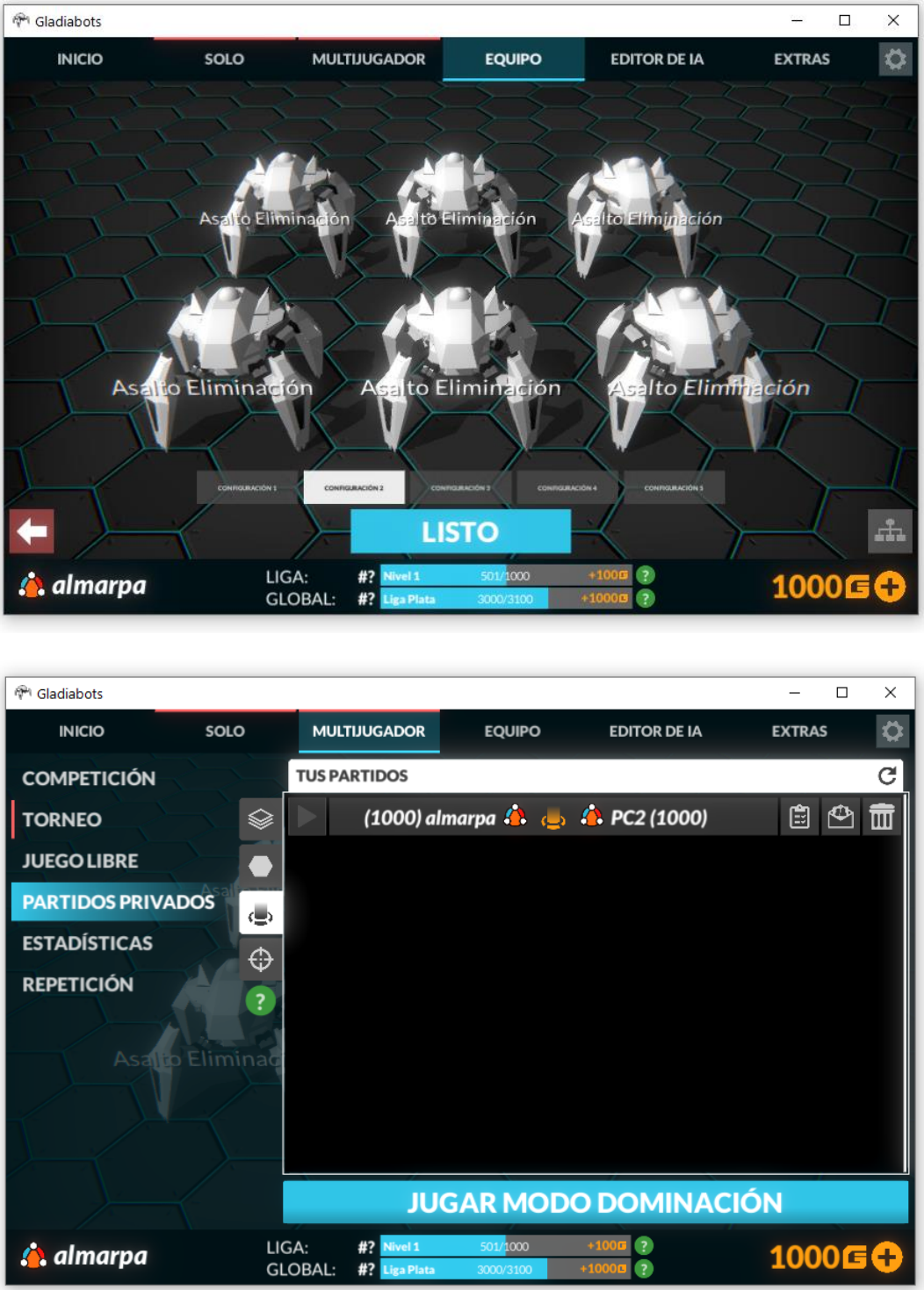


Figura 4.3. Proceso de creación de una partida privada.

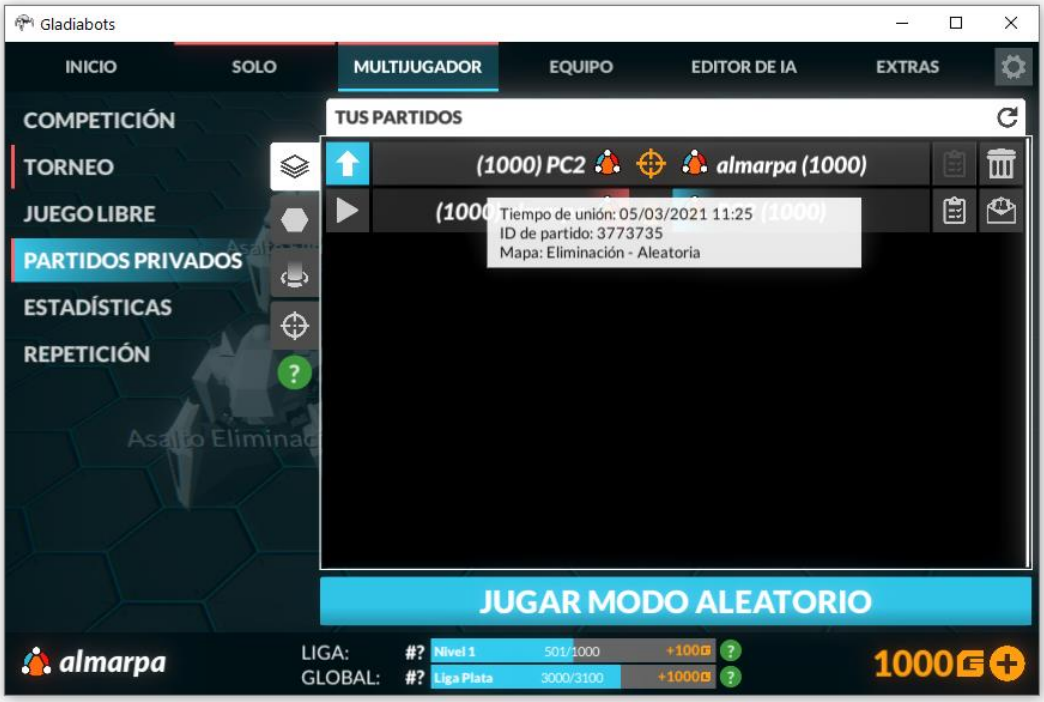
Tras esto, aparece la ventana donde puede configurar los robots de mi escuadrón, escibir las IAs de mis robots, etc. Al pulsar el botón LISTO, la partida se crea y aparece en la ventana de partidas privadas disponibles en modo Dominación. En la ventana de partidas privadas del usuario PC2 debería aparecer la misma partida ya creada.

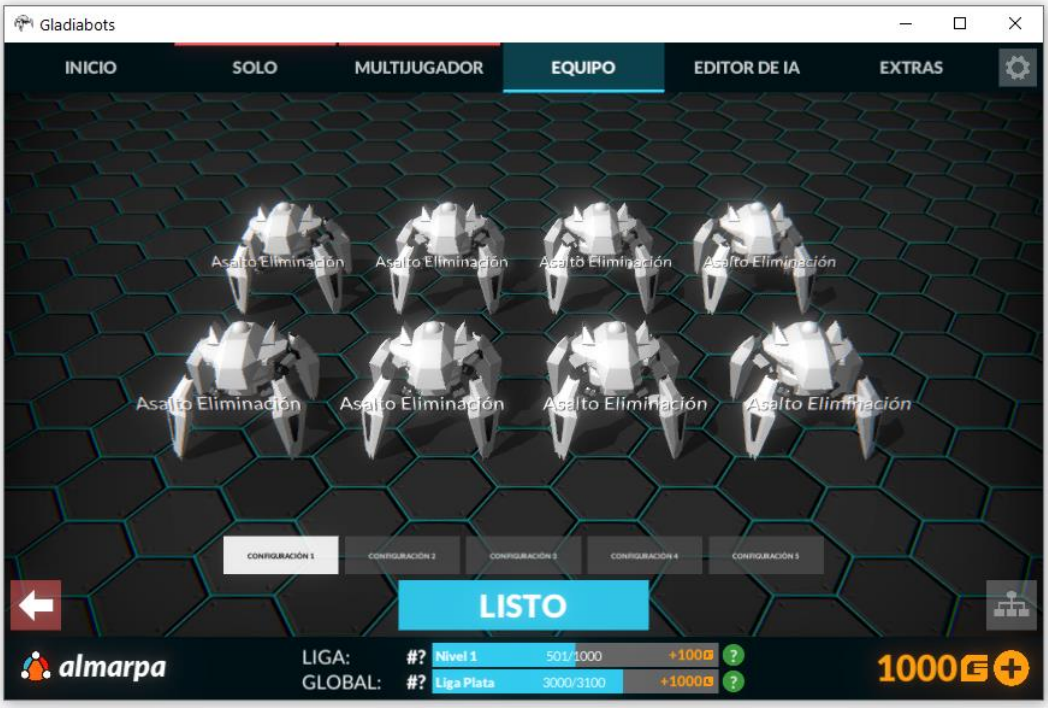
JUGAR UNA PARTIDA PRIVADA.

El modo de juego Multijugador es asíncrono, lo que significa que los usuarios de una partida multijugador no la juegan al mismo tiempo.

En el proceso de creación de la partida, el usuario que la creó ya configuró su escuadrón de robots y escribió las IAs correspondientes. Una vez creada la partida, ésta aparece en el historial de partidas del

otro usuario, el cual puede pasar a configurar su equipo y a escribir las IAs de su escuadrón. A modo de ejemplo, la serie de figuras a continuación muestran el proceso a seguir para jugar una partida en modo Eliminación. Esta partida fue creada por el usuario PC2 para combatir contra el usuario almarpa. Las figuras muestran las ventanas del usuario invitado (almarpa). Como vemos, laa partida creada por PC2 aparece en el historial de partidas de almarpa. Al seleccionar la opción Desplegar (botón azul con la flecha hacia arriba), el usuario almarpa entra en la ventana donde puede configurar su equip de robots y escribir las IAs de control (recoordemos que el usuario PC2 ya seleccionó la configuración de su equipo y proporcionó las IAs de control en el proceso de creación de la partida). Cuando el usuario almarpa termina y pulsa LISTO, el combate comienza y los escuadrones empiezan a batallar (el escuadrón de almarpa en azul y el de PC2 en rojo). En esta partida, PC2 estaba jugando con un escuadrón de robots de tipo Asalto, igual que almarpa. La IA de los robots de PC2 era la IA del Starter Kit que Gladiabots proporciona por defecto para el modo Eliminación, llamada "_Elimination" (ver figura a continuación), mientras que en mi caso (almarpa) utilicé la IA llamada "Asalto Eliminación" (ver figura 3.5). El resultado es una rotunda victoria del equipo de almarpa por 8 a 0, en menos de un minuto.





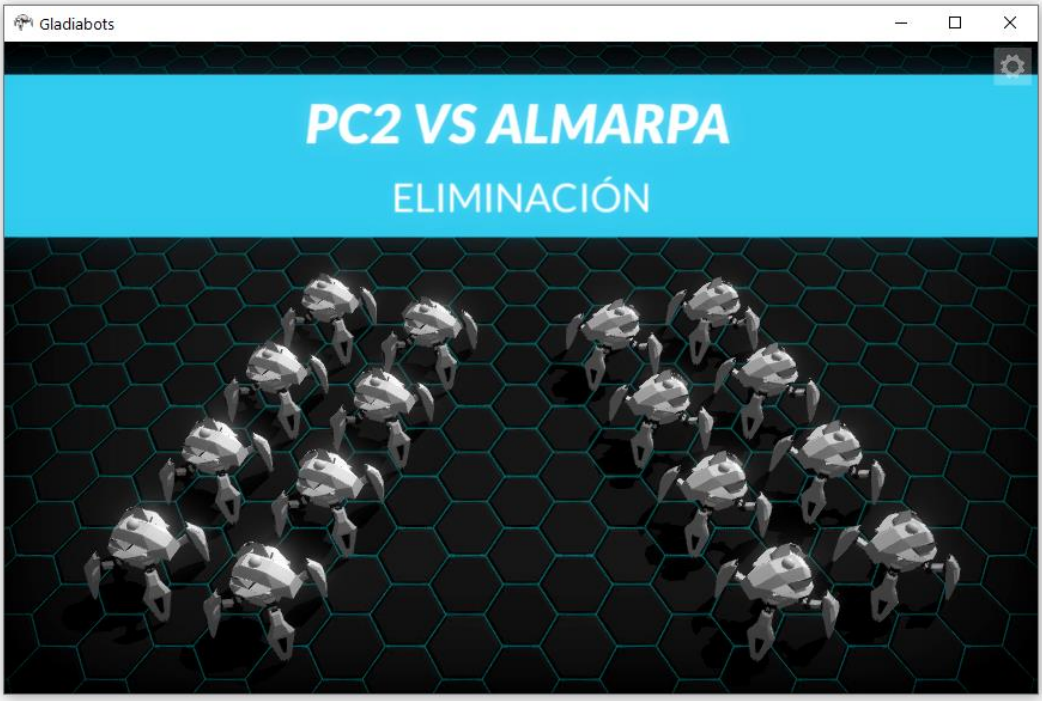
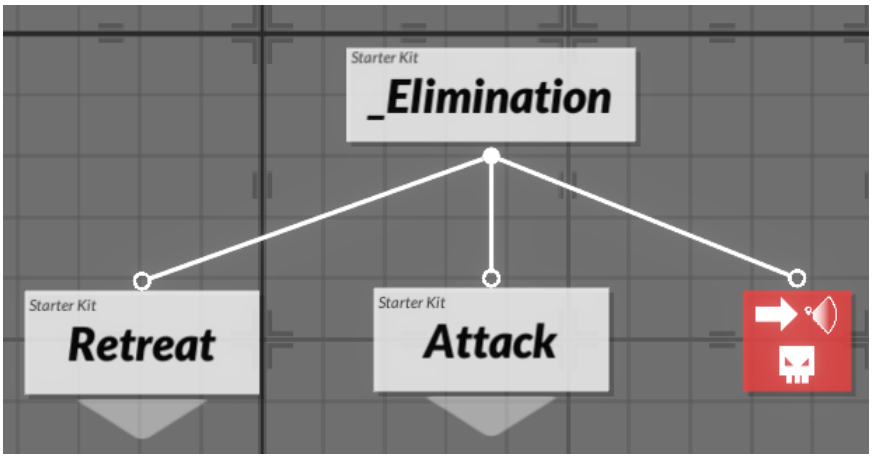


Figura 4.4. Jugar una partida privada en modo Multijugador.



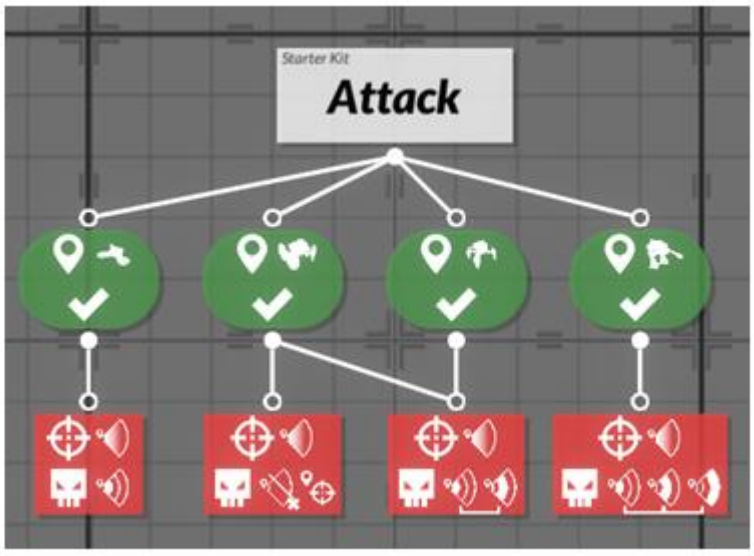


Figura 4.5. La IA "_Elimination" del equipo "PC2".

Figura 4.6. Desarrollo y resultado de la partida privada entre PC2 y almarpa.

3.15. COMPETICION ENTRE ALUMNOS.

Nosotros vamos a utilizar el modo Multijugador en su versión de partidas privadas para enfrentar a los escuadrones de los alumnos de la asignatura.

El profesor organizará el torneo entre los alumnos, ya sea en forma de liga, o de eliminatorias dos a dos. Se organizarán tres competiciones distintas, una para cada modo de juego (Colección, Dominación, y Eliminación).

El vencedor de cada torneo recibirá el reconocimiento de la clase, y lo que es más importante, puntuación extra en la nota final de la unidad (el valor de esta puntuación extra dependerá del número de alumnos en

competición. Si el número de participantes es muy elevado, el profesor podría recompensar al segundo y tercer clasificados, al finalista y los semifinalistas, etc.).

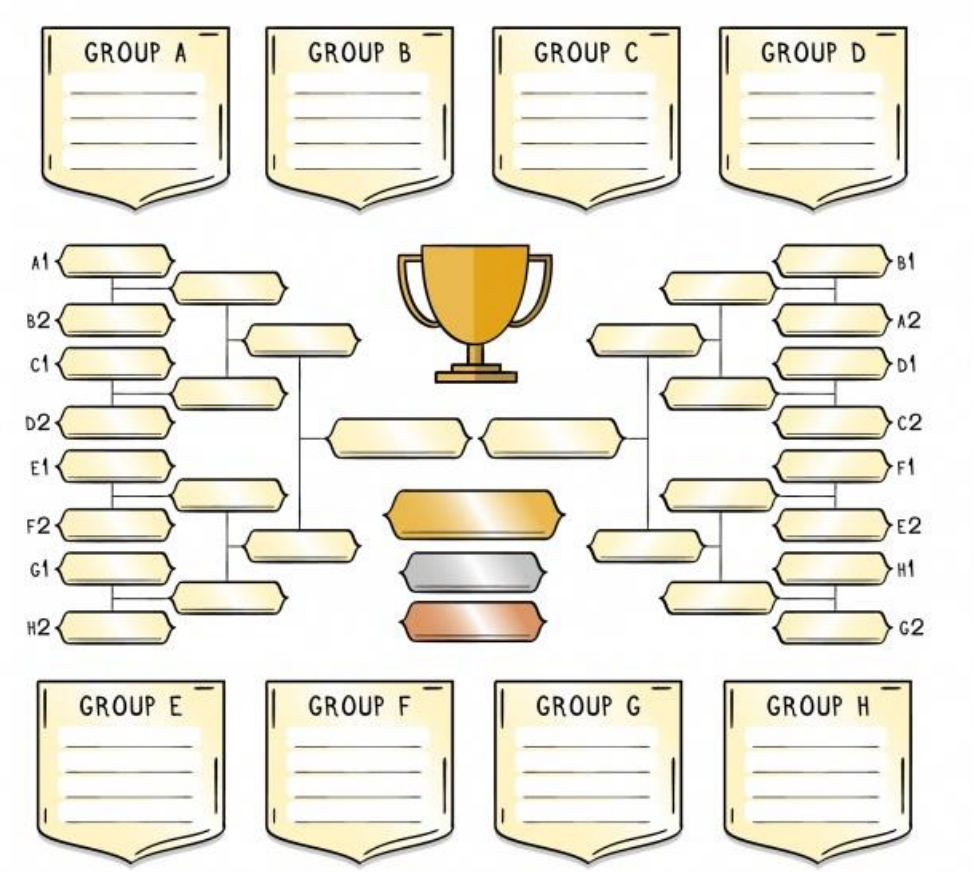
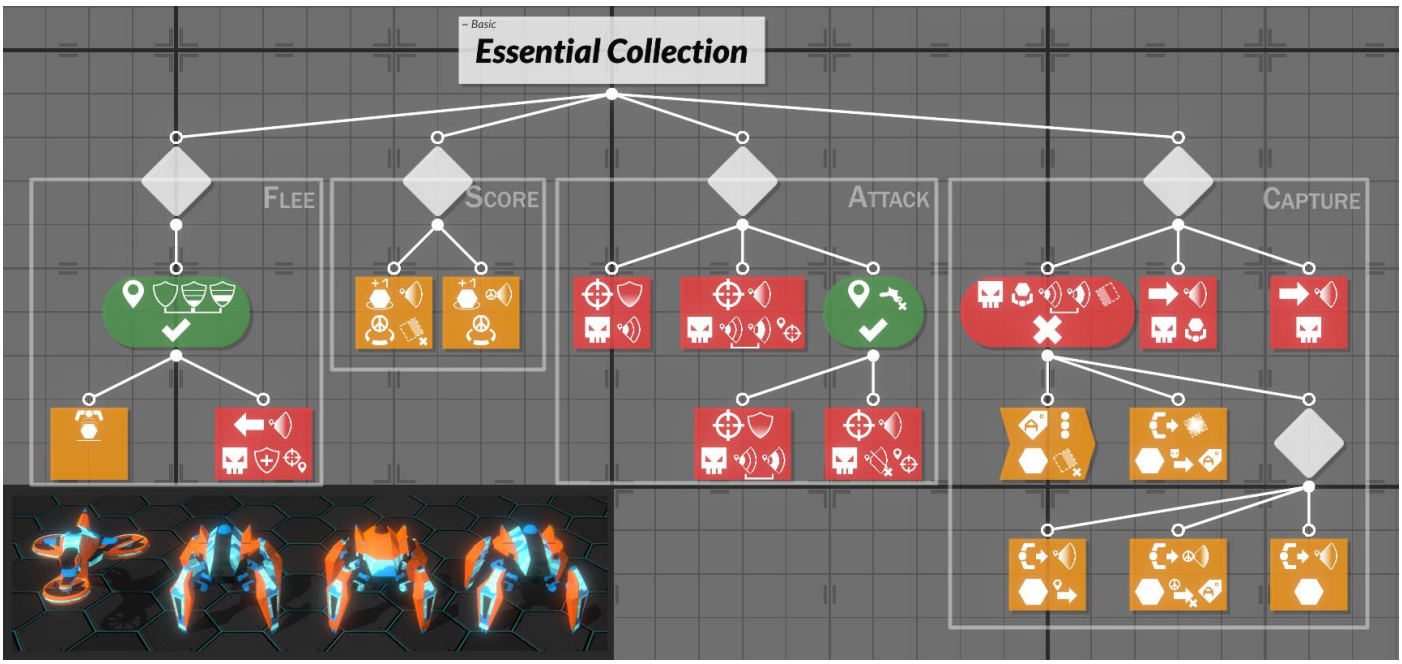


Figura 4.5. Competición.

3.16. OTRAS IAs BÁSICAS.

COLECCIÓN.

Para terminar con el tema, y a modo de anexo, se adjuntan otras IAs básicas en las que nos podemos basar para construir nuestras propias IAs para los modos Colección, Dominación, y Eliminación. (Fuente: https://wiki.gladiabots.com/index.php?title=Strategies_SimpleBots).



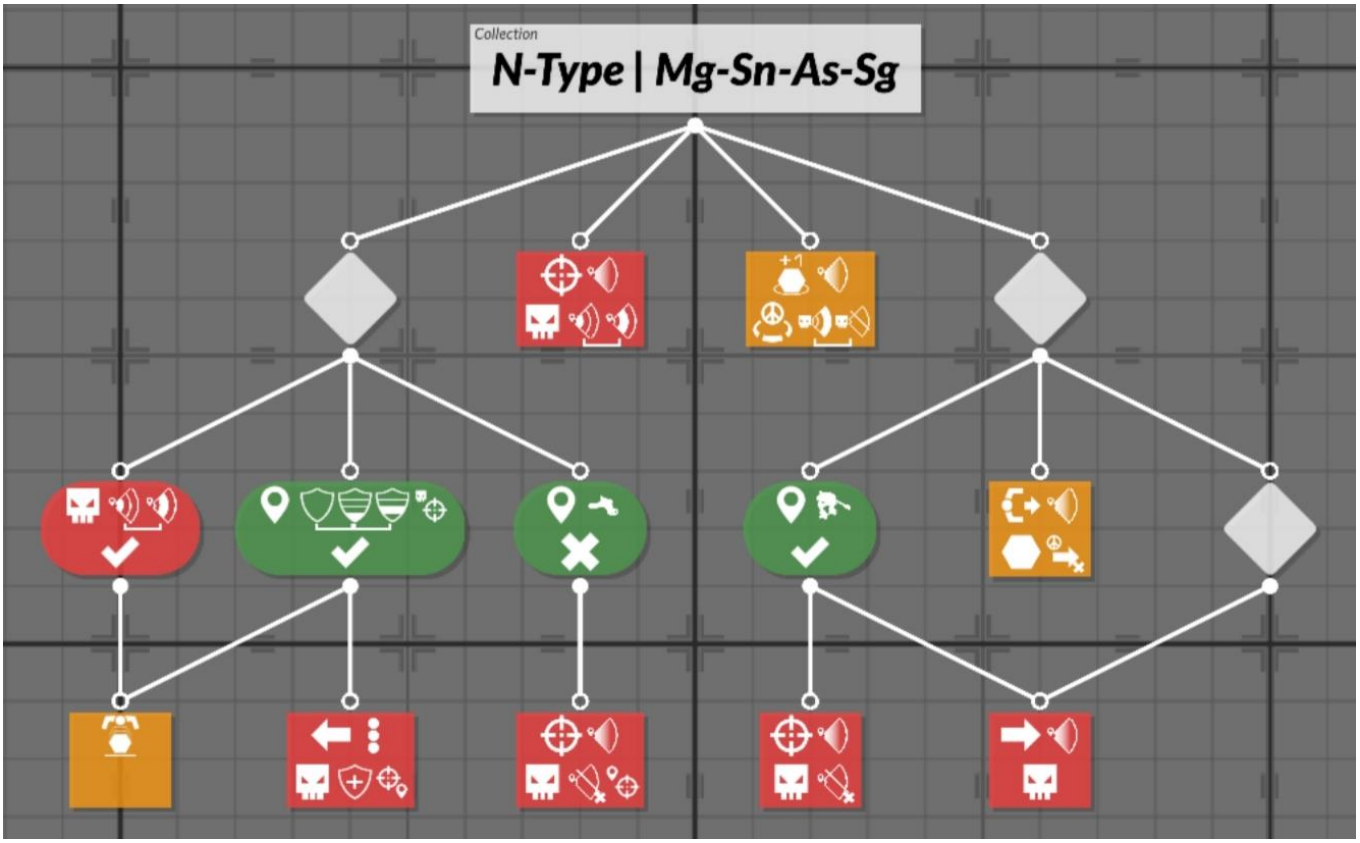
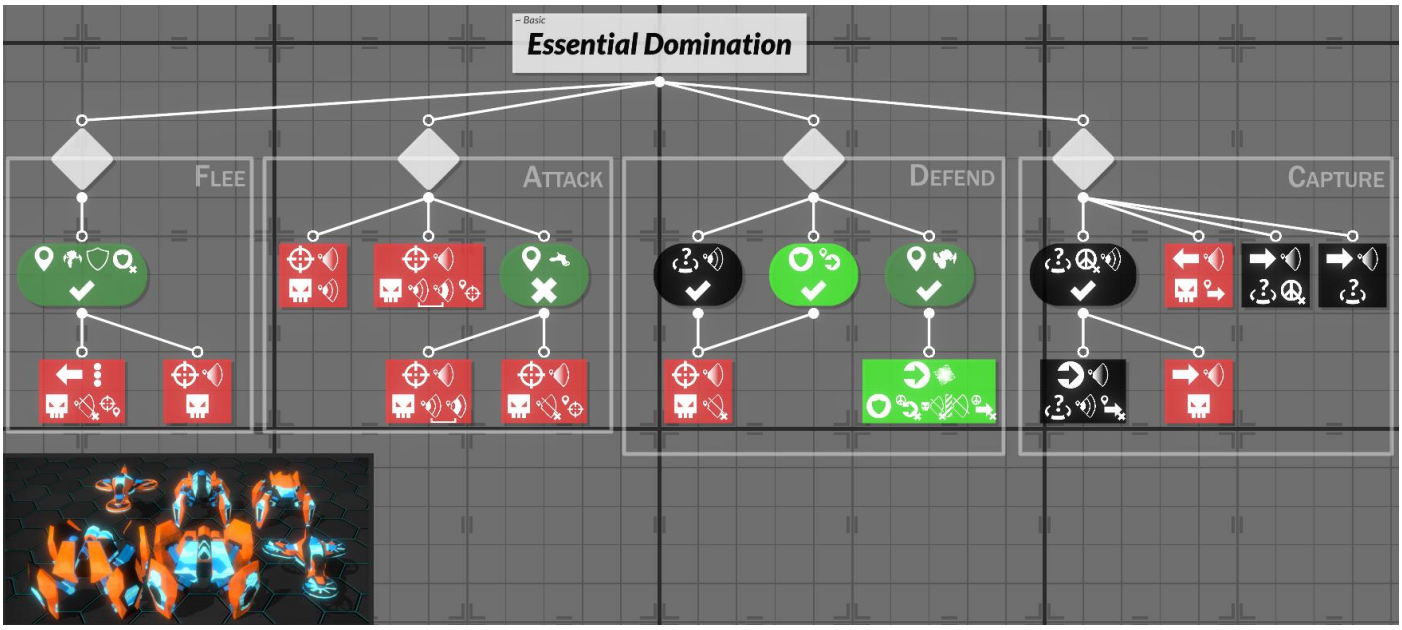


Figura 4.6. Otras IAs básicas para el modo Colección.

DOMINACIÓN.



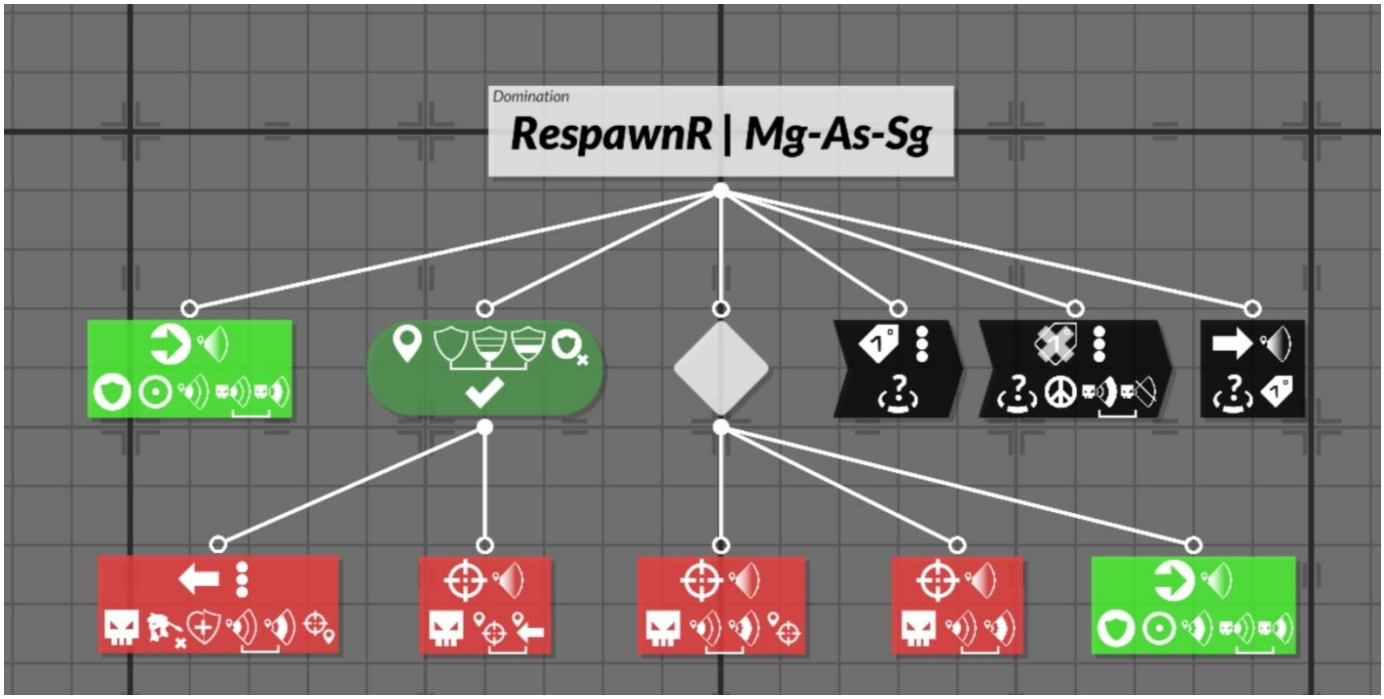
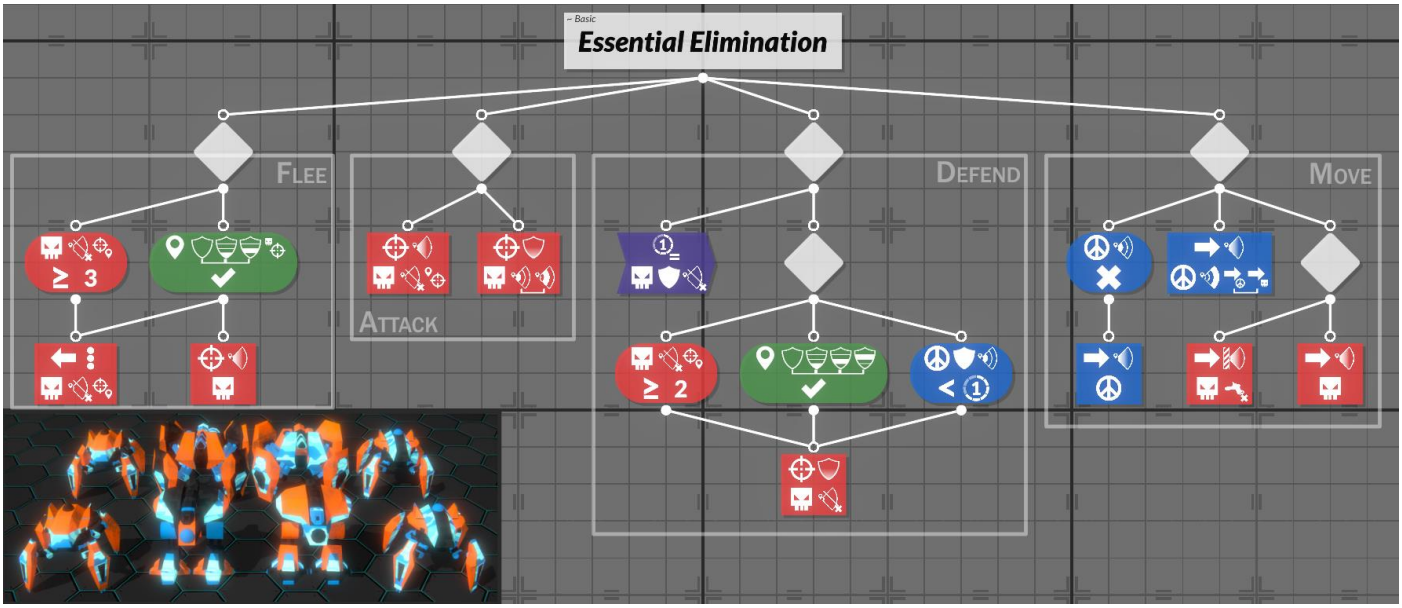


Figura 4.7. Otras IAs básicas para el modo Dominación.

ELIMINACIÓN.



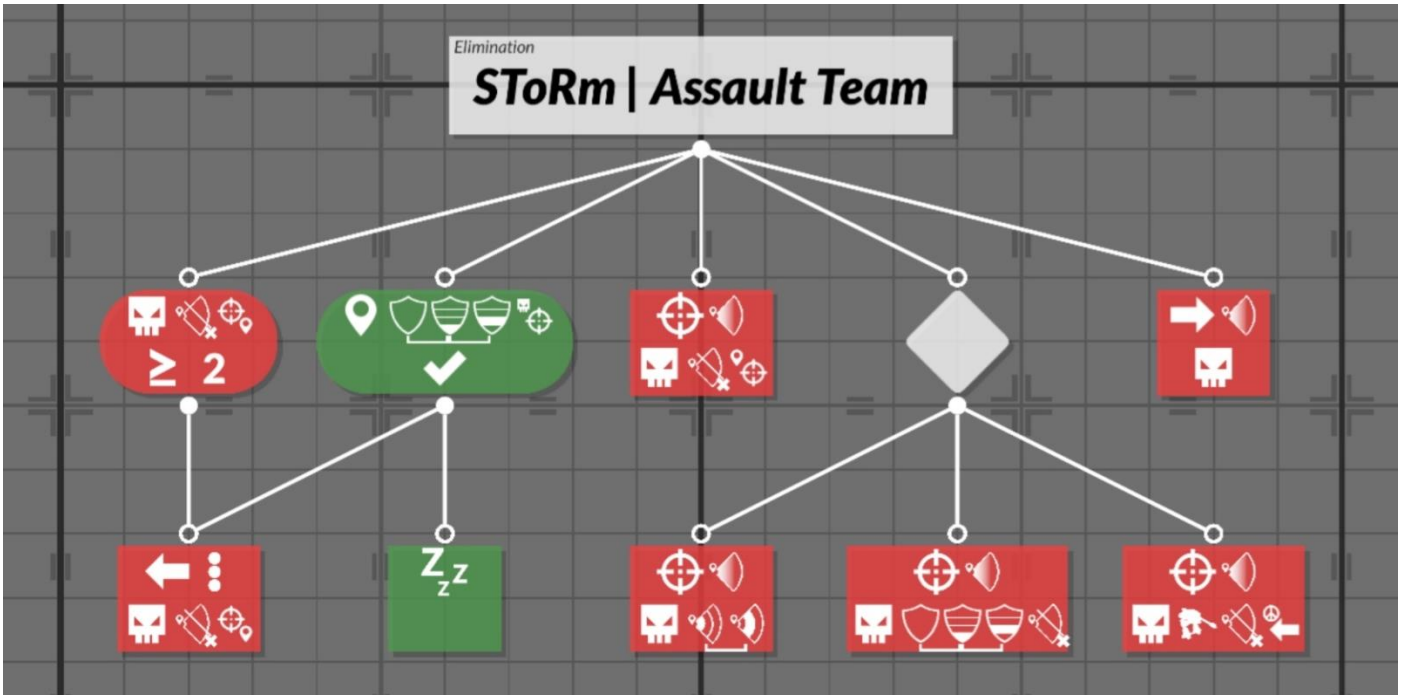


Figura 4.8. Otras IAs básicas para el modo Eliminación.

ANEXO A. LA INFORMACIÓN DIGITAL.

Para comunicar información, las personas utilizamos diferentes sistemas de representación de la información. Algunos ejemplos son los gestos (levantar la mano para pedir permiso para hablar, el lenguaje de signos, etc.), sonidos (el silbato de un árbitro para indicar una falta, el lenguaje hablado, etc.), imágenes (banderas de los barcos), o símbolos (el alfabeto, las señales de tráfico, etc.). Gracias a estos sistemas de representación de la información, los seres humanos podemos construir mensajes, y con ellos comunicar información, recibir información, y tratarla en nuestro cerebro (pensar).

Sin embargo, un ordenador es incapaz de entender estas formas de representar la información. Los ordenadores son máquinas compuestas por dispositivos electrónicos como interruptores, diodos, transistores, etc. En general, estos componentes pueden adoptar dos estados (abierto o cerrado, conduciendo o no conduciendo, activado o desactivado, etc.) que suelen traducirse en presencia o ausencia de tensión o corriente. Por consiguiente, para un ordenador la forma más sencilla de representar la información es mediante dos símbolos: cero (0) y uno (1). El símbolo 0 puede codificarse como una ausencia de corriente, $0\,V$ de tensión, un interruptor abierto, un diodo no conduciendo, o un transistor desactivado. El símbolo 1 puede codificarse como una presencia de corriente, $5\,V$ de tensión, un interruptor cerrado, un diodo conduciendo, o un transistor activado.

Así pues, la unidad más pequeña de información que un ordenador puede almacenar y procesar es la que corresponde a un evento en el que sólo hay dos alternativas posibles. Esta unidad mínima de información puede representarse mediante un único dígito del sistema binario que solo puede tomar los valores 0 ó 1. A este dígito se le denomina **bit**, abreviatura de las palabras Binary digIT (dígito binario).

Con un bit solo podemos representar informaciones con dos valores, como Sí o No, Blanco o Negro, Arriba o Abajo, Par o Impar, etc. Para denotar un bit se utiliza el símbolo b (por ejemplo, $16\,b$ significa 16 bits). La necesidad de codificar informaciones cada vez más complejas implica agrupar varios bits, siguiendo un orden determinado. Por ejemplo, si juntamos dos bits podemos representar cuatro informaciones distintas (por ejemplo, las cuatro direcciones cardinales) mediante las combinaciones 00, 01, 10, 11. Con tres bits podemos representar ocho informaciones distintas, con cuatro bits 16 informaciones, y así sucesivamente. En general, con n bits podemos representar el número de informaciones dados por la fórmula:

$$nº\ informaciones\ independientes = 2^n$$

Un **nibble** es un conjunto de 4 bits que permite representar hasta $2^4 = 16$ valores distintos. Por ejemplo, con un nibble podríamos codificar los doce meses del año, y aún nos sobrarían cuatro combinaciones. Un nibble suele representarse mediante su símbolo correspondiente en el sistema numérico hexadecimal (ver tabla A.1). Por ejemplo, el nibble 1101 se corresponde con el valor decimal 13, que en hexadecimal se representa con el símbolo D.

Un **byte** es un conjunto de 8 bits, con el que se pueden representar hasta $2^8 = 256$ informaciones distintas. Notar que un byte está compuesto por dos nibbles. Por ejemplo, el byte 10100111 está formado por los nibbles 1010 (A) y 0111 (7). Este byte se representa en código hexadecimal como $A7$. El término byte es la contracción de las palabras inglesas BinarY TErm (término binario) y se denota mediante el símbolo B (por ejemplo, $1024\,B$ significa 1024 bytes).

Los múltiplos del byte son el **kilobyte** (kB), que equivale a $2^{10} = 1024$ bytes; el **megabyte** (MB), que equivale a $2^{20} = 1024$ kilobytes; el **gigabyte** (GB), que equivale a $2^{30} = 1024$ megabytes; el **terabyte** (TB), que equivale a $2^{40} = 1024$ gigabytes, etc. El motivo por el que la proporción entre los distintos múltiplos es de 1024 en lugar de 1000 (que es lo habitual en el sistema decimal) se debe a que 1024 es la potencia de la base 2 que más se aproxima a 1000 (10^3). Es por ello que $2^{10} = 1024$ es el valor que equivale al prefijo

"kilo". Lo mismo ocurre con los demás prefijos (M, G, T, etc.). En consecuencia, el factor de multiplicación es 1024 veces el valor anterior, en lugar de 1000.

Binario	Decimal	Hexadecimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Tabla A.1. Correspondencia de un nibble (4 bits binarios) con sus valores decimal y hexadecimal.

ANEXO B. TABLA ASCII.

Decimal - Binary - Octal - Hex – ASCII Conversion Chart

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

ANEXO C. SISTEMA NUMÉRICOS POSICIONALES.

C.1. PRINCIPALES SISTEMAS NUMÉRICOS.

Los sistemas numéricos posicionales usan conjuntos de símbolos para representar números. Sin embargo, el valor de cada símbolo depende tanto de su **valor nominal** (*face value*) como de su **valor posicional** (*place value*). El valor nominal es el valor inherente del propio símbolo, y el valor posicional es el valor asociado a la posición que ocupa en el número del que forma parte. En otras palabras, tenemos que:

$$\text{Valor del símbolo} = \text{Valor nominal} \times \text{Valor posicional}$$

$$\text{Valor del número} = \text{Suma de los valores de todos los símbolos}$$

En este anexo solo discutiremos los enteros (esto es, los números sin parte decimal). El tratamiento de los números reales sería muy similar. En primer lugar veremos la forma en la que los enteros pueden representarse en cuatro sistemas numéricos distintos: Base 10, base 2, base 16, y base 256.

BASE 10: DECIMAL.

El primer sistema posicional que discutiremos se denomina **sistema decimal**. El sistema decimal usa 10 símbolos (0, 1, 2, 3, 4, 5, 6, 7, 8, y 9) con los mismo valores nominales que los símbolos. En el sistema decimal, los valores posicionales son potencias de 10. La figura B.1 muestra los valores posicionales y los valores de los símbolos en el entero 4782:

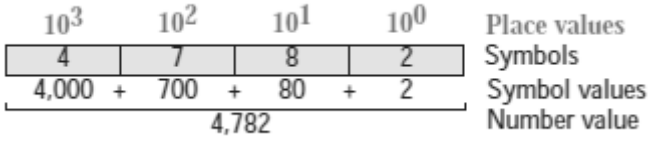


Figura C.1. Un ejemplo de número decimal.

BASE 2: BINARIO.

El segundo sistema posicional que discutiremos se denomina **sistema binario**. El sistema binario usa dos símbolos (0 y 1) con los mismos valores nominales que los símbolos. En el sistema numérico binario, los valores posicionales son potencias de dos. La figura B.2 muestra los valores posicionales y los valores de los símbolos en el número binario 1101_2. Notar que usamos el subíndice 2 para indicar que ese número está escrito en sistema binario.

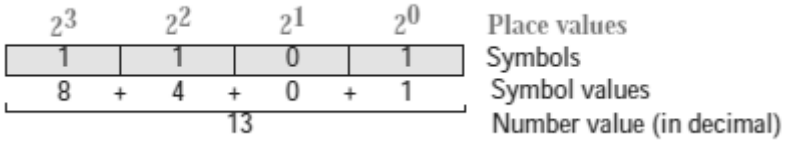


Figura C.2. Un ejemplo de número binario.

BASE 16: HEXADECIMAL.

El tercer sistema posicional que estudiaremos es el **sistema hexadecimal**. El sistema hexadecimal utiliza 16 símbolos (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, y F). El valor nominal de los diez primeros símbolos es el mismo que los símbolos, pero los valores nominales de los símbolos del A al F son de 10 a 15, respectivamente. En el sistema hexadecimal, los valores posicionales son potencias de 16. La figura B.3

muestra los valores posicionales y los valores de los símbolos en el número hexadecimal $A20E_{16}$. Notar que el subíndice 16 indica que el número es hexadecimal.

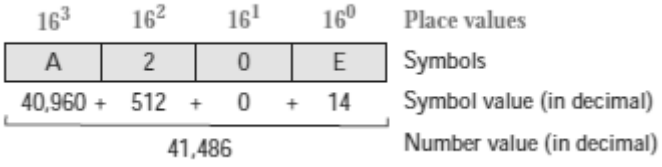


Figura C.3. Un número hexadecimal.

BASE 256: NOTACIÓN DECIMAL CON PUNTOS.

El cuarto sistema posicional que trataremos es el base 256, al que se denomina **notación decimal con puntos**. Este sistema se usa en el direccionamiento IPv4. En este sistema, los valores posicionales son potencias de 256. Sin embargo, como usar 256 símbolos sería del todo inviable, los símbolos de este sistema son los números decimales del 0 al 255, con el mismo valor nominal que los símbolos. Para separar un símbolo de otro, este sistema usa puntos. La figura B.4 muestra los valores posicionales y los valores de los símbolos de la dirección 14.18.111.252. Notar que nunca se usan más de cuatro símbolos en una dirección IPv4.

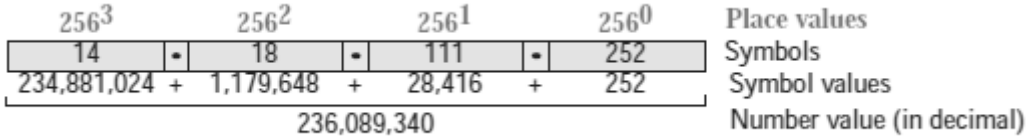


Figura C.4. Ejemplo de notación decimal con puntos.

COMPARATIVA.

La tabla B.1 muestra la forma en la que los tres primeros sistemas numéricos representan los números decimales del 0 al 15. Por ejemplo, el decimal 13 es equivalente al binario 1101, y al hexadecimal D.

Decimal	*Binary*	*Hexadecimal*	*Decimal*	*Binary*	*Hexadecimal*
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

Tabla C.1. Comparativa entre los tres sistemas numéricos.

C.2. CONVERSIÓN ENTRE SISTEMAS.

Lo siguiente que debemos aprender es cómo convertir un número de un sistema a otro.

CONVERSIÓN DE CUALQUIER BASE A DECIMAL.

Las figuras B.2 a B.4 muestran la forma en la que podemos convertir manualmente un número de cualquier base a decimal. Sin embargo, en la práctica es más sencillo usar el algoritmo mostrado en la figura B.5. El algoritmo usa el hecho de que el siguiente valor es el valor previo multiplicado por la base (2, 16, o 256).

Este algoritmo es general, y puede usarse para convertir una cadena de símbolos escrita en cualquier base a un número decimal. La única parte del algoritmo que es diferente de una base a otra es la forma en la que se extrae el siguiente símbolo de la cadena y se obtiene su valor nominal. En el caso de base 2, esto es fácil: El valor nominal puede obtenerse cambiando el símbolo a un valor numérico. En el caso de base 16 necesitamos considerar el hecho de que el valor nominal del símbolo A es 10, el del símbolo B es 11, etc. En base 256 tenemos que extraer cada cadena delimitada por puntos y cambiar la cadena a su valor numérico. No profundizaremos más en los detalles del algoritmo, porque dependen del lenguaje de programación que se utilice para implementarlo.

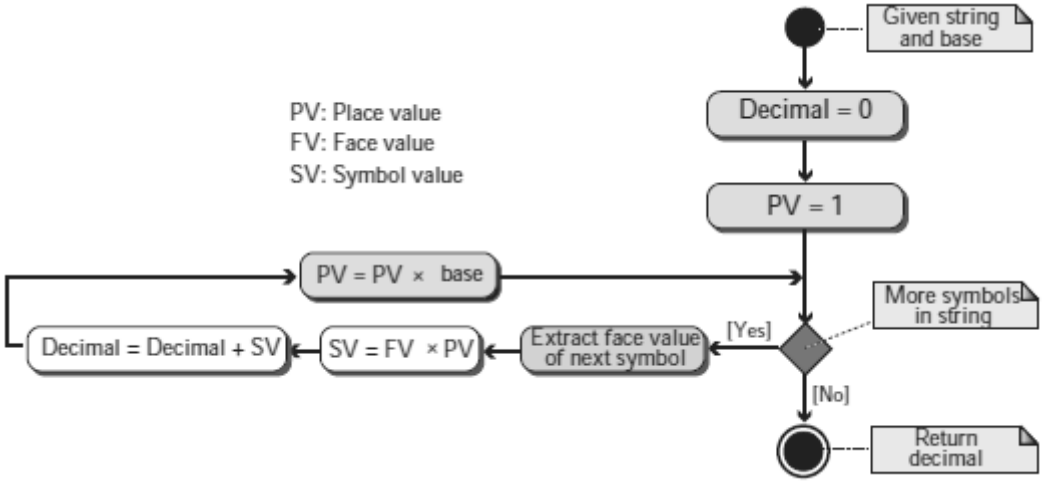


Figura C.5. Algoritmo para convertir de cualquier base a decimal.

Los siguientes ejemplos muestran la implementación manual del algoritmo para unos pocos números pequeños.

Ejemplo C.1. Para convertir el número binario 11100111_2 a decimal aplicamos el algoritmo previo como se muestra a continuación:

128		64		32		16		8		4		2		1	Place values
1		1		1		0		0		1		1		1	Face values
128		64		32		0		0		4		2		1	Symbol values
231		103		39		7		7		7		3		1	Decimal = 0

El valor del decimal se fija inicialmente a 0. Cuando el bucle termina, el valor del decimal es 231.

Ejemplo C.2. Para obtener el equivalente de la dirección IP 12.14.67.24 a decimal aplicamos el algoritmo como indicamos:

16,772,216		65,536		256		1	Place values
12	•	14	•	67	•	24	Face values
201,266,592		917,504		17,152		24	Symbol values
202,255,272		988,680		71,176		24	Decimal = 0

El valor del decimal se fija inicialmente a 0. Cuando se completa el bucle, el valor del decimal es 202255272.

CONVERSION DE DECIMAL A CUALQUIER BASE.

La conversión de un valor decimal a una base cualquiera se hace dividiendo continuamente el número decimal entre la base deseada para obtener el resto y el cociente. El resto es el valor nominal del siguiente símbolo; el cociente es el valor decimal a usar en la siguiente iteración. Como en el caso de la conversión inversa, existe un algoritmo para cambiar el valor nominal de un símbolo, en la base correspondiente, al símbolo actual, y para insertarlo en la cadena que representa el número convertido. La figura B.6 muestra el algoritmo:

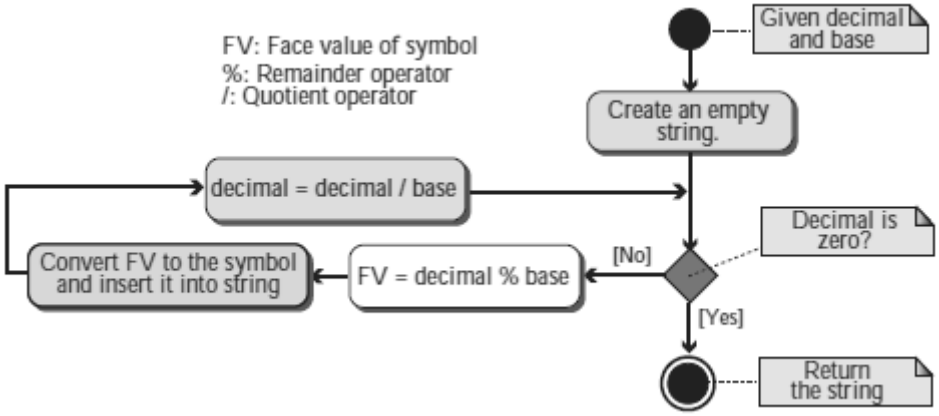


Figura C.5. Algoritmo para convertir un decimal a cualquier otra base.

Vamos a ver cómo aplicar el algoritmo de forma manual en unos pocos ejemplos.

Ejemplo C.3. Vamos a convertir el número decimal 25 a su equivalente binario. Para ello, dividimos continuamente el valor decimal entre 2 (la base del sistema binario) hasta que el cociente se haga 0. En cada división tomamos el valor del resto como el siguiente símbolo a insertar en la cadena binaria. Las flechas hacia abajo muestran el resto; las flechas hacia la izquierda muestran el cociente. Cuando el valor decimal llega a 0, nos paramos. El resultado es la cadena binaria 11001_2.

0	←	1	←	3	←	6	←	12	←	25	Decimal
		↓		↓		↓		↓		↓	
		1		1		0		0		1	Binary

Ejemplo C.4. Ahora vamos a convertir el número decimal 21432 a su equivalente hexadecimal. Para ello, dividimos continuamente el valor decimal entre 16 (la base del sistema hexadecimal) hasta que el cociente se hace 0. En cada división tomamos el valor del resto como el siguiente símbolo a insertar en la cadena hexadecimal. El resultado es la cadena hexadecimal $53B8_{16}$.

0	←	5	←	83	←	1339	←	21432	Decimal
		↓		↓		↓		↓	
		5		3		B		8	Hexadecimal

Ejemplo C.5. Por último, vamos a convertir el número decimal 73234122 a base 256 (una dirección IPv4). Para ello, dividimos continuamente el valor decimal entre 256 (la base) hasta que el cociente se haga 0. En cada división tomamos el valor del resto como el siguiente símbolo a insertar en la dirección IPv4. Además, insertamos puntos como se requiere en la notación decimal con puntos. El resultado es la dirección IPv4 4.93.118.202.

0	←	4	←	1,117	←	286,070	←	73,234,122	Decimal
		↓		↓		↓		↓	
		4	•	93	•	118	•	202	IPv4 address

OTRAS CONVERSIONES.

Las conversiones de un sistema no decimal a otro sistema no decimal suelen ser muy fáciles. Por ejemplo, podemos convertir un número de binario a hexadecimal convirtiendo cada grupo de 4 bits en un dígito hexadecimal. Y a la inversa, también podemos convertir un dígito hexadecimal a un grupo de 4 bits. Los siguientes ejemplos muestran el proceso:

Ejemplo C.6. Vamos a convertir el número binario 1001111101_2 a su equivalente hexadecimal. Para ello, creamos grupos de 4 bits comenzando desde la derecha. A continuación reemplazamos cada grupo con su equivalente hexadecimal. Notar que hemos tenido que añadir dos 0s extra a la izquierda del último grupo. El resultado es $27D_{16}$.

0010	**0111**	**1101**	Binary
↓	↓	↓	
2	7	D	Hexadecimal

Ejemplo C.7. A continuación, vamos a convertir el número hexadecimal $3A2B_{16}$ a su equivalente en binario. Para lograrlo, cambiamos cada dígito hexadecimal a su equivalente binario de 4 bits. El resultado es $0011\ 1010\ 0010\ 1011_2$.

3	**A**	**2**	**B**	Hexadecimal
↓	↓	↓	↓	
0011	1010	0010	1011	Binary

Ejemplo C.8. Finalmente, vamos a convertir la dirección IPv4 112.23.78.201 a formato binario. Para ello, reemplazamos cada símbolo por su equivalente binario de 8 bits. El resultado es $01110000\ 00010111\ 01001110\ 11001001_2$.

112	•	**23**	•	**78**	•	**201**	IPv4 address
↓		↓		↓		↓	
01110000		00010111		01001110		11001001	Binary

REFERENCIAS Y RECURSOS.

LIBROS Y APUNTES.

Python.

- Automate The Boring Stuff with Python. (Al Sweigart). Editorial No Starch Press).
- Python Programming for the Absolute Beginner. (Michael Dawson). Editorial Cengage Learning.
- Python Crash Course. (Eric Matthes). Editorial No Starch Press.
- Invent your Own Computer Games with Python. (Al Sweigart). Editorial No Starch Press.

Redes.

- Data Communications and Networking 5th. Edition (Behrouz A. Forouzan). Editorial McGraw-Hill.
- TCP/IP Protocol Suite 4th. Edition (Behrouz A. Forouzan). Editorial McGraw-Hill.
- Foundations of Computer Science 4th. Edition (Behrouz A. Forouzan). Editorial Cengage Learning.
- Computer Networking. A top – down approach 6th. Edition (Kurose – Ross). Editorial Pearson.
- Microsoft Official Academic Course on Networking Fundamentals 2nd. Edition (Varios autores). Editorial Wiley.

ENLACES WEB.

Redes.

- Conversor ASCII binari a texto: https://www.binaryhexconverter.com/binary-to-ascii-text-converter

Gladiabots.

- Video-tutorial de Gladiabots: https://www.youtube.com/watch?v=Y3vI_f2JjKo&list=PLLEegc-maTgP29G9tVngpjnP4qmRWGdxY
- Wiki de Gladiabots: https://wiki.gladiabots.com/index.php?title=Main_Page
- Wiki de Gladiabots (Estrategias): https://wiki.gladiabots.com/index.php?title=Strategies https://wiki.gladiabots.com/index.php?title=Strategies_SimpleBots
- Foro de Gladiabots: https://forum.gladiabots.com/
- Consejos para los nuevos desarrolladores: https://learncodebygaming.com/blog/gladiabots-review-tips-for-new-programmers
- Modelos de los bots: https://sketchfab.com/gfx47/collections/gladiabots