GUÍA DE INICIACIÓN A APP INVENTOR

Tabla de contenido

Objetivo

A pesar de ser un recurso muy útil para iniciarse en el mundo de la programación, no parece haber demasiados tutoriales o guías de App inventor en castellano. Este documento pretende facilitar un poco las cosas a los hispanohablantes que quieren acercarse al apasionante mundo de la programación a través de la fantástica herramienta que es App Inventor.

El documento está organizado en sesiones de trabajo, sólo con el fin de estructurar mínimamente los proyectos y clarificar los objetivos. Las sesiones no tienen una duración determinada, y cada uno puede avanzar a su propio ritmo. Es muy conveniente que el estudiante se desvíe del guión principal cuando lo considere interesante, y desarrolle sus propias ideas, para volver de nuevo al guión al principio de la siguiente sesión. La programación es una artesanía, y la creatividad, la libertad, y la imaginación son fundamentales.

Este documento fue escrito inicialmente para utilizarlo como guía durante las sesiones de iniciación a App Inventor que se desarrollan cada tarde del sábado en el Coder Dojo del centro MediaLab Prado de Madrid, en España. Las posibilidades de App Inventor son extensísimas, y la intención es simplemente que sirva de ayuda en los primeros pasos, para que el estudiante pueda después continuar por sí mismo, buscando más recursos de aprendizaje en la web, y llevado a cabo sus propias ideas.

Muy gustosamente atenderé las preguntas, dudas, comentarios o ideas a través del correo electrónico raulconm@gmail.com, o la cuenta de Twitter @raulconm

¡Feliz programación!

Raúl C.

¿Qué es AppInventor?

App Inventor parte de una idea conjunta del Instituto Tecnológico de Massachusetts y de un equipo de Google Education. Se trata de una herramienta web de desarrollo para iniciarse en el mundo de la programación. Con él pueden hacerse aplicaciones muy simples, y también muy elaboradas, que se ejecutarán en los dispositivos móviles con sistema operativo Android.
App Inventor es un lenguaje de programación basado en bloques (como piezas de un juego de construcción), y orientado a eventos. Sirve para indicarle al "cerebro" del dispositivo móvil qué queremos que haga, y cómo.
Es por supuesto muy conveniente disponer de un dispositivo Android donde probar los programas según los vamos escribiendo.

Sesión #1

Objetivos

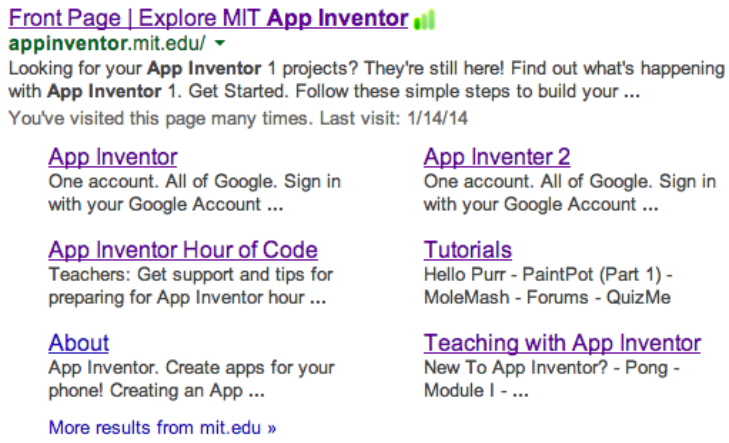
1. Crear una cuenta Google
2. Instalar App Inventor 2
3. Instalar MIT AI2 Companion en el dispositivo Android, o conocer el emulador
4. Crear una aplicación e instalarla en el móvil
5. Conocer la página web de App Inventor (tutoriales, ejemplos, guías etc.)

Crear una cuenta Google

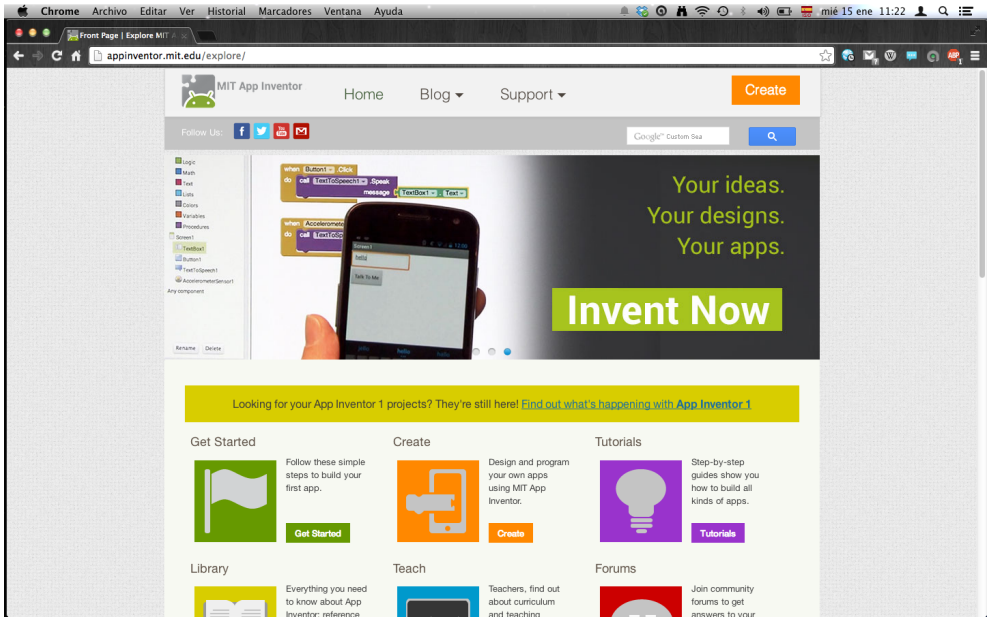
Es necesario crear una cuenta Google porque App Inventor es un trabajo conjunto entre Google y el MIT (http://web.mit.edu/). Recuérdese que Android es de Google.
Abrir el navegador. OJO, tiene que ser Google Chrome, Safari o Firefox. Internet Explorer aún no es compatible con App Inventor.
Ir a la página https://accounts.google.com/
Utilizar una cuenta ya existente o crear una nueva. Nos hará falta para usar App Inventor.
Seguir las instrucciones de la página de Google para crear una nueva cuenta.

Instalar App Inventor 2

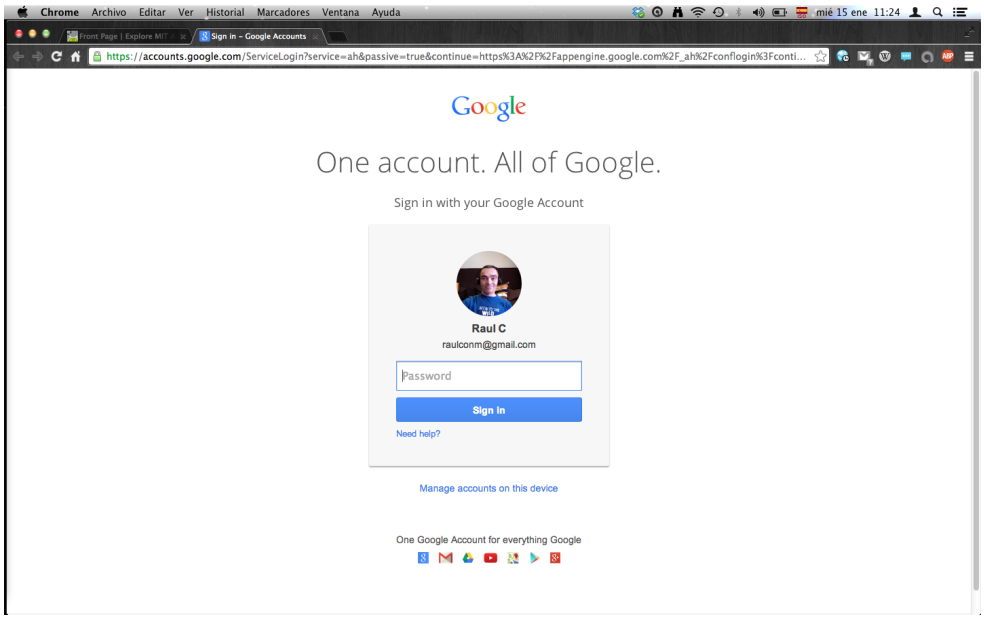
Buscamos "App Inventor" en el buscador de Google.
Hacer clic sobre **Front Page | Explore MIT App Inventor**.



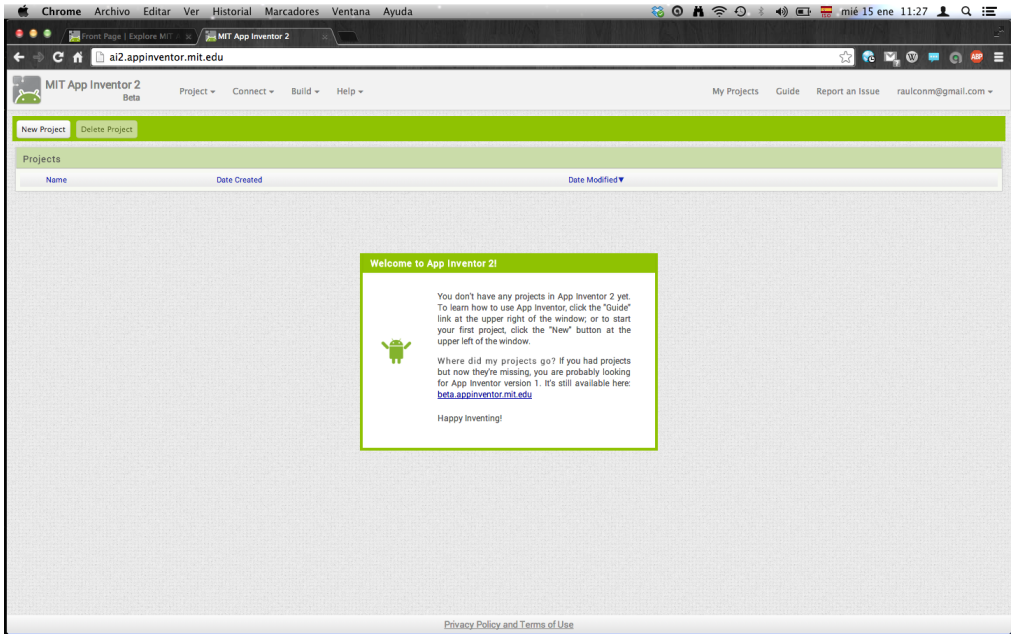
Se abrirá la siguiente página:



Hacemos clic sobre el botón **Create**. Si no hemos abierto sesión en Google, el navegador nos pedirá que lo hagamos ahora.

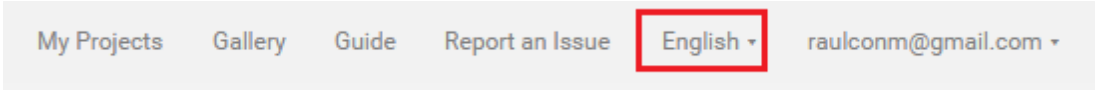


Al acceder a la cuenta de Google veremos nuestra página en App Inventor.



Configurar el idioma

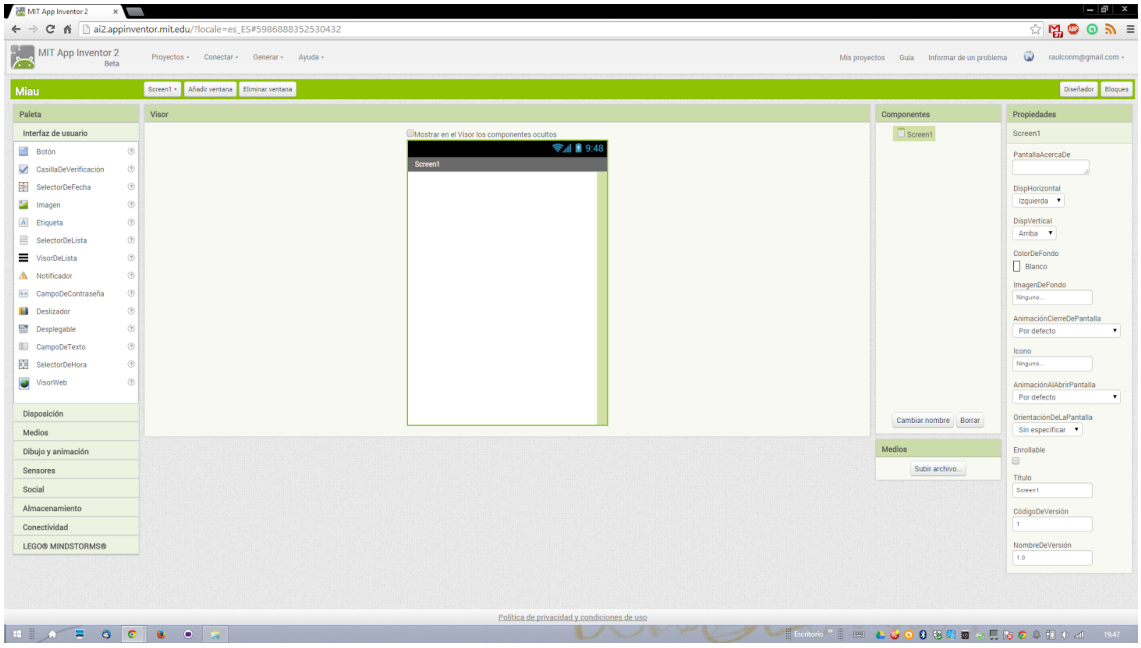
App Inventor nos muestra inicialmente el interfaz en inglés, sin embargo, podemos utilizarlo también en castellano. La elección del idioma se hace pulsando sobre el icono de la bola del mundo situado en la esquina superior derecha de la página de App Inventor.



El interfaz está traducido al castellano, pero podemos aún encontrar algunos textos genéricos de ayuda en inglés. En todo caso, serán muy pocos, y no deberíamos encontrar problemas para trabajar con la herramienta sin saber inglés.

¡Empezamos!

Una vez configurado en castellano pulsamos el botón **Comenzar un proyecto nuevo...** y le damos a nuestro proyecto el nombre "Miau"
Esto nos lleva a la ventana principal de App Inventor.



La pantalla que estamos viendo se divide en partes:
- A la izquierda están los objetos que vamos a usar para diseñar la pantalla de nuestra aplicación. Botones, imágenes, dibujos, etc. Es como la paleta de un pintor
- Después explicaremos la parte derecha
- ¿Qué es la pantalla del centro? Representa la pantalla del móvil, y sirve para DISEÑAR el aspecto de la aplicación. La llamaremos Visor

Arriba a la izquierda hay un botón importante: **Conectar.** Para poder probar cómo funciona lo que vamos haciendo necesitamos transferirlo a un móvil, o usar el emulador incluido en App Inventor (esta opción es menos recomendable).

Para conectar App Inventor con el móvil hacer clic en **Conectar**, y elegir la opción AI COMPANION. Se abrirá una pantalla como esta:

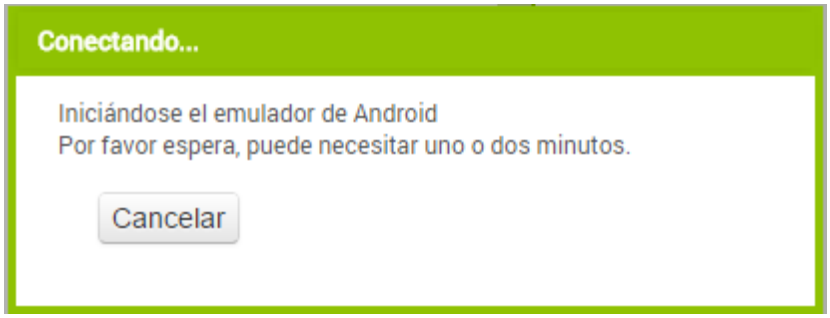
Instalar MIT AI2 Companion en el dispositivo Android, o conocer el emulador

Para poder utilizar nuestro dispositivo como banco de pruebas tenemos que descargarnos de Google Play Store una aplicación que se llama "AI2 Companion". Buscarla en Google Play Store con ese nombre y descargarla ahora. Ocupa poco, y sólo hará falta descargarla una vez.

Abriremos ahora en el móvil la aplicación que hemos descargado, haciendo clic sobre el icono **MIT AI2 Companion**. Puede estar en la página principal o dentro del grupo de Aplicaciones.

Cuando se abra, elegiremos **Connect with code** (color naranja), y escribiremos el código de letras y números (alfanumérico) que aparece en la pantalla del ordenador. Quien pueda leer códigos QR puede hacerlo desde la pantalla ahora pulsando en el botón azul **scan QR code**. Para que esto sea posible el ordenador y el dispositivo deben estar en la misma red, es decir, que deben tomar la IP del mismo rango, típicamente del mismo enrutador (router). Si no disponemos de WIFI podremos utilizar el emulador, o una conexión USB (ver detalles para USB en la web de App Inventor).

Para abrir el emulador, haremos clic sobre **Conectar** y elegiremos la opción **Emulador**. No hay que hacer nada más, tarda un poco, pero una vez que cargue se verá la pantalla en blanco de nuestra aplicación.



Una vez establecida la conexión entre App Inventor y el móvil veremos una pantalla en blanco con el título **Screen1**.

¡Empezamos a programar!

Antes de nada, para este ejercicio necesitamos dos recursos que tenemos que descargar en nuestro ordenador:

- **Kitty.png**
- **Miau.mp3**

Podemos descargarlos del siguiente contenedor:

http://coderdojo-medialabprado.4shared.com

Haremos clic con el botón izquierdo sobre el archivo que queremos descargar. Se abrirá una pantalla mostrando el archivo.

Pulsaremos entonces sobre el botón **Descargar**.

Saldrá una pantalla mostrándonos el tiempo que tendremos que esperar para que comience la descarga gratuita.

Al pulsar **DESCARGA GRATIS** el servidor nos pedirá que le autoricemos a usar una cuenta. Le autorizaremos a usar la de Google+, ya que la hemos creado antes.

Comenzará entonces la descarga del archivo. En algunos casos puede que el navegador abra directamente el archivo de imagen que hemos descargado. En ese caso bastará que hagamos clic con el botón derecho y seleccionemos la opción **Guardar imagen como...** para que quede almacenado en nuestro disco. Repetiremos este proceso para los dos archivos mencionados.

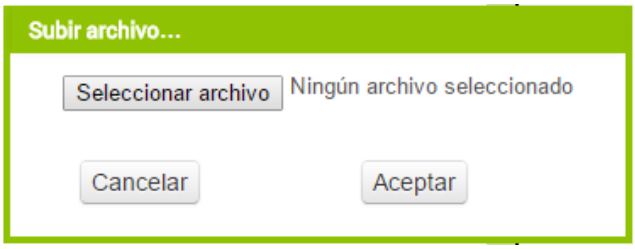
Incluir un botón en la pantalla

En la parte izquierda hacemos clic en el tipo de objeto **Botón**, y sin soltar arrastramos hasta el visor. Si todo funciona bien se verá en el visor, y también en la pantalla del móvil, o del emulador.

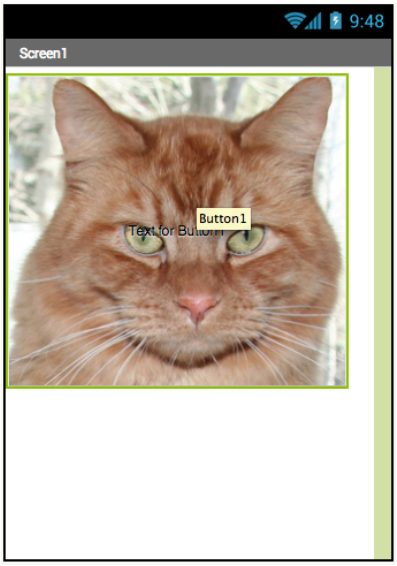
Un botón es un objeto sobre el que podemos hacer clic, y puede tener diferentes aspectos.

Para que el botón tenga la imagen del gato hacemos clic en el botón, y en la parte derecha de App Inventor, en Propiedades, y bajo la propiedad **Imagen**, hacemos clic en **Ninguno**...

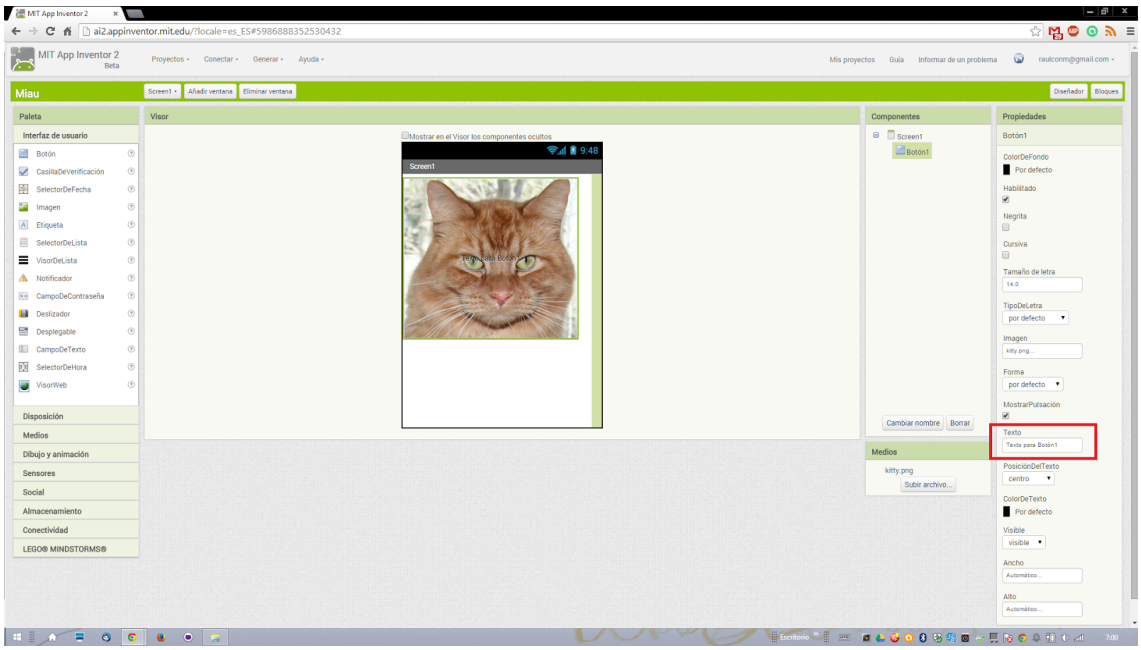
Elegimos la opción **Subir archivo...**, y después **Seleccionar archivo**



Elegimos el archivo del gato en nuestro disco duro y pulsamos **Aceptar** para subirlo a la página de nuestro proyecto en App Inventor. Se verá el gato como imagen del botón, que ahora será más grande.



Para quitar el texto "Texto para el Botón1" que aparece por debajo del gato hay que borrar el valor de una propiedad Texto del botón, en la parte derecha de la ventana.



Si no vemos la cara del gato entera en la pantalla del dispositivo deberemos cambiar los valores de las propiedades **Ancho** y **Alto** del botón por “Ajustar al contenedor”, para que se ajusten al tamaño máximo disponible en la pantalla del dispositivo.

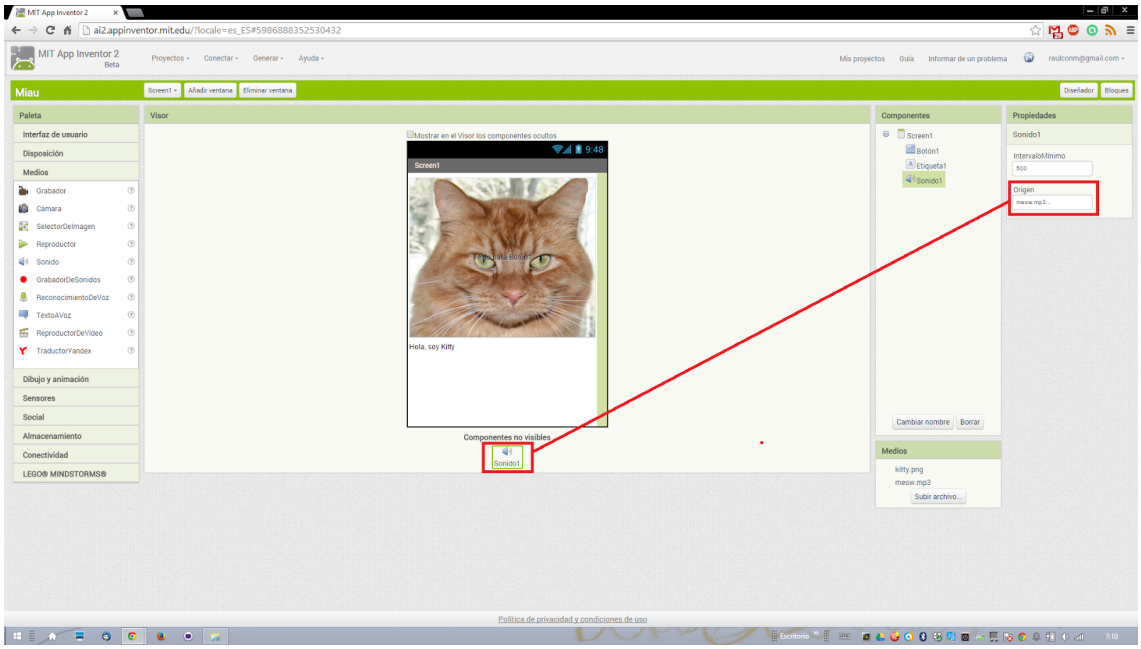
Para incluir una etiqueta debajo del gato que ponga “Hola, soy Kitty” arrastramos un componente **Etiqueta** hasta el visor, y la soltamos debajo del gato.

Investiguemos ahora para descubrir cómo cambiar el texto “Texto para Etiqueta1” por “Hola, soy Kitty”. Una pista: hay que seleccionar la etiqueta en el visor, y luego cambiar sus propiedades en la parte derecha de la ventana de trabajo de App Inventor.

Añadir un sonido

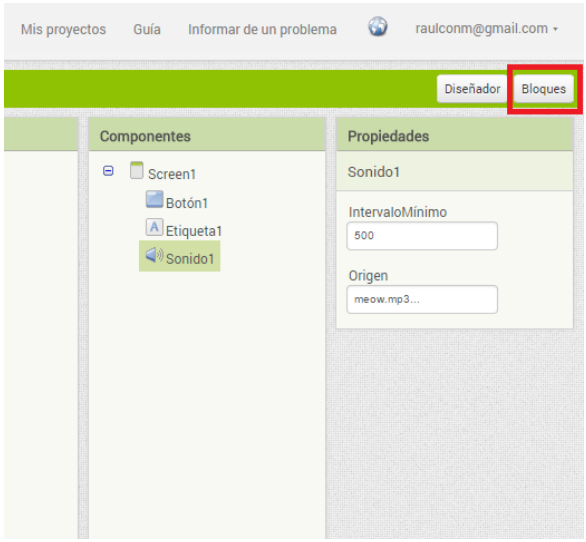
Ahora añadiremos un sonido a nuestra aplicación, arrastrando hasta el visor el icono **Sonido**, que está dentro del grupo **Medios**, en la Paleta. Ojo, este objeto no se verá en el móvil o en el emulador, porque no es una imagen, ni un botón, ni una etiqueta. Por eso aparece debajo del visor, en el apartado **Componentes no visibles**.

Investiguemos ahora de nuevo para saber cómo asociar a este objeto que hemos creado el sonido “Miau.pm3” que hemos descargado. De nuevo hay que usar el panel de propiedades para este componente. No es difícil, haremos clic sobre el valor de la propiedad **Origen** del componente **Sonido1** y subiremos el archivo descargado.

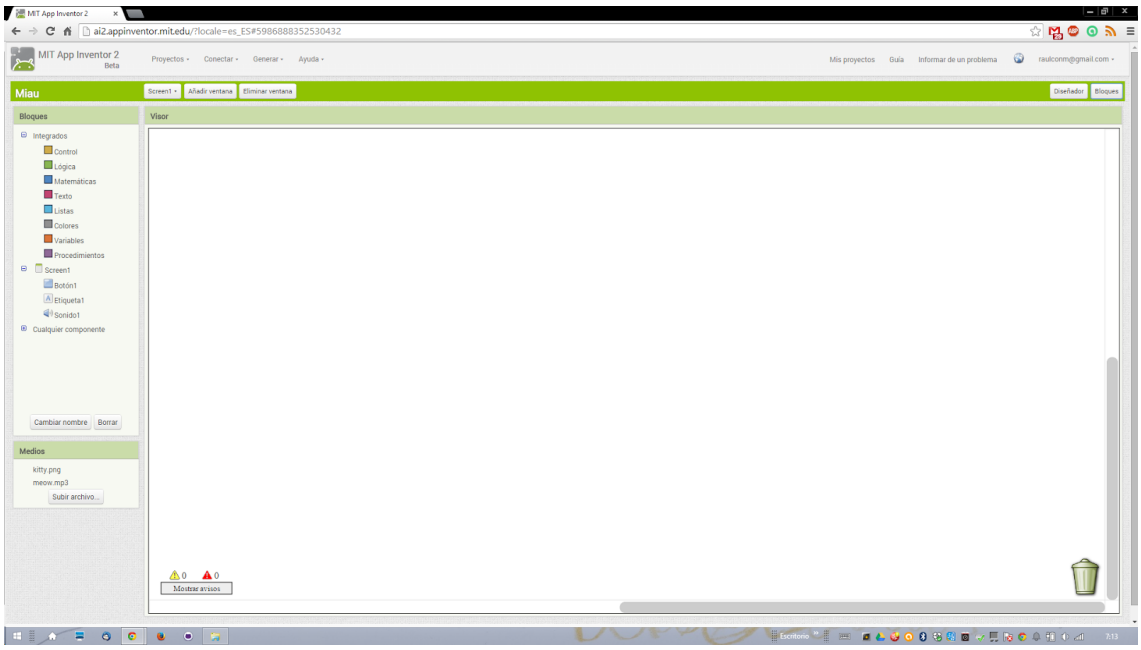


Con esto hemos terminado de diseñar el aspecto de nuestra aplicación Ahora viene la magia, tenemos que programar cómo se comportará la aplicación. ¡Eso es programar!

Hacemos clic en el botón **Bloques** situado en la esquina superior derecha.

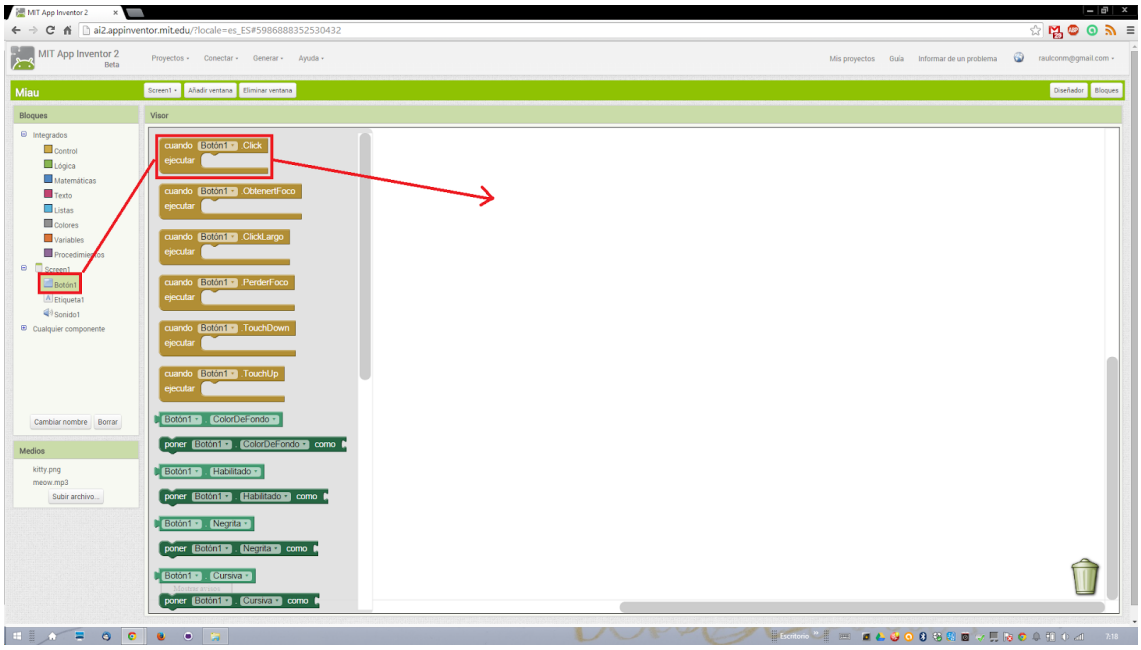


Esto abre la ventana de programación con bloques. La parte más amplia, ahora en blanco, es el Editor, donde colocaremos los bloques de nuestros programas.



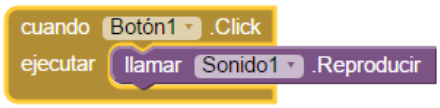
Vamos a hacer que suene el sonido del gato cada vez que hagamos clic sobre la imagen del gato (botón).

Hacemos clic en **Botón** para que se muestren los bloques de colores disponibles para escribir nuestro código, el programa. Se abre un "cajón de herramientas" con todos los bloques que podemos utilizar. Arrastramos hasta el editor el que dice **Botón1.Clic**.



Los bloques color mostaza son los **manejadores o gestores de sucesos**. Indican qué hay que hacer cuando sucede algo en la aplicación. En este caso, el manejador nos permitirá decirle al ordenador qué debe hacer cuando hagamos clic sobre el gato.

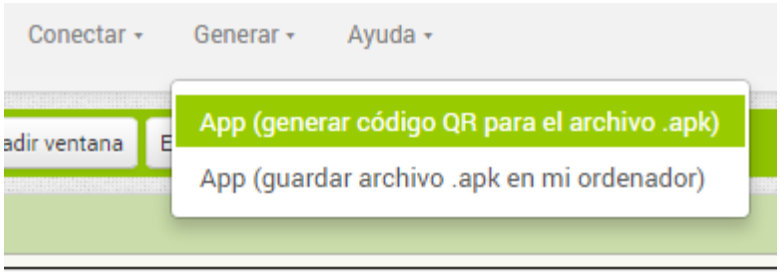
Ahora hacemos clic sobre nuestro componente **Sonido1** para abrir su cajón. Entonces arrastramos la instrucción **Llamar.Sonido1.Reproducir** hasta "encajarla" dentro del manejador que hemos creado para el botón.



¡Enhorabuena por la primera aplicación!

Ya podemos probarla en el móvil, pero la perderemos si cerramos la aplicación AI2 que nos conecta con el ordenador. Para instalarla en el móvil permanentemente, como cualquier otra aplicación, podemos generar un código QR.

Para ello hacemos clic en **Generar** y elegimos la opción **App (generar código QR para el archivo .apk).**



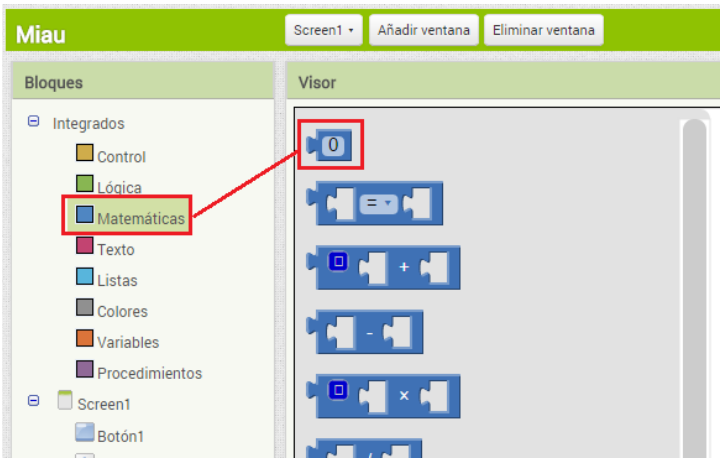
Esto tomará un poco de tiempo, después del cuál aparecerá un código QR que podremos capturar en nuestro móvil. La aplicación quedará descargada entonces en nuestro teléfono/Tablet, para ejecutarla siempre que queramos.

Ahora a vibrar

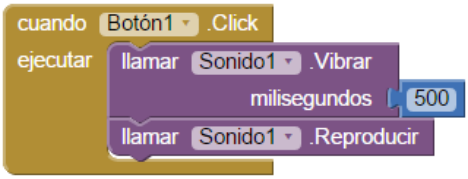
Podemos mejorar la aplicación, haciendo que el teléfono vibre a la vez que el gato maúlla. Una pista: hay que buscar dentro del cajón de bloques del objeto **Sonido1**.

El bloque que hace que el móvil vibre es **Llamar.Sonido1.Vibrar**. Este bloque, a diferencia del anterior, tiene un "encajador" por el lado derecho. Sirve para indicar cuánto tiempo tiene que vibrar el dispositivo, en milisegundos.

Para poner aquí un valor de tiempo hay que abrir el cajón **Matemáticas** y arrastrar el bloque de arriba hasta el encajador libre a la derecha del bloque **Llamar.Sonido1.Vibrar,** donde indica **milisegundos**.



Ahora cambiamos el valor 0 por el valor 500, para que vibre durante medio segundo. Los bloques quedarán así

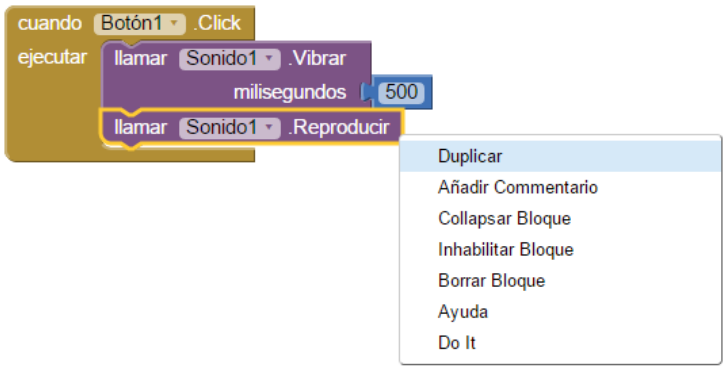


¿Qué pasa si ponemos 2500 en el bloque azul? Vibrará durante 2,5 segundos. Para cambiar el valor hay que hacer clic sobre el número, y escribir el nuevo valor.

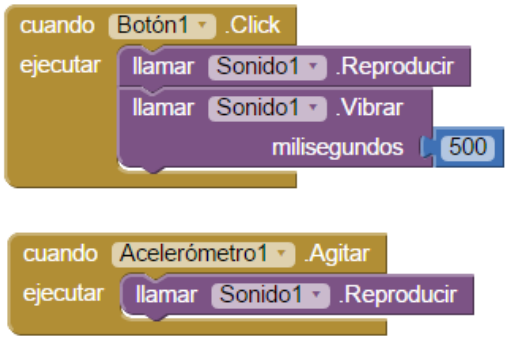
Ahora volveremos al Diseñador y añadiremos un objeto **Acelerómetro**, que se encuentra en la Paleta, dentro del cajón **Sensores**. Se quedará bajo la ventana del visor, porque no se refiere a un objeto visible en nuestra interfaz de usuario.

Ahora, en el editor de bloques, elegiremos en la ventana de bloques el objeto **Acelerómetro1** que hemos creado. De su cajón elegiremos el bloque mostaza **cuando.Acelerómetro1.Agitar**

Ahora copiamos el bloque **Llamar.Sonido1.Reproducir** de arriba, haciendo clic con el botón derecho sobre él y seleccionando **Duplicar**.



Una vez duplicado lo encajaremos con el bloque mostaza que hemos creado. El editor de código aparecerá como en la siguiente figura.



¿Qué va a ocurrir? El gato también maullará cada vez que agitemos el móvil. El acelerómetro es el sistema que detecta que el móvil se mueve, o cambia de orientación vertical a horizontal. Es muy útil para muchas aplicaciones.

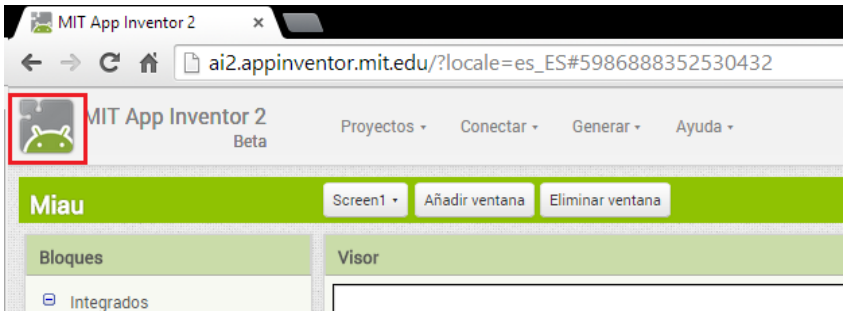
Aquí hay dos EVENTOS distintos, y le estamos indicando al móvil, a través de este programa, qué debe hacer cuando suceda cada uno de estos eventos.

POR ESO SE DICE QUE APP INVENTOR ES PROGRAMACIÓN ORIENTADA A EVENTOS (EVENT-DRIVEN PROGRAMMING).

Intentemos ahora generar nosotros mismos el código QR para esta nueva aplicación.

Conocer la página web de App Inventor (tutoriales, ejemplos, guías etc.)

Podemos entrar en la web de App Inventor haciendo clic en el icono del androide de la esquina superior izquierda, y dar “un paseo” por los recursos que hay en la página de App Inventor.



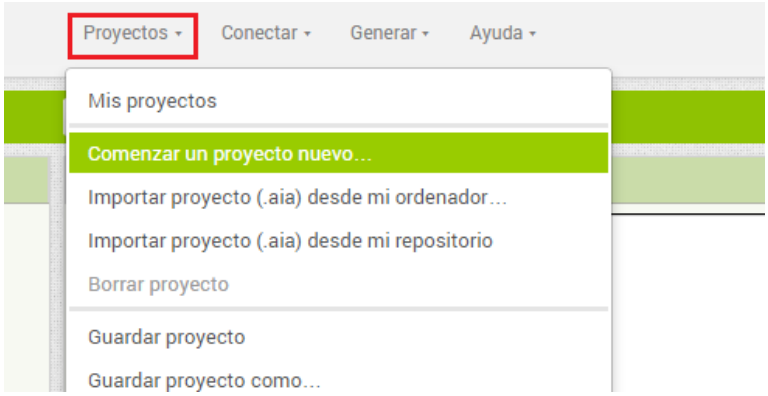
Ideas para profundizar nosotros mismos

Conocer otros componentes de la pantalla
Investigar cómo renombrar los objetos que hemos creado.
Descubrir dónde están los recursos “Medios” que vamos agregando.
Comprobar cómo se eliminan objetos del editor de bloques, arrastrándolos a la papelera.

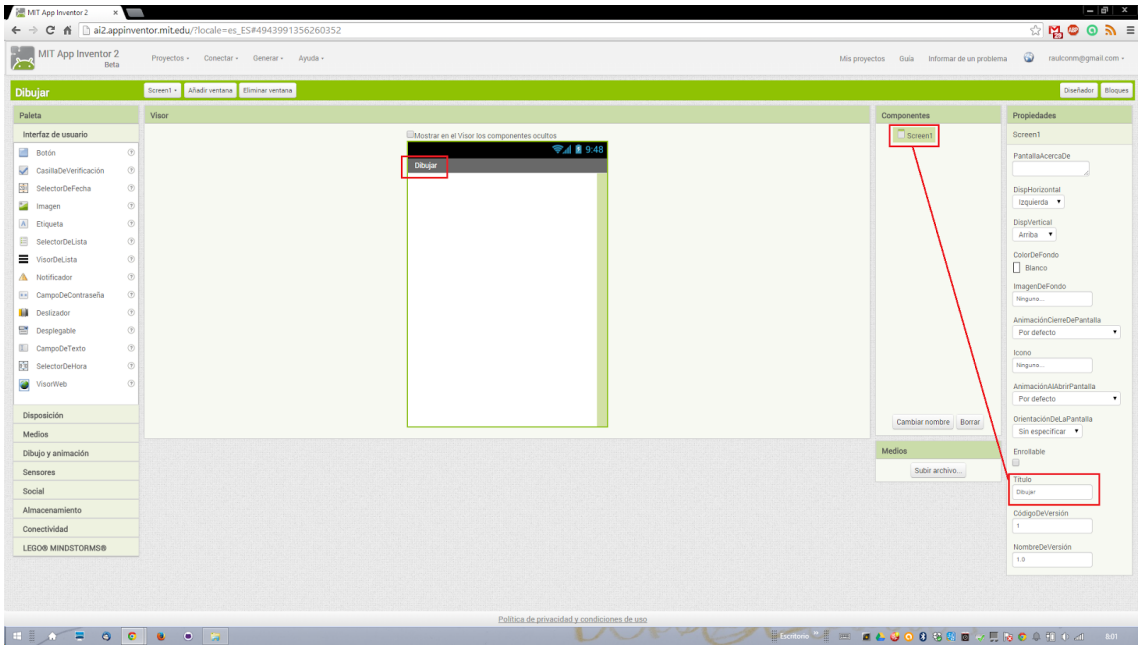
Sesión #2

Crear una aplicación para dibujar

Una vez que hemos visto el manejo más básico de las pantallas, el Diseñador, y el Editor de Bloques de App Inventor, ya estamos listos para hacer algo más avanzado. Comenzaremos haciendo clic en el desplegable **Proyectos**, y eligiendo la opción **Comenzar un proyecto nuevo...**



Una vez dentro, cambiaremos el nombre de la pantalla **Screen1** por **Dibujar** (en la propiedad **Título** del componente **Screen1**).

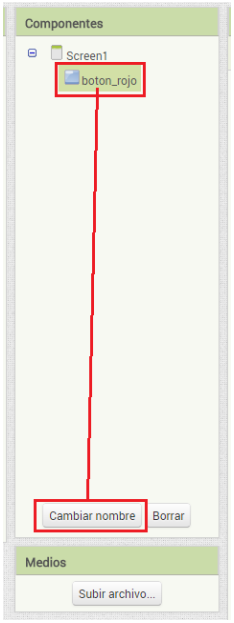


ATENCIÓN

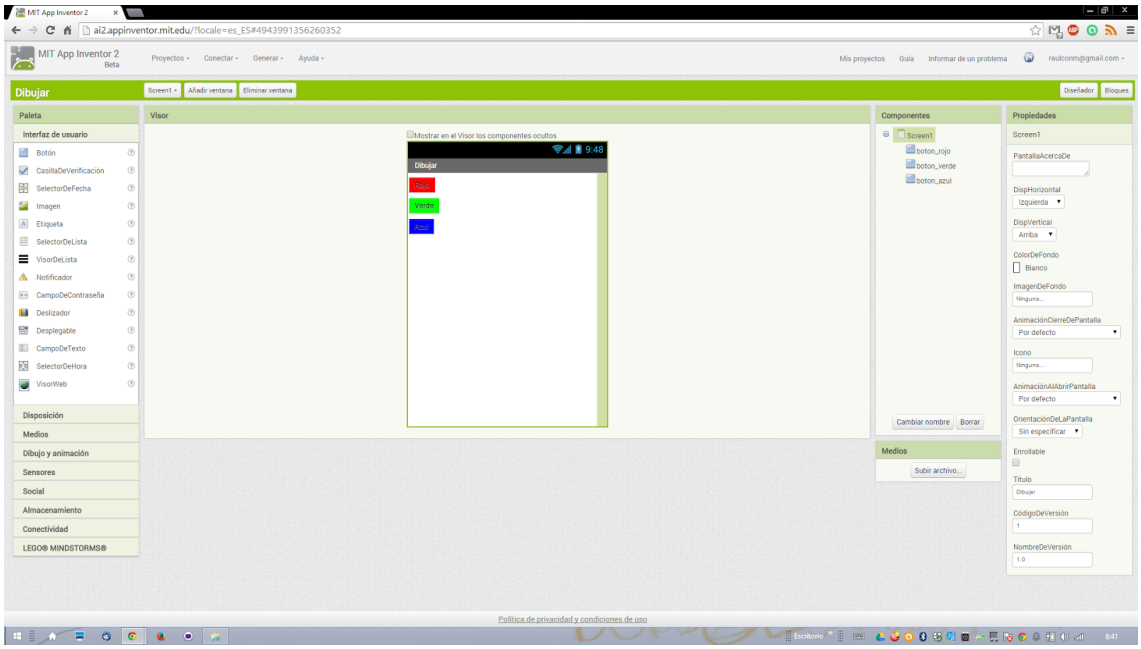
Es importante siempre utilizar nombres descriptivos para los objetos que vamos creando. Al principio, cuando nuestras aplicaciones son pequeñas, es fácil recordar cada objeto, pero a medida que los programas van haciéndose más complejos es fundamental saber para qué sirve cada objeto, cada variable, y sólo podremos hacerlo si le hemos dado un nombre que describe qué es o para qué sirve.

Ahora crearemos por nuestra cuenta un botón, y cambiaremos su propiedad **Texto** por "Rojo", y finalmente pondremos en rojo su propiedad **ColorDeFondo**. Adelante, solamente hay que jugar con las propiedades del botón.

Cambiaremos el nombre del botón **Botón1** por **boton_rojo** en el cuadro de componentes del Diseñador. En lugar de usar un espacio utilizaremos un signo de subrayado, y no emplearemos tildes, porque el sistema entiende que es un carácter especial no válido (esto aún está en inglés ☺).



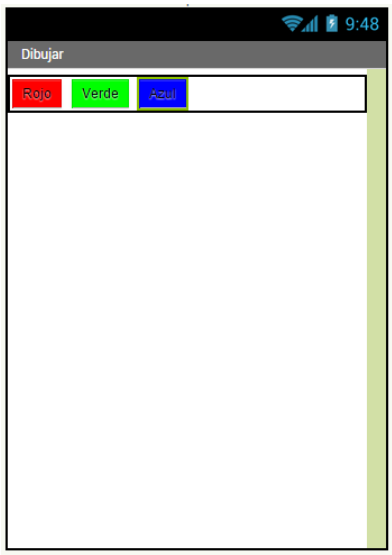
Crearemos después otro botón verde, y finalmente otro azul, y haremos lo mismo que hemos hecho con el rojo, cambiaremos su nombre, el color de fondo, y el texto que aparece en el botón.



Como queremos que los botones queden en una misma línea horizontal, vamos a recolocarlos en la pantalla añadiendo un componente **DisposiciónHorizontal**. Lo arrastramos desde el cajón **Disposición** hasta el Visor. Este objeto aparecerá en el Visor como un cuadro vacío.

Para que se ajuste al ancho de la pantalla del visor ¿qué haremos? En su propiedad **Ancho** elegiremos la opción **Ajustar al contenedor...**

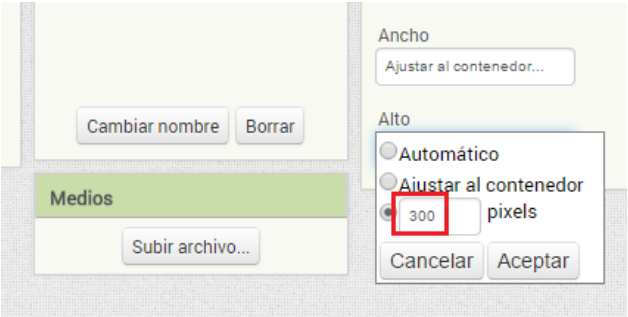
Ahora, en el Visor, arrastraremos los tres botones dentro de **DisposiciónHorizontal1**. Si no nos gusta este nombre podemos cambiarlo por **MarcoHorizontal**, por ejemplo. Para que la altura del marco se ajuste al tamaño de los botones podemos decir que su **Altura** sea automática. Así siempre se adaptará al tamaño del componente más alto de todos los que contiene.



Ahora añadimos el lienzo, arrastrando al Visor el componente **Lienzo**, que se encuentra dentro del cajón **Dibujo y animación** de la Paleta. Lo colocamos justo debajo, fuera del **MarcoHorizontal**. Lo más cómodo es definir que su anchura (propiedad **Ancho**) sea automática, para que se extienda hasta los bordes izquierdo y derecho del Visor. En cuanto a su altura, mejor experimentar con diferentes tamaños hasta que ocupe el espacio que queremos.

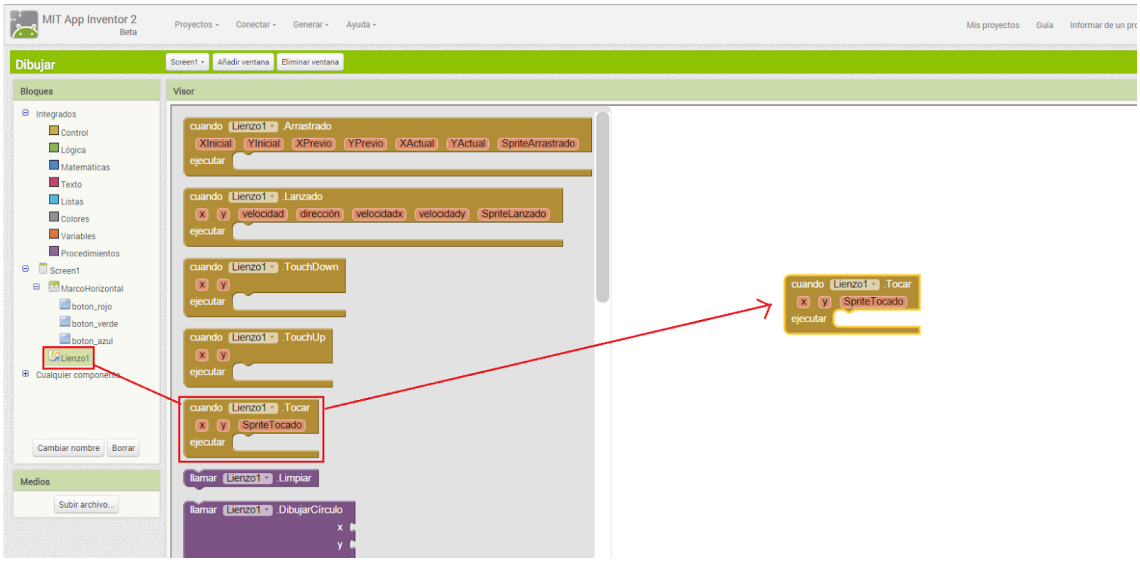
El tamaño se especifica en *pixels*, es decir, en puntos de la pantalla. Cada línea de la pantalla tiene un número de pixels. El número de ellos que haya, en líneas y columnas,

define la resolución de la pantalla. Podemos probar con 300 pixels, y modificarlo más tarde si vemos que no es el mejor tamaño.



Ahora vamos a definir el comportamiento del programa, a decir cómo tiene que funcionar. Vamos entonces al editor de bloques.

Arrastraremos el bloque **cuando.Lienzo1.Tocar** desde el cajón del objeto **Lienzo1** hasta el editor. Esto indica que cada vez que toquemos el lienzo con el dedo tendrá que ocurrir lo que digamos dentro de este bloque mostaza.



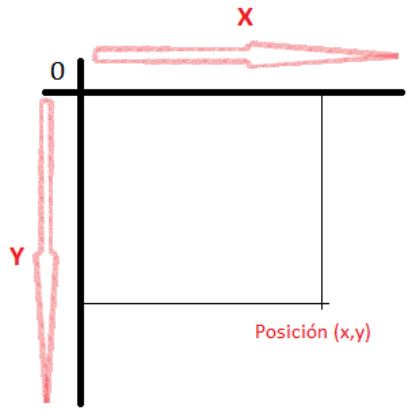
Ahora arrastraremos el bloque morado (de acción) **llamar.Lienzo1.DibujarCírculo** hasta encajarlo dentro del bloque mostaza. Veremos que para funcionar necesita que le asociemos a la derecha tres bloques de información adicional (x, y, r).

Coordenadas x e y

El pensador René Descartes inventó hace tiempo un sistema para definir la posición de un objeto sobre un plano. En su honor, este sistema se conoce como *Coordenadas Cartesianas*. Para conocer más sobre este tema visitemos la página http://es.wikipedia.org/wiki/Coordenadas_cartesianas

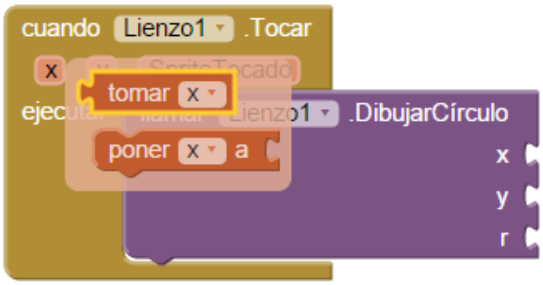
Nosotros vamos a utilizar el sistema de Descartes para referirnos a un punto cualquiera dentro de nuestro **Lienzo1**. Siempre que toquemos con el dedo dentro del lienzo, el

bloque mostaza guardará cuál es el valor de la coordenada x y la coordenada y del punto que estamos tocando, porque seguro que lo necesitaremos. App Inventor utiliza el sistema cartesiano para determinar la posición de un punto determinado dentro de un lienzo. La x aumenta desde la izquierda a la derecha, y la y lo hace desde arriba hacia abajo.



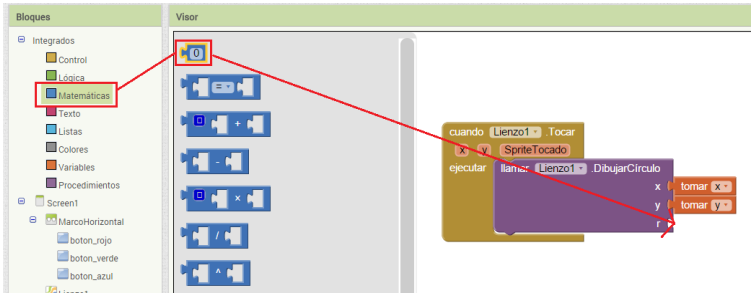
Precisamente para saber dónde dibujar el círculo, el bloque morado deberá saber que saber qué **x** e **y** tiene el punto de la pantalla que estamos tocando. Lo tomaremos del bloque mostaza.

Al dejar quieto el puntero del ratón sobre el icono de la **x** del bloque mostaza nos saldrá una ventanita en la que aparecerá un pequeño bloque **tomarx** de color naranja. Tenemos que arrastrarlo hasta encajarlo con el hueco superior del bloque morado **llamar.Lienzo1.DibujarCírculo**.



Haremos lo mismo con la **y**. Esto hará que el círculo se dibuje sobre la x y la y que estamos tocando, y no en cualquier otro punto de la pantalla.

Nos falta definir la **r**. ¿Qué es? Indica el tamaño del círculo que vamos a dibujar, su radio (de ahí la r). Para definir el tamaño, el radio, abrir el cajón **Matemáticas** de la zona de bloques y arrastrar el bloque azul que indicar el valor 0 (cero) hasta el hueco **r** de nuestro bloque morado.



Queremos que los círculos se vean bien, así que le asignaremos a **r** el valor 10. El radio del círculo será 10.

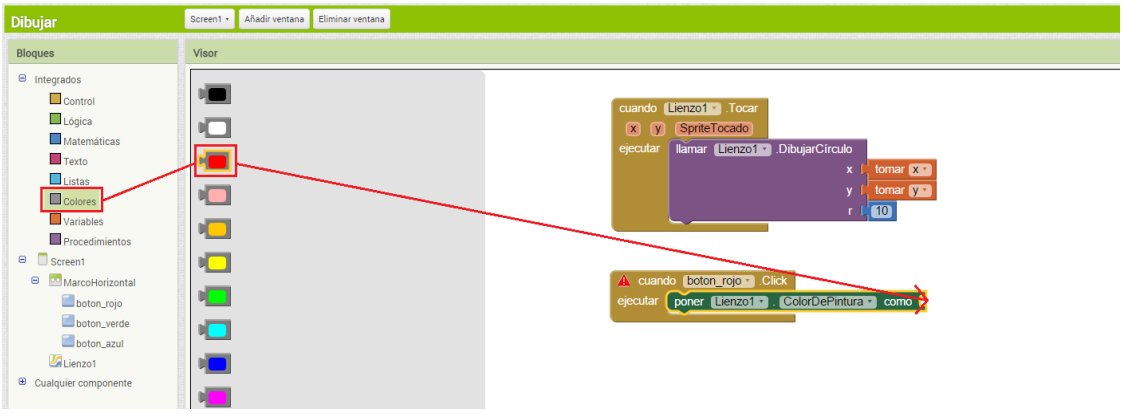
Ahora podemos tocar dentro del lienzo, en la pantalla del móvil o el emulador, para ver cómo se dibujan pequeños círculos. Pero para que los círculos sean rojos, verdes o azules en lugar de negros, hay que decirle al móvil que lo haga creando nuevos bloques. Una pista: hay que utilizar un bloque mostaza del objeto **botón_rojo**. Ya hemos usado este tipo de bloque mostaza en el programa del gato.

Utilizaremos el bloque **cuando.boton_rojo.Clic**. Lo arrastramos hasta el editor de bloques.

¿Qué tiene que ocurrir cuando hagamos clic sobre el botón rojo? Tiene que cambiarse a rojo el color del pincel que usamos en el lienzo. ¿Dónde tendremos que definir eso? Una pista: el color del pincel es una característica del **Lienzo1**, así que tendremos que buscar la manera de hacerlo usando algún bloque del cajón del objeto **Lienzo**. Otra pista: es un bloque de color verde oscuro.

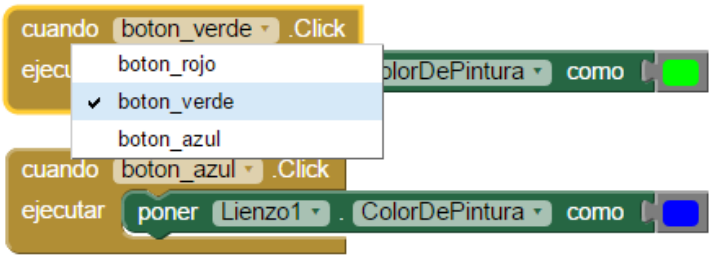
Es el bloque **poner.Lienzo1.ColorDePintura.como.** Lo arrastraremos hasta encajarlo dentro del bloque mostaza **cuando.boton_rojo.Clic**.

Falta indicar que queremos que sea el color rojo. Para ello escogeremos el pequeño bloque que identifica a este color de entre los colores que hay en el cajón **Colores** de la Paleta de App Inventor.



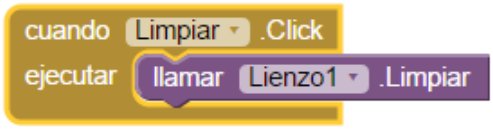
Haremos lo mismo con los botones para color verde y color azul. Lo mejor es hacerlo duplicando el bloque mostaza que hemos creado para el color rojo. Una vez hecho, en el nuevo bloque mostaza, podemos desplegar la lista del nombre de los botones y

elegir el del color verde. Entonces sólo tendremos que cambiar el bloque rojo por el verde, y listo. Lo mismo con el botón para el azul.



Cuando probemos el resultado se nos "ensuciará" el lienzo enseguida, así que hay que poner un botón para dejar el lienzo en blanco de nuevo. ¿Cómo se hará? Varias pistas:

- Crear **DisposiciónHorizontal1** debajo del lienzo
- Meter dentro un nuevo botón y llamarlo **Limpiar**
- Cambiar el texto del botón por "Limpiar"
- Poner los bloques para que al hacer clic sobre **Limpiar** se limpie el lienzo

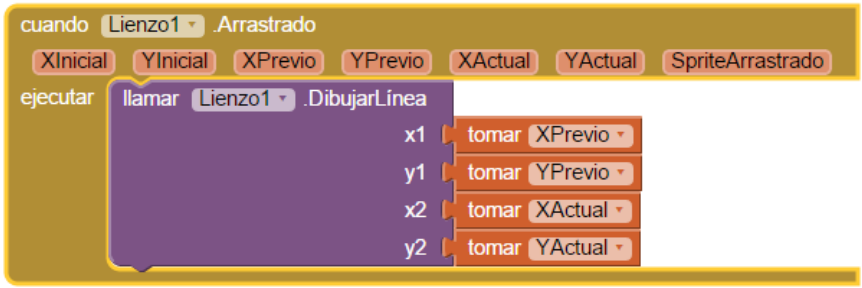


En realidad sería más interesante si en lugar de hacer circulitos, pudieran dibujarse líneas al arrastrar el dedo por la pantalla. ¿Qué bloque mostaza del objeto **Lienzo** habría que usar para que se dibujara según fuéramos arrastrando el dedo por el lienzo? Una pista: deberemos utilizar un bloque naranja del componente **Lienzo1**.

ATENCIÓN
En programación no hay una única manera de hacer bien las cosas, es decir, podemos conseguir el mismo resultado utilizando bloques diferentes. El objetivo crear el programa de la forma más simple y más eficiente, para que nuestro código sea más "elegante".

Podemos usar estos dos tipos de bloques, por ejemplo, para dibujar un círculo por cada punto donde vayamos arrastrando el dedo. **XActual** e **YActual** se refieren precisamente al punto representado por las coordenadas x e y actuales. Veremos como funciona esto al arrastrar el dedo despacio por el lienzo. Si deslizamos el dedo demasiado rápido se verán los puntos separados.

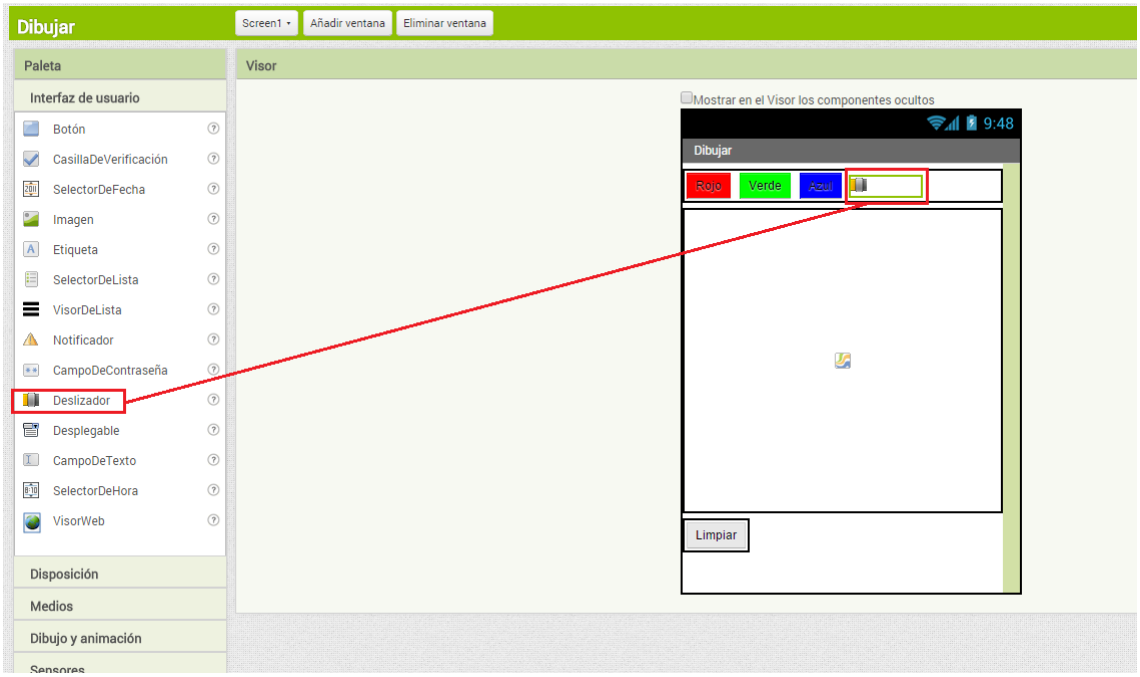
Más apropiado puede ser entonces utilizar este otro bloque morado, que se basa en dibujar líneas, y no círculos. Para dibujar una línea hace falta especificar el punto inicial de la línea y el punto final. Por ello para usar este bloque hay que indicarle dos coordenadas x y dos y. El primer (x1, y1) define el punto inicial de la línea, y el segundo par (x2, y2) define el punto final de la misma. Al hacer clic sobre el lienzo, sin levantar el dedo, el programa guardará en su memoria el valor de la x e y del punto que hemos tocado (determinado por XPrevio e YPrevio). Según vayamos deslizando el dedo el programa irá detectando el cambio de posición y dibujará una línea desde ese punto guardado hasta el punto que estamos tocando actualmente (determinado por XActual e YActual). Esto se repetirá todo el tiempo, muy rápidamente, mientras sigamos arrastrando el dedo por la pantalla. Se dibujarán por lo tanto muchas líneas muy cortas, una detrás de la otra, dando la sensación de ser una única línea continua.



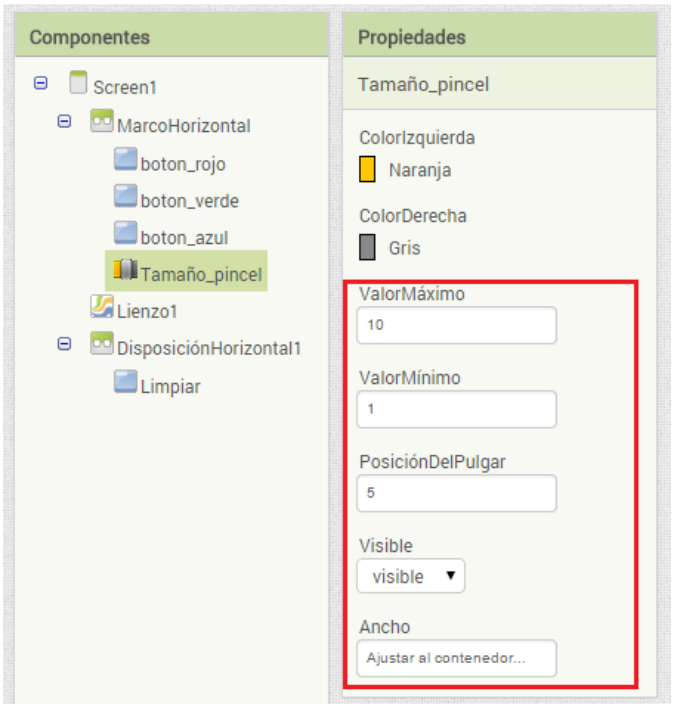
Como hemos decidido utilizar el bloque **cuando.Lienzo1.Arrastrado** podemos eliminar el bloque naranja **cuando.Lienzo1.Tocar** que había utilizado anteriormente, y que dibujaba círculos, a no ser que queramos mantener ambos botones, par que nuestra aplicación haga dos cosas distintas dependiendo de la acción que tomemos: si sólo tocamos el lienzo se dibujará un círculo, y si tocamos el lienzo y deslizamos el dedo se dibujará una línea.

Cambiar el tamaño del pincel con un Deslizador

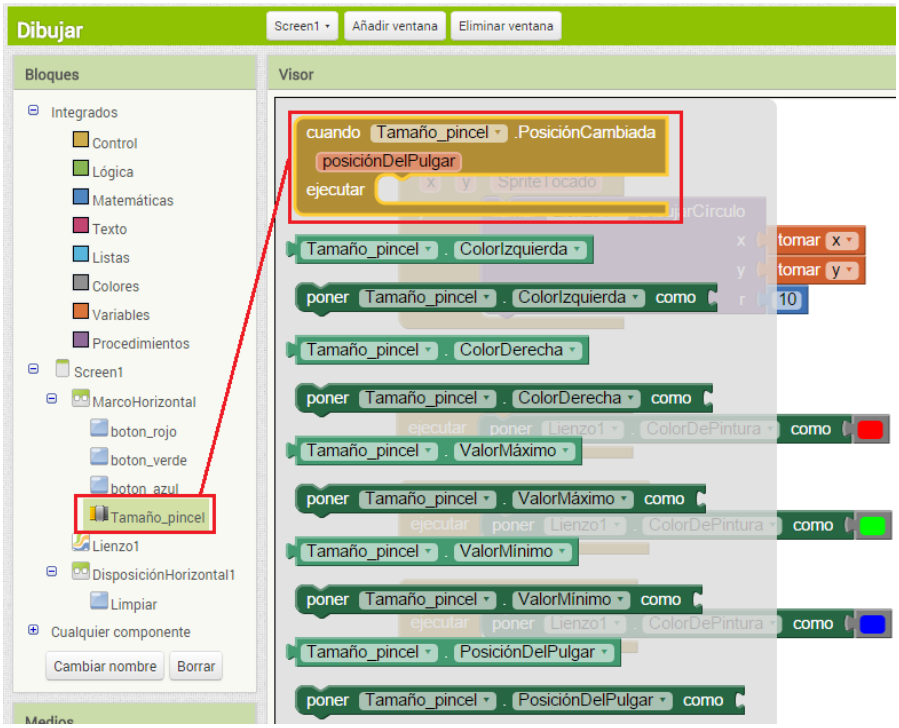
Un objeto **Deslizador** es perfecto para modificar el valor numérico de un parámetro sin tener que escribirlo. Para incluir un deslizador en nuestro interfaz de usuario tenemos que arrastrarlo desde la Paleta. En nuestro caso lo llamaremos **Tamaño_pincel**.



Tenemos que definir el valor mínimo y máximo que podrá tomar el deslizador, y el valor que tendrá inicialmente (**PosiciónDelPulgar**). También es conveniente especificar la anchura del Slider para que se ajuste al espacio restante (**Ajustar al contenedor**).

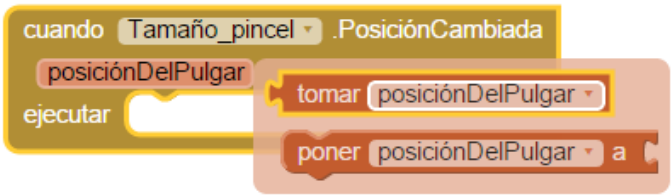


Tenemos que indicarle ahora a nuestro programa que modifique el ancho de la línea de nuestro dibujo según variemos la posición de **Tamaño_pincel**. Para ello, utilizaremos el bloque mostaza **cuando.Tamaño_pincel.PosiciónCambiada**, que se ejecuta cada vez que se cambia la posición del selector dentro del deslizador.

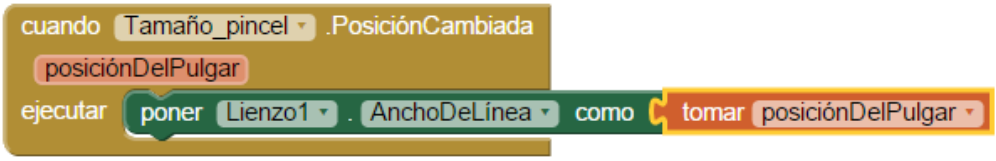


Este bloque incluye una variable, **posiciónDelPulgar**, que contiene el valor actual de la posición, y que puede utilizarse para nuestro propósito.

Si dejamos momentáneamente el puntero del ratón sobre la casilla **posiciónDelPulgar** del bloque mostaza, se mostrará un desplegable que nos permitirá elegir el bloque que define el valor de esa variable (**tomar**), o definir un nuevo valor para ella (**poner**).



Elegiremos en este caso el bloque **tomar**, y lo combinaremos con el bloque verde del objeto **Lienzo** que define el ancho de línea de dibujo, para indicar al programa que cambie el ancho de la línea cada vez que arrastremos el puntero del deslizador **Tamaño_pincel** a un nueva posición.



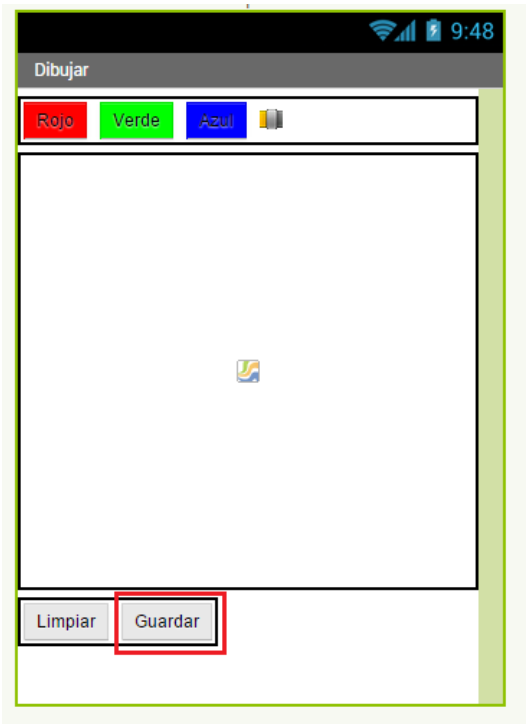
Igual que ocurre con los bloques de color naranja **tomar** y **poner** que apareccen dentro de algunos bloques mostaza, y que sirven para poner o tomar un valor (en este caso para **posiciónDelPulgar**), en muchos componentes existen bloques de color verde, que nos permiten tomar o poner valores a sus propiedades.

Por ejemplo, el siguiente bloque nos permite **poner**, asignar, definir el valor del ancho de línea del componente **Lienzo1**. La propiedad **AnchoDeLínea** de **Lienzo1** tendrá entonces el valor del bloque que encajemos a la derecha del bloque verde oscuro.

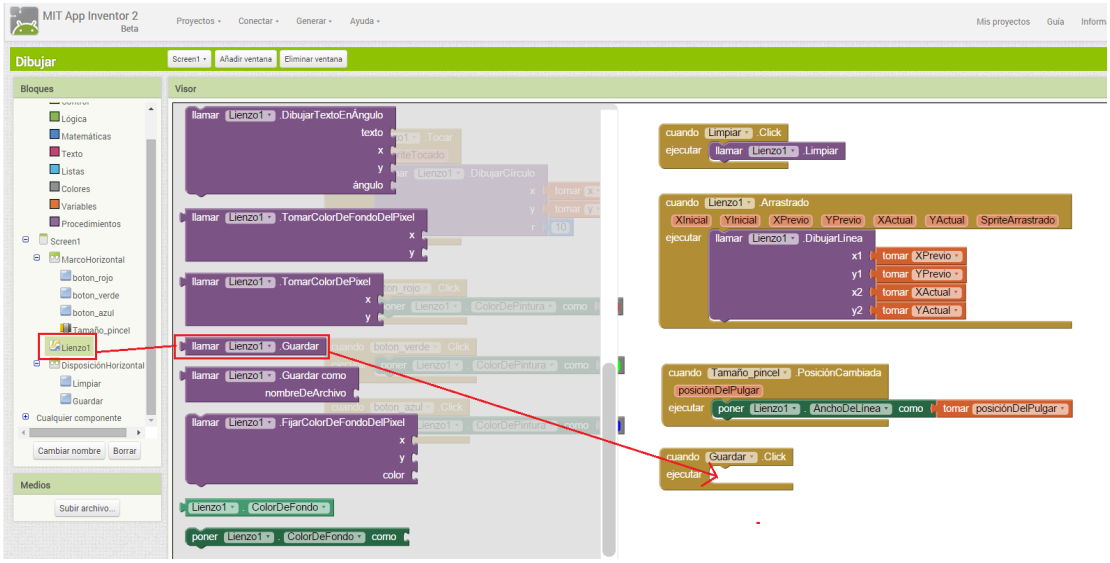
Y este otro sirve para **tomar** el valor actual del ancho de línea del componente **Lienzo1**. Es decir, cuando encajemos este bloque verde claro a la derecha de un bloque verde oscuro, éste tomará el valor de la propiedad **AnchoDeLínea** de **Lienzo1**.

Guardar un archivo con el dibujo que hemos hecho

Cuando el usuario termine de hacer un dibujo interesante, seguramente querrá conservarlo en la memoria permanente del teléfono. Para ofrecerle esa posibilidad, tendremos que indicarle a nuestro programa cómo almacenar en la memoria el contenido actual del lienzo. En primer lugar, hay que incluir en el interfaz un botón **Guardar**, que al ser pulsado ejecutará el código correspondiente.

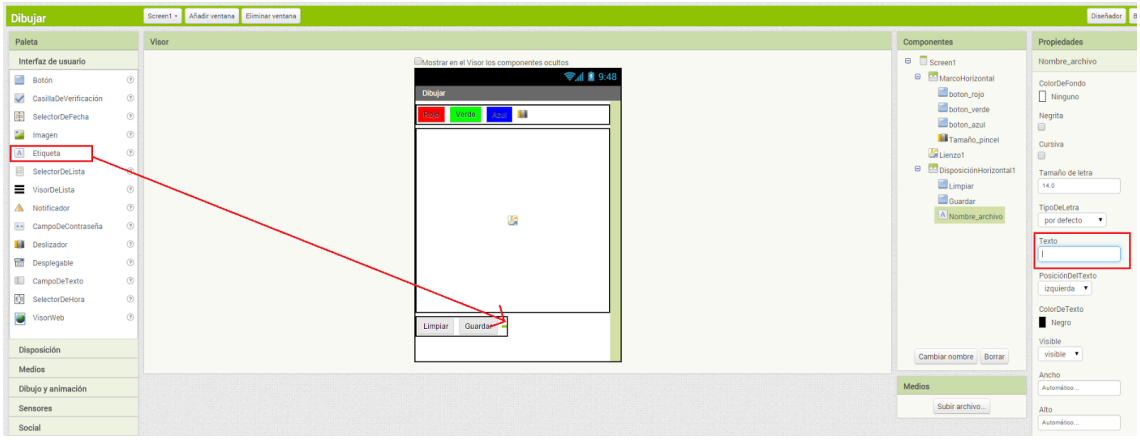


El bloque de código que usaremos, incluido dentro del cajón Lienzo, será **llamar.Lienzo1.Guardar**.



Este bloque es diferente a los vistos hasta ahora, porque no puede ser encajado directamente debajo de otro bloque, o dentro de un bloque color mostaza. El bloque tiene que ser encajado a otro por su izquierda. Esto se debe a que es un bloque que devuelve un valor, como acabamos de ver al final del apartado anterior, con el bloque de color verde claro **Lienzo1.AnchoDeLínea**. En este caso el bloque violeta devuelve una texto, que contiene el nombre del archivo donde se ha almacenado el dibujo dentro de la memoria del dispositivo.

Es conveniente colocar un componente Etiqueta, que llamaremos **Nombre_Archivo,** a la derecha del botón **Guardar**, y que tomará el valor de texto devuelto por el procedimiento **Lienzo1.Guardar**. Definiremos que su propiedad **Texto** esté inicialmente en blanco. El componente será difícil de ver en el dispositivo mientras no le asignemos un nuevo valor.



Veamos cuidadosamente el nuevo componente y los bloques utilizados, hasta interiorizar mentalmente su funcionamiento.

Cada vez que se pulse el botón **Guardar**, el dibujo quedará almacenado con un nombre que se asignará automáticamente, en función del día y la hora. Este nombre aparecerá en el texto de la etiqueta **Nombre_archivo** cada vez que hagamos clic en el botón **Guardar**.

Ideas para mejorar la aplicación

Podemos mejorar esta aplicación hasta donde queramos, usando la imaginación, e investigando cómo podemos utilizar los recursos en App Inventor para incluir en nuestra aplicación todo lo que vayamos inventando.

Por ejemplo...

- Poner un sello con tu nombre cuando pulses un botón
- Tomar una foto existente en la memoria del dispositivo y usarla como fondo del lienzo, etc.

Sesión #3

Objetivos para hoy

1. Hacer una aplicación para adivinar qué número está pensando
2. Descargarla en nuestro móvil

Para ello aprenderemos...

1. Qué es una variable y cómo manejarla (**tomar** y **poner**)
2. Comprenderemos cómo se genera un número aleatorio (*random*)
3. Conoceremos qué es una instrucción condicional (bucle **si-entonces**)
4. Usaremos instrucciones de comparación

¿Qué número está pensando?

Vamos a hacer un programa sencillo pero eficiente. Él pensará un número y nosotros tenemos que adivinarlo. ¡Parecerá que el móvil piense!

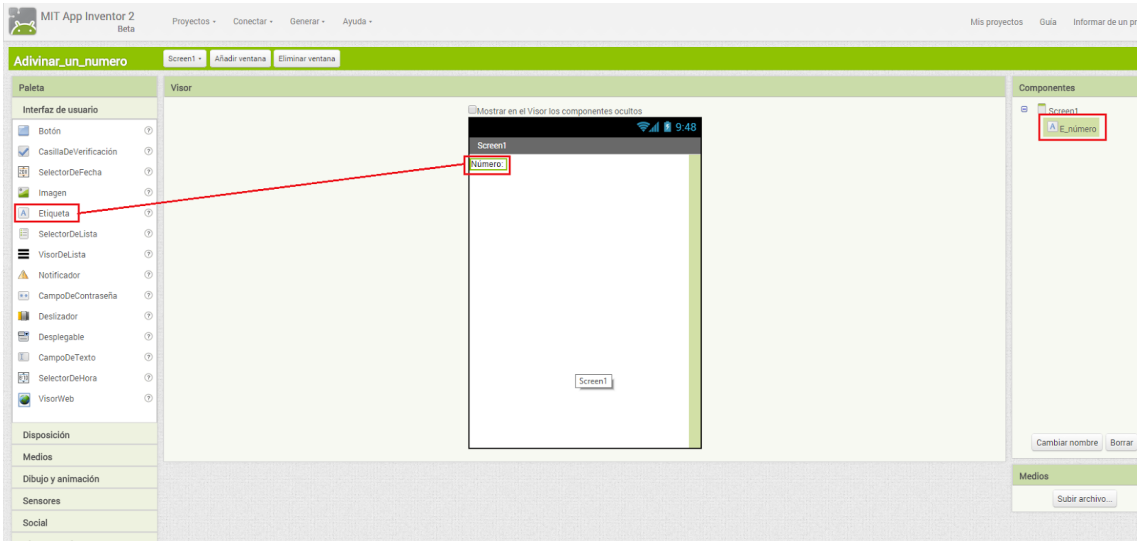
Para que el dispositivo pueda hacer esto correctamente, tenemos que enseñarle cómo hacerlo, paso a paso, añadiendo bloques de código. Se trata de enseñarle al programa la lógica que seguimos nosotros, los seres humanos, cuando jugamos a este juego.

Definimos la interfaz del juego

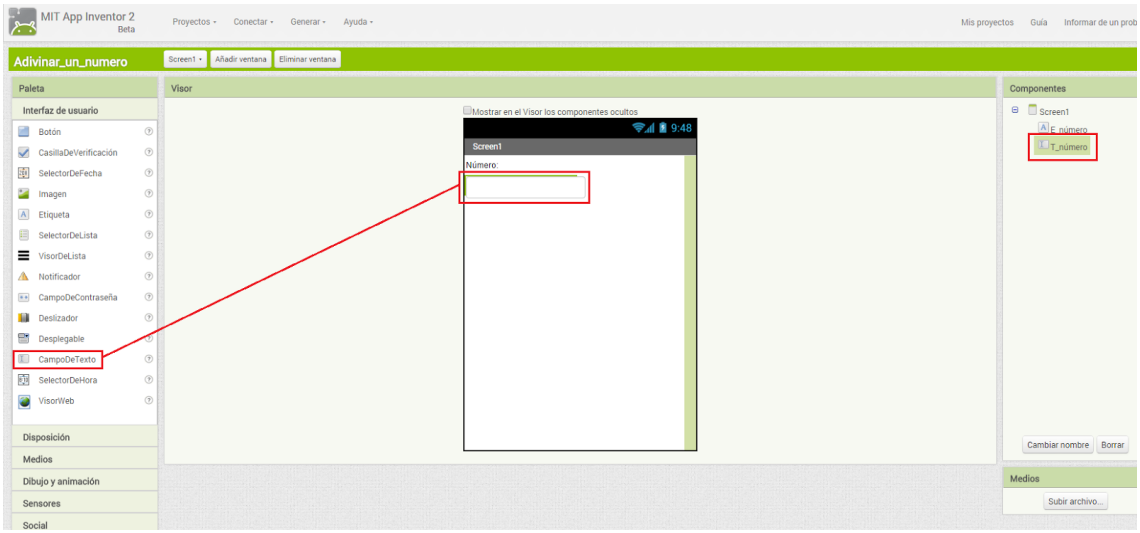
Tenemos que crear un interfaz para para que la aplicación se relacione con el usuario, es decir, le pregunte un número, y le vaya dando pistas, diciendo si es demasiado alto, demasiado bajo, o si finalmente ha acertado el número secreto.

Creamos entonces en el Diseñador una etiqueta, y cambiamos el texto para que aparezca en ella la palabra “Número:”.
Uno de los retos más importantes cuando programamos, como hemos dicho antes, es ser ordenados. Los programadores tendemos a crear y definir muchos objetos para llevar a cabo rápidamente la idea que hemos tenido, antes de que se nos vaya de la cabeza. Aunque parezca imposible, con el tiempo, cuando revisamos el código, ya no sabemos para qué servía cada objeto que hemos creado, y esto nos puede hacer perder mucho tiempo cuando revisamos nuestro programa un mes más tarde. Para evitarlo, debemos acostumbrarnos a seguir prácticas o métodos que nos libren de este problema a largo plazo. El futuro parece lejano, pero llega.

A partir de este proyecto vamos entonces a intentar ser más ordenados. Voy a dejar una E_ delante del nombre mi nueva etiqueta. Así siempre sabré, cuando vea los bloques, que esa es una etiqueta, y no un botón, ni campo de texto, ni otro tipo componente. Llamaré a la etiqueta **E_número**.

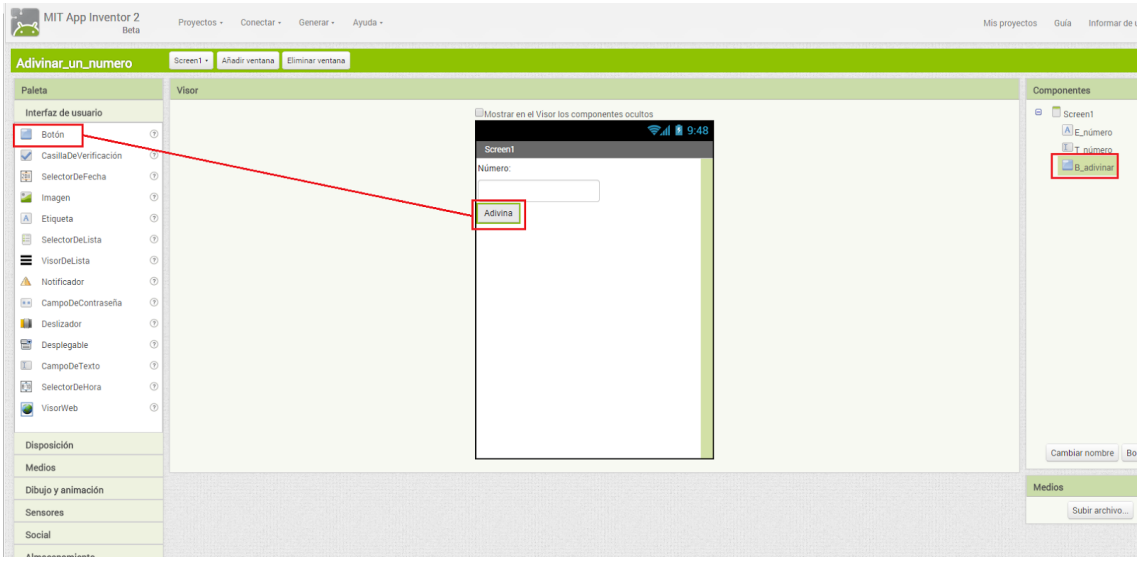


Además el dispositivo va a tener que "escuchar" cada número que le digamos, para decidir si hemos acertado, así que añadiremos en nuestra ventana del visor un campo de texto. Servirá para que el jugador indique qué número cree que ha pensado el programa.



Para diferenciarlo de la etiqueta que hemos creado antes, al campo de texto le llamaremos **T_número**. Así cuando estemos en el editor de bloques podremos diferenciar fácilmente entre la etiqueta y el campo de texto, porque uno comienza con E_ y otro con T_. Aunque parezca innecesario, esta costumbre en la asignación de nombres puede ser muy interesante. Con el tiempo cada programador va desarrollando sus propios métodos, o tomándolos de otros programadores, para disfrutar del "arte" de programar sin complicarse la vida.

Finalmente tendremos que añadir un botón con el texto "Adivina" para que el dispositivo sepa cuál hemos elegido. A este botón lo llamaremos **B_adivinar**. Cuando el jugador lo pulse el programa tendrá que hacer algunas comprobaciones que ahora iremos viendo.



Ya está. El dispositivo podrá escuchar lo que el jugador le diga.

Generamos un número aleatorio

Cuando las personas jugamos a este juego, ¿qué es lo primero que hacemos cuando le decimos a un amigo que adivine qué número estamos pensando? Pensamos un número al azar.

Para hacer lo mismo en nuestra aplicación necesitamos crear código, así que vamos al editor de bloques.

Lo primero que el programa tiene que hacer es pensar en un número. En este caso le vamos a indicar cómo debe pensar en un número entre 1 y 10. Al ser una instrucción matemática, abriremos el cajón **Matemáticas**, y usaremos el bloque **entero aleatorio**. Tendremos que especificarle entre qué dos números debe pensar su número.

A continuación vamos a hacer que el programa guarde este número en su memoria, porque si no lo olvidaría y no podríamos jugar.

Veamos que este tipo de bloque tiene que ser encajado a la derecha de otro. Esto es porque el resultado de este bloque, el número entre 1 y 10, será el dato de entrada para otro bloque.

Guardamos el número en una variable

Una variable es un espacio de la memoria del dispositivo reservado para guardar datos que nuestros programas tienen que manejar durante su funcionamiento. Para poder utilizar las variables de memoria es necesario en primer lugar darles un nombre.

Para ello abrimos el cajón **Variables** y elegimos el bloque **initialize global ... to**. Podemos darle a la variable el nombre **V_número_pensado**. Es importante dar a las

variables un nombre descriptivo, porque en programas más complejos, con más variables, nos facilitará saber para qué sirve cada una. Como venimos haciendo, y para identificar rápidamente que se trata de una variable, el nombre comienza con una V_.

Los bloques deben quedar así:

Con estos dos bloques le hemos dicho al el juego que tiene que pensar un número entre uno y diez, y guardárselo en una variable de su memoria, sin mostrárselo al jugador.

ATENCIÓN
Una variable de memoria es como una caja dentro de un gran armario lleno de cajas, que es la memoria total del ordenador. La memoria total del ordenador está compuesta por millones de estas pequeñas cajitas de memoria, que sirven para guardar la información que el ordenador recibe del exterior, y la que él mismo genera durante la ejecución de las aplicaciones.
También se guarda en la memoria el propio programa que está ejecutándose.

¿Cuál sería el paso siguiente en el juego?

Pedimos un número al jugador

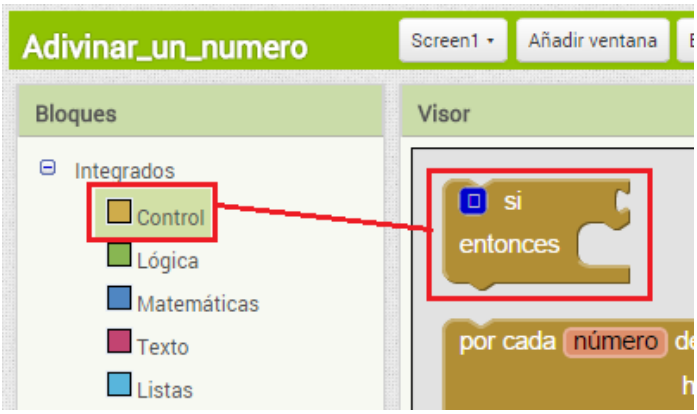
Para comenzar a jugar deberíamos pedirle al jugador que nos diga un número. Esto lo conseguiremos por medio del campo de texto **T_número** que hemos creado en el Visor. El jugador irá escribiendo números en este campo y el programa le irá indicando si el número aleatorio generado secretamente es mayor o menor al que el jugador ha escrito.

Por lo tanto, cada vez que el jugador escriba su número y pulse el botón **B_adivinar** el programa lo comparará con el número secreto.

Hacemos comparaciones con la instrucción si-entonces

Ahora que ya tenemos el número pensado y el número que ha elegido el jugador ¿cuál será el siguiente paso? La aplicación tiene que comparar ambos números, para saber si ha acertado, o si es mayor o menor.

Este bloque tan importante es el que nos permitirá a los programadores enseñarle al dispositivo cómo debe tomar las decisiones durante la ejecución de un programa. El bloque **si-entonces** bloque sirve para darle inteligencia a las aplicaciones. Usaremos este bloque muchísimas veces cuando hagamos programas.



Dependiendo del resultado de la comparación la aplicación deberá hacer una cosa u otra:

> ***Si*** *se cumple una condición*
> ***Entonces*** *Ejecuta esto*

Con un ejemplo se entenderá más fácilmente:

> ***Sí*** *numero_x>2*
> ***Entonces*** *Escribe "numero_x es mayor que dos"*

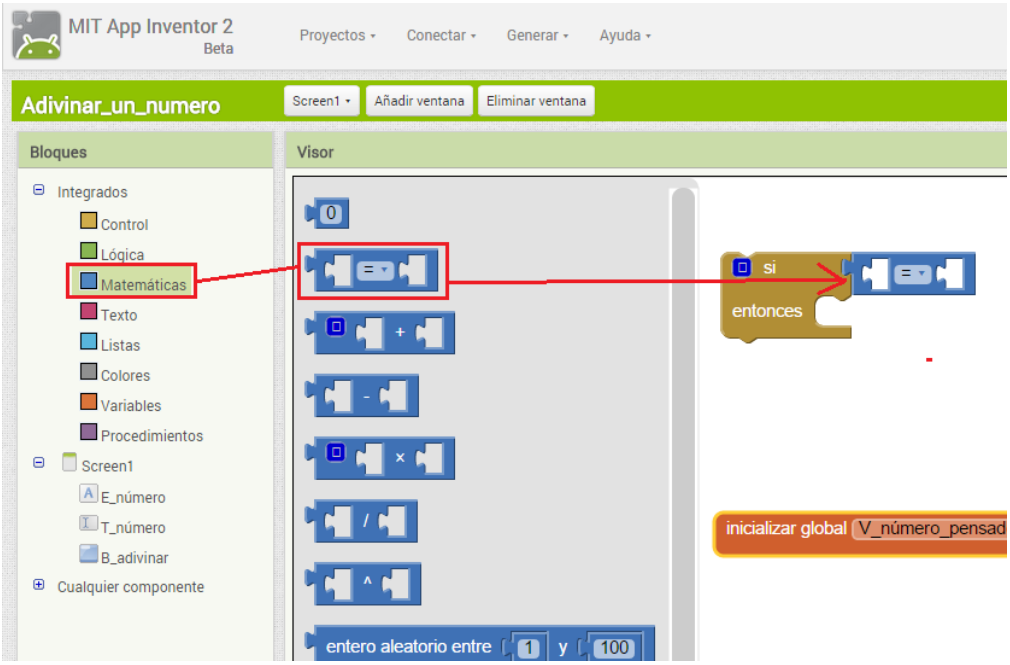
Al bloque **si-entonces** se le tienen que encajar dos bloques más por el lado derecho. El primer encajador, el que sigue a **si**, sirve para indicar la expresión que se va a evaluar. En el caso de nuestro programa, vamos a comparar dos números, el pensado y el que ha dicho el jugador.

El segundo enchufe, el que sigue a **entonces**, sirve para decirle a la aplicación qué tiene que hacer si se cumple la condición que hemos puesto en el **si**.

Primero definimos la parte del **si**, la expresión a evaluar, a comprobar.

Comparamos los números

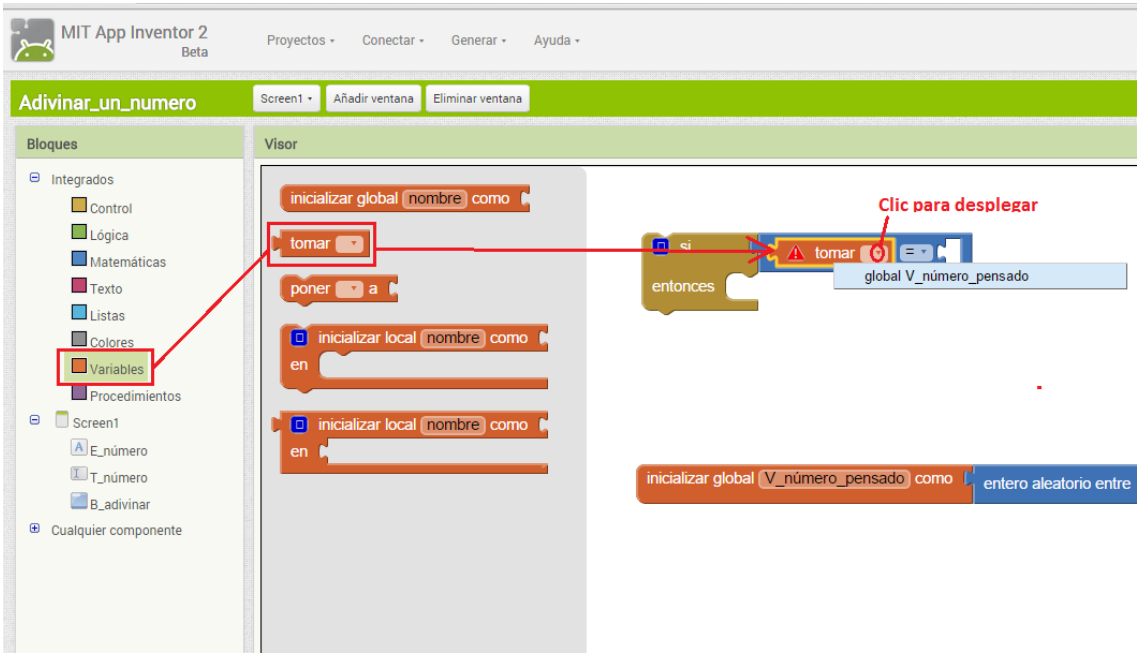
El bloque que sirve para hacer una comparación entre dos números está en el cajón **Matemáticas**. Lo arrastraremos al editor de bloques, hasta encajarlo con el primero de los huecos del bloque **si-entonces**.



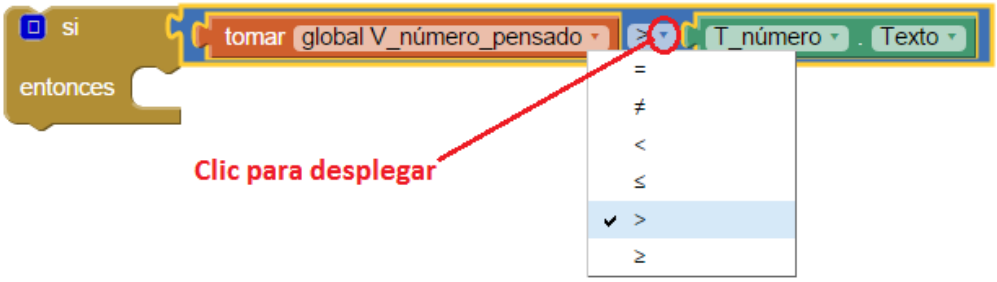
En el primer hueco del bloque azul pondremos el número secreto que almacenamos antes en la variable, y en el otro lado indicaremos cuál es el número que ha escrito el jugador en el campo **T_número**. Antes de hacer esto tenemos que saber cómo manejar el contenido de una variable.

Bloque tomar para conocer el valor de una variable, texto o etiqueta

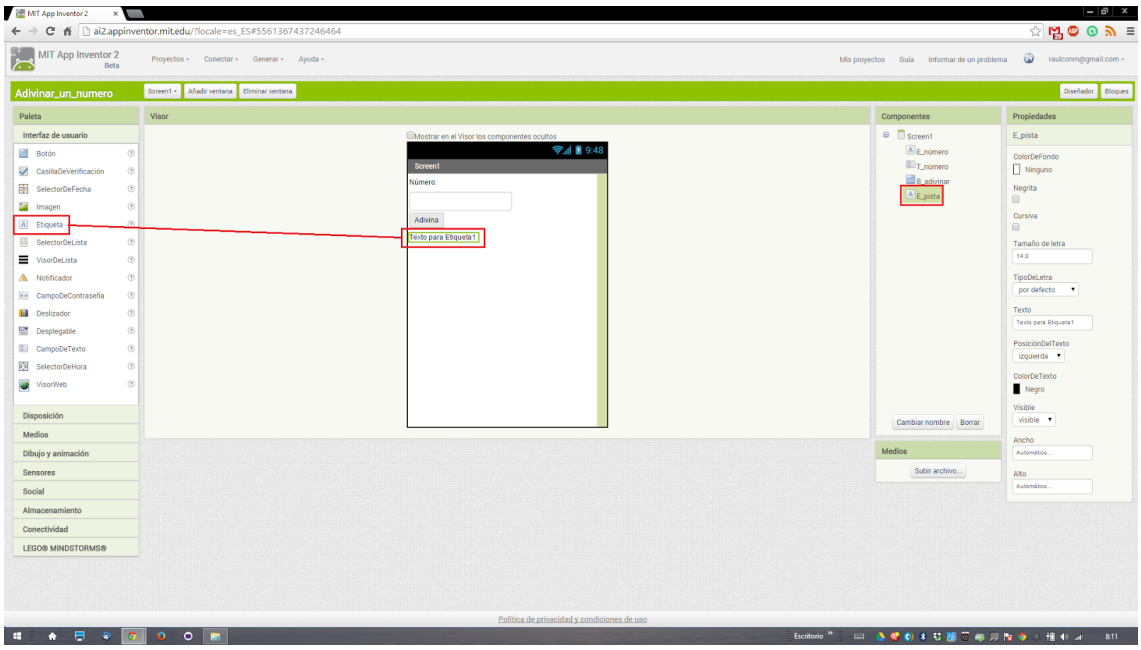
Podemos saber cuál es el contenido de una variable con el bloque **tomar**. Arrastramos al editor el bloque **tomar** que se encuentra dentro del cajón **Variables** hasta el primer hueco del bloque azul de comparación, y una vez colocado elegimos qué variable es la que queremos usar, desplegando la lista de variables situada dentro del bloque **tomar**.



Ahora que ya sabemos cómo consultar cuál es contenido de la variable **V_numero_pensado**, podemos modificar la comparación haciendo clic en el centro del bloque azul. Empezaremos evaluando si el número pensado es mayor que el número del jugador.



En caso de que se cumpla la condición deberemos darle al jugador una pista, diciéndole que el número que nos ha dicho es demasiado bajo. Lo haremos creando una nueva etiqueta **E_pista** en el interfaz del juego.



Bloque poner para guardar el valor de una variable, texto o etiqueta

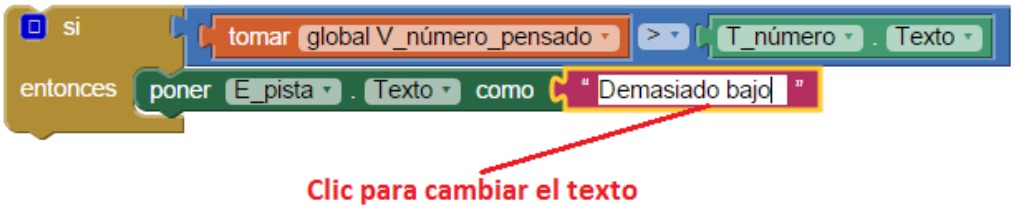
Igual que con el bloque **tomar** tomábamos el valor de la variable, con el bloque **poner** vamos a asignar el valor "Demasiado bajo" al texto de la etiqueta **E_pista,** para que el jugador lo vea en la pantalla del juego.

ATENCIÓN

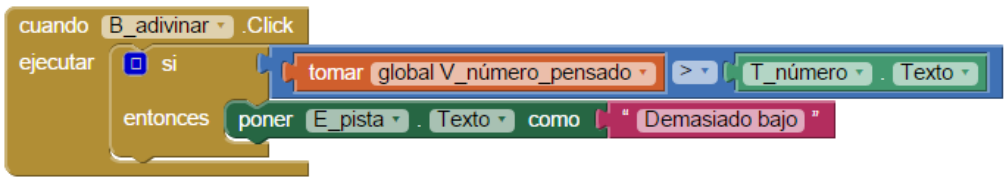
Utilizaremos el componente **Etiqueta** cuando el valor de su propiedad **Texto** sólo va a ser modificada por el programa, es decir, cuando el usuario de la aplicación no tiene que escribir sobre ella para modificarlo. En caso de que el usuario pudiera modificarlo utilizaríamos un componente **CampoDeTexto**.

Para incluir el texto "Demasiado bajo" usaremos un bloque fucsia que hay dentro del cajón **Texto**.

Y ahora, en el editor, crearemos los bloques para completar la instrucción **si-entonces**.



¿Cuándo debemos hacer la comparación? Cada vez que el usuario haga clic en el botón **B_adivinar**. Por lo tanto, incluiremos todo el bloque anterior dentro del bloque **cuando.B_adivinar.Clic**



Pero esto sólo dará la pista al jugador cuando su número sea menor al pensado, así que habrá que hacer tres bloques **si-entonces**, uno para cuando el número sea menor, otro para cuando sea mayor, y el último para cuando sea igual, en cuyo caso el jugador habrá acertado el número.

Observemos que en cada bloque azul el signo de comparación será diferente. Podemos duplicar los bloques **si-entonces** completos y luego modificar los bloques azules y fucsia para que se adapten a cada comparación.

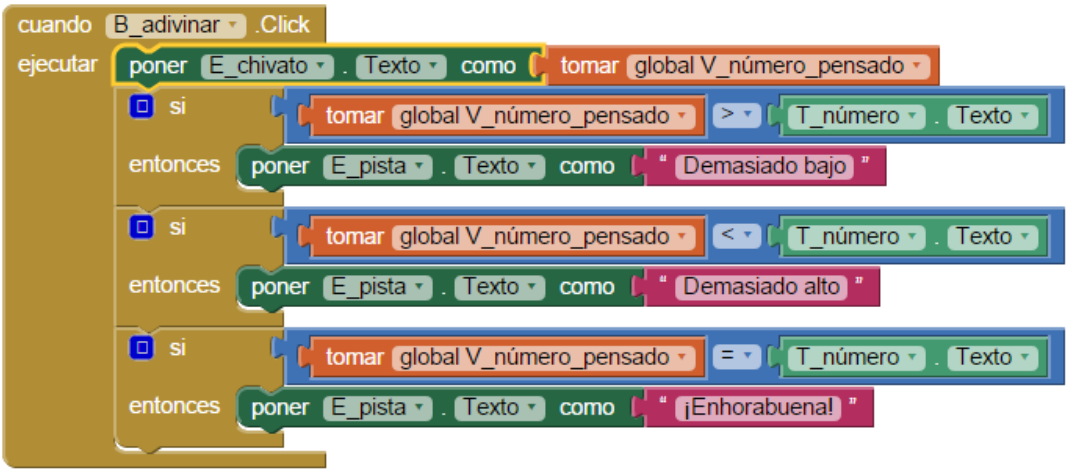
Sucede que como programadores, cuando estamos probando la aplicación, no sabemos cuál es el contenido de la variable **V_número_pensado**, así que no tenemos certeza de si el programa está ejecutándose bien, es decir, si está informando correctamente al jugador si el número es demasiado alto, o demasiado bajo.

Para saber si el programa funciona como debe tenemos que poner "chivatos". Por ejemplo, podemos hacer que se muestre siempre el número pensado en pantalla, para saber si el programa funciona bien. El número no será secreto, pero ayudará al programador a saber si la aplicación está tomando las decisiones como él quiere que lo haga.

ATENCIÓN
Un chivato se utiliza sólo durante la fase de desarrollo del programa, mientras se está escribiendo. Cuando todo está probado, y antes de ponerlo a disposición de los usuarios, hay que acordarse de quitar todos los chivatos. A veces los programadores se olvidan de quitarlos, ¡y eso queda fatal!

Para poner un chivato podemos definir una etiqueta **E_chivato**, y luego usar estos bloques:

De momento, el mejor sitio para ubicar este bloque es dentro del evento **cuando B_adivinar.Clic**, como primera instrucción. Así se actualizará en la pantalla el valor del chivato cada vez que usuario pulse el botón de adivinar.



Bucles si-entonces anidados

Cada vez que el jugador pulsa el botón Adivina el programa hace tres comparaciones seguidas (es menor, es mayor, y es igual). Sin embargo, sólo una de ellas será cierta, así que no es necesario hacer las tres preguntas. Es decir, si se cumple la primera condición no es necesario evaluar la segunda condición, ni la tercera. En realidad, no solamente no es necesario hacer las tres preguntas, sino que no es tampoco conveniente, porque la aplicación hará siempre trabajo inútilmente, haciendo la aplicación más lenta, menos eficiente.

ATENCIÓN

Es importante que como programadores intentemos escribir un código tan eficiente y limpio como podamos. Para ello hay que pensar un poco cuál es la mejor manera de llevar a cabo un proceso concreto, evitando líneas innecesarias o redundantes, que afean el código. La programación tiene un toque "artístico", y como tal hay es preferible buscar siempre la elegancia y la belleza.

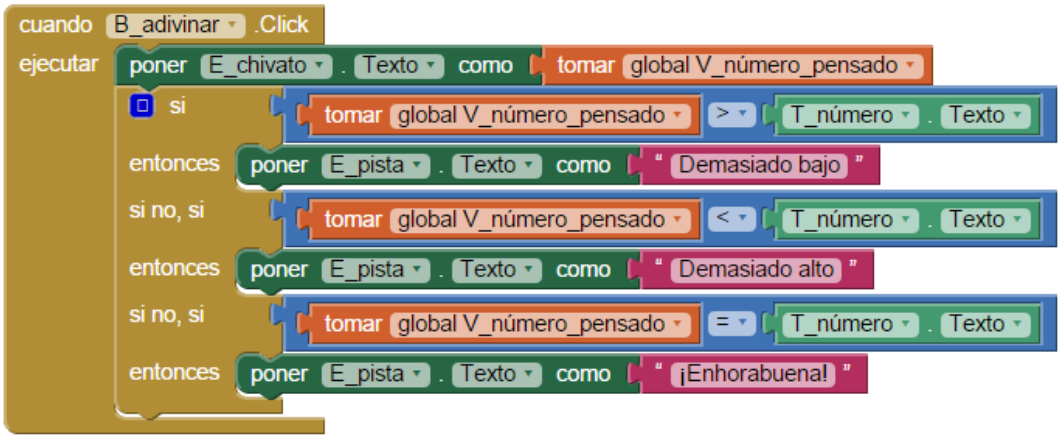
En nuestro programa, para evitar las ejecuciones de código innecesarias, vamos a utilizar sentencias **si-entonces** anidadas, o **si-entonces-si no**. Sólo se ejecutarán los bloques del **si no** cuando NO se cumpla la condición del **si-entonces** anterior.

Si se cumple una condición
Entonces Ejecuta esto
Si no Ejecuta esto otro

Con un ejemplo sería

Sí numero_x>2
Entonces Escribe "numero_x es mayor que dos"
Si no Escribe "numero_x es menor o igual que dos"

En este caso concreto, vamos a enlazar un **entonces** con el siguiente **si**, de modo que sólo se evaluará el segundo **si** cuando el primero no se haya cumplido. Y sólo se evaluará el tercer **si** en el caso de que el segundo tampoco se cumpla.



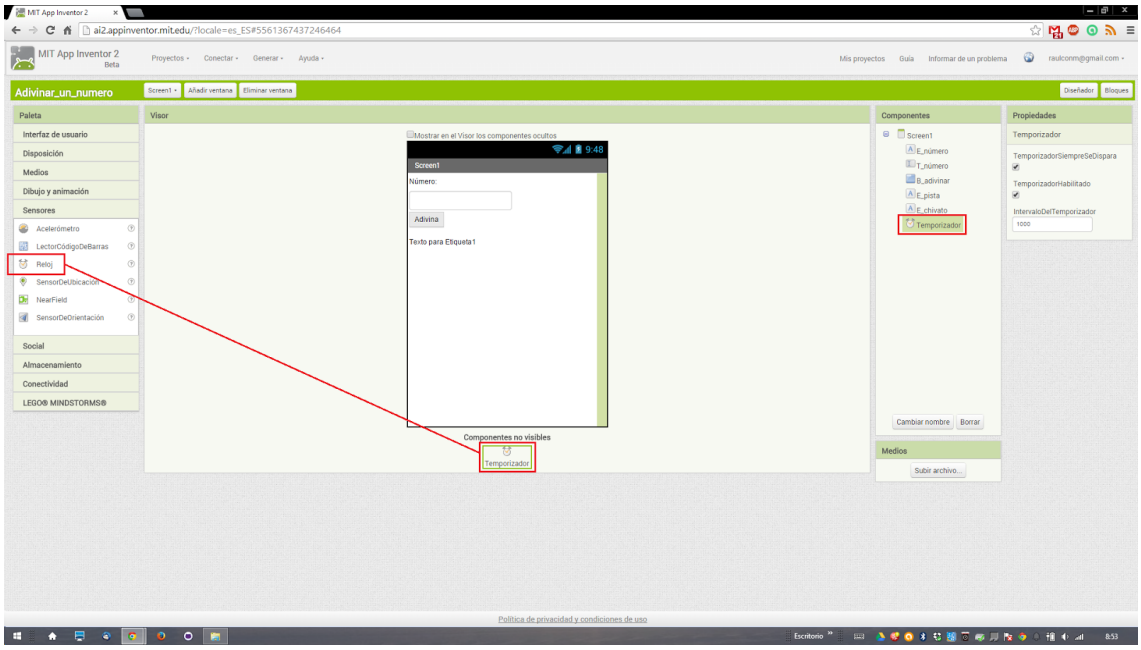
De esta forma el bloque **cuando.B_adivinar_Clic** se ejecutará mucho más rápido, al evitar pasar por bloques sin necesidad.

Uso de un reloj para calcular el tiempo

Un componente muy útil para controlar y gestionar la ejecución de los programas es el temporizador, o reloj. Gracias a este componente podemos definir cuándo suceden cosas, independientemente de lo que haga el usuario de la aplicación. O podemos controlar el tiempo que duran los procesos que se están ejecutando, por ejemplo.

En este caso lo utilizaremos para limitar el tiempo de que dispone el usuario para adivinar el número pensado. Cuando el tiempo se cumpla, el usuario no podrá seguir intentando adivinar el número.

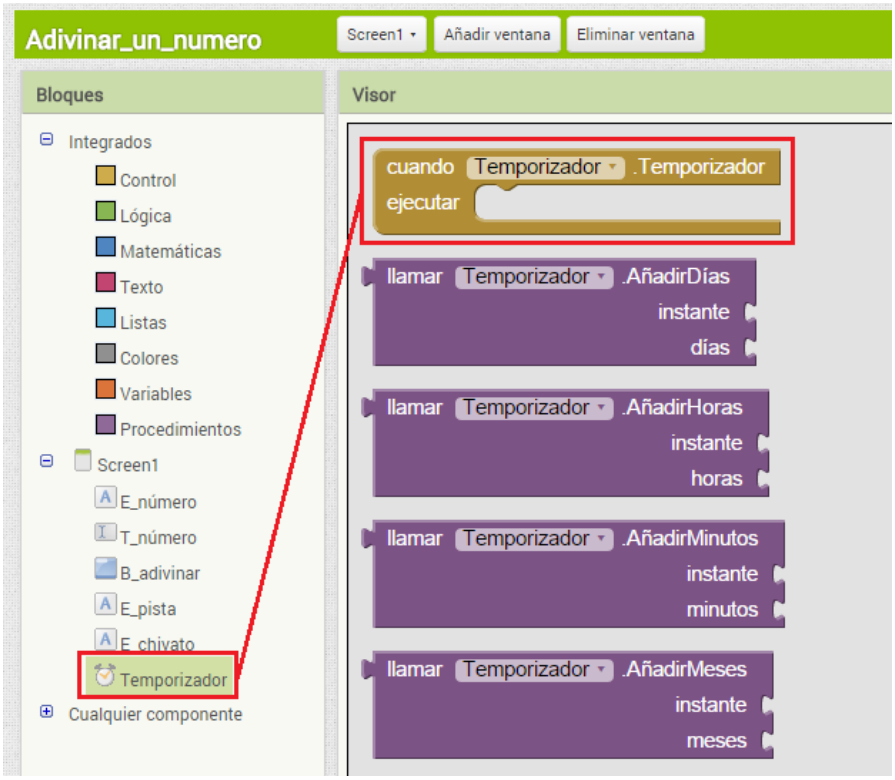
Empezamos añadiendo desde la Paleta al Visor un componente del tipo **Reloj**. Como el reloj es un componente no visible, aparecerá fuera de la zona visible del Visor. Lo llamaremos **Temporizador**.



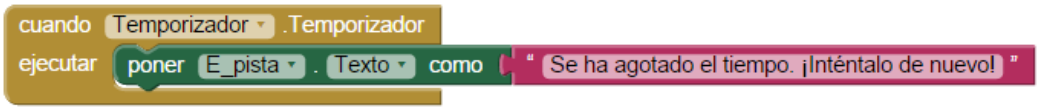
Una de las propiedades más importantes del reloj es **IntervaloDelTemporizador**, que controla cada cuánto tiempo se "dispara" el reloj. Se expresa en milisegundos, y su valor inicial es 1000, o lo que es lo mismo, un segundo. Esto significa que cada segundo el reloj dirá "¡Ha pasado una unidad de tiempo!". Si cambiamos su valor por 500, por ejemplo, el reloj avisará cada medio segundo. Si ponemos 60000, entonces nos avisará cada minuto, porque 1000 milisegundos por 60 es exactamente un minuto.

Para esta aplicación lo definiremos con un valor de 30000, y así le daremos al jugador la posibilidad de intentar adivinar el número durante treinta segundos.

Para saber cuándo se cumplen los treinta segundos usaremos el bloque mostaza **cuando Temporizador.Temporizador ejecutar**, que se encuentra dentro del cajón del reloj, en el editor de bloques. Todo lo que queramos que suceda cuando se cumplan los treinta segundos habrá que ponerlo dentro de este bloque.

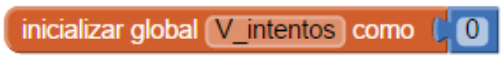


Ahora pondremos los bloques para que el juego nos informe de que han transcurrido los treinta segundos a través de un mensaje de texto. Como llegados a este punto ya no vamos a utilizar más la etiqueta **E_pista**, podemos reutilizarla para mostrar al jugador el texto de tiempo agotado.



Otras propuestas de mejora para la aplicación

Siempre será posible mejorar el juego, para hacerlo más atractivo e interesante para el jugador. Por ejemplo, podemos hacer que el jugador sólo disponga de tres intentos para adivinar el número. Para hacer esto tenemos que crear una variable, **V_intentos**, en la que guardaremos el número de intentos que el usuario ha consumido. Inicialmente esta variable tendrá el valor cero, para indicar que no hemos consumido aún ningún intento.



Después, con un gran **si-entonces**, indicaremos qué hacer cuando aún queden intentos, y qué otra cosa cuando ya se hayan consumido todos.

Es importante que al final del proceso de preguntas se sume uno al número de intentos, para que se reduzcan los intentos restantes para la siguiente vuelta.

Mejoras en la interfaz de usuario

Es muy importante cuidar al máximo el interfaz de usuario de cualquier aplicación, porque de ello dependerá que los potenciales usuarios la encuentren atractiva, o la descarten y no la usen más.

Las aplicaciones de uso complejo normalmente no son muy atractivas, así que hay que esforzarse para que el jugador pueda manejar la aplicación con la facilidad que le gustaría. Tenemos que pensar qué nos gustaría a nosotros que la aplicación hiciera si fuéramos el jugador, cómo nos haría más cómodo su uso, y crear el código necesario para que la aplicación se comporte de esa manera. Por ejemplo, es importante que el jugador sólo pueda meter un número dentro del rango de números entre los cuales está la aplicación está pensando su número aleatorio.

Otra importante mejora sería poner un botón para empezar de nuevo el juego, es decir, para que el móvil piense un nuevo número. En otro caso, el jugador tendría que salir de la aplicación y volver a entrar para que la aplicación pensara un nuevo número para jugar, y seguro que el jugador no estaría dispuesto a hacer esto muchas veces.

Sesión #4

Objetivos para hoy

1. Diseñar y escribir un juego gráfico
2. Probar el juego y definir mejoras
3. Descargarlo en nuestro móvil

Y para ello aprenderemos...

1. Uso de *sprites*
2. Definir y configurar el escenario
3. Uso del sensor **Acelerómetro**
4. Cómo definir y usar procedimientos
5. Implementación de una barra de progreso

Componentes ImageSprite *y* Pelota

Un recurso muy importante disponible en App Inventor es el componente **SpriteImagen**, que se encuentra en la paleta de diseño, dentro del cajón **Dibujo y animación**.

SpriteImagen es un componente muy interesante para crear juegos en los que queramos incluir y manejar objetos gráficos.

Pelota es un tipo específico dentro del conjunto **SpriteImagen**. La única diferencia es que en el caso del componente **Pelota** no podemos cambiar su aspecto, la imagen del objeto, que siempre será una circunferencia. Sí podremos hacerlo sin embargo para cualquier otro **SpriteImagen**.

En general, un *sprite* es una imagen en dos dimensiones, más o menos pequeña, incluida dentro de un escenario más grande, y que ocupa un lugar en la memoria gráfica del ordenador. Se utiliza tradicionalmente para la programación de juegos.

Más adelante se describirán los diferentes bloques que permiten definir el comportamiento de los *sprites* en App Inventor.

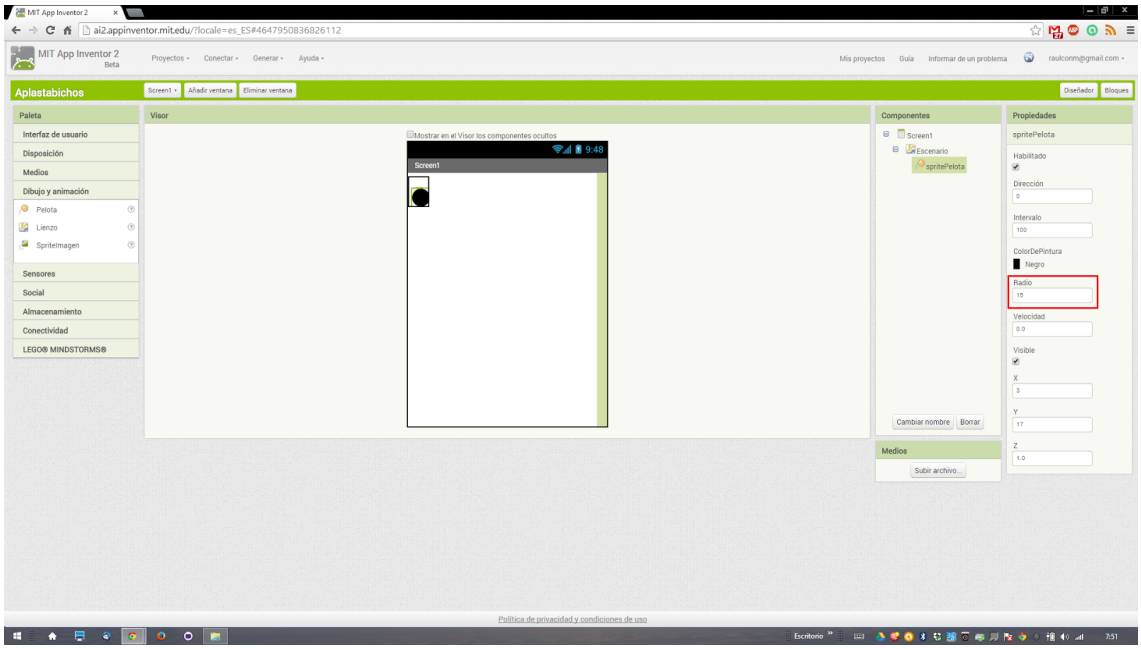
Este tipo de componentes deben "existir" siempre dentro de un componente **Lienzo**, que define el escenario donde se desenvuelve el sprite o sprites. Es decir, primero debemos definir un **Lienzo**, y a continuación incluiremos nuestro sprite dentro de él.

Definición del escenario de juego

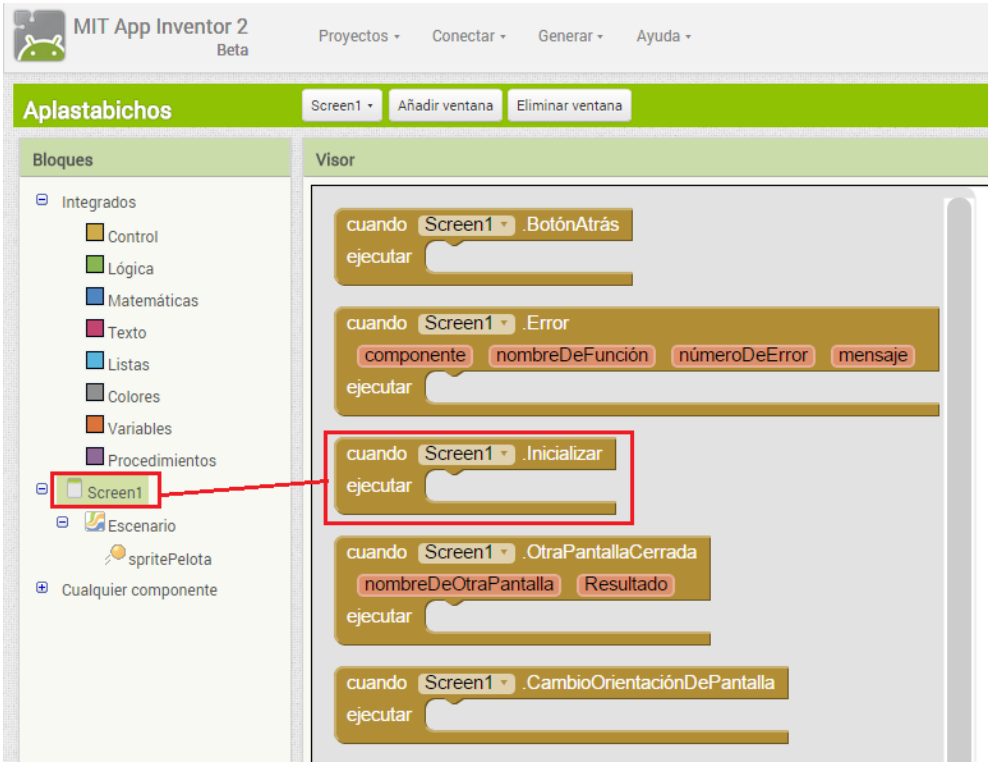
Vamos a programar un nuevo juego un poco más sofisticado, para seguir aprendiendo conceptos y componentes. Se trata de conducir una pelota a través de un escenario de juego para ir aplastando enemigos. Habrá que gestionar el movimiento de la pelota, la ubicación de los enemigos, el marcador, la duración del juego, etc.

Comenzaremos un nuevo proyecto y le daremos un nombre descriptivo, como "Aplastabichos".

Empezaremos creando el lienzo **Escenario**, y dentro él incluiremos un componente **Pelota**, que llamaremos **spritePelota**. De momento indicaremos en el Diseñador que la altura y la anchura del escenario se ajusten automáticamente al contenedor. En cuanto a las propiedades de la pelota, definiremos que su **Radio** es 15, para que se vea suficientemente grande en el escenario.



Para que el escenario siempre ocupe todo el espacio de la pantalla del dispositivo debemos definir algunos bloques dentro del bloque mostaza **cuando.Screen1.Inicializar**. Todo lo que incluyamos en este bloque se ejecutará en cuanto se abra la pantalla, es decir, en este caso será lo primero que suceda cuando se ejecute la aplicación.



Tenemos que adaptar el escenario a los límites de la pantalla, tomando las propiedades de ancho y largo de la pantalla de cada dispositivo. Además, evitaremos que la pantalla rote automáticamente poniendo el valor de la propiedad **OrientaciónDeLaPantalla** a 1. Con este valor la pantalla siempre mantendrá la orientación vertical, aunque inclinemos el dispositivo. No obstante, como esto no funciona con todos los dispositivos, en algunos casos será necesario desactivar manualmente la rotación de la pantalla en el propio dispositivo.

Proporciones y límites. Matemática aplicada

Para diseñar este juego es necesario hacer algunos pequeños cálculos matemáticos. En general, para hacer aplicaciones gráficas siempre es necesario conocer algunos conceptos matemáticos de los que se estudian en el colegio. Es un buen momento para saber cuál es la aplicación de esos conceptos que pensábamos que nunca utilizaríamos.

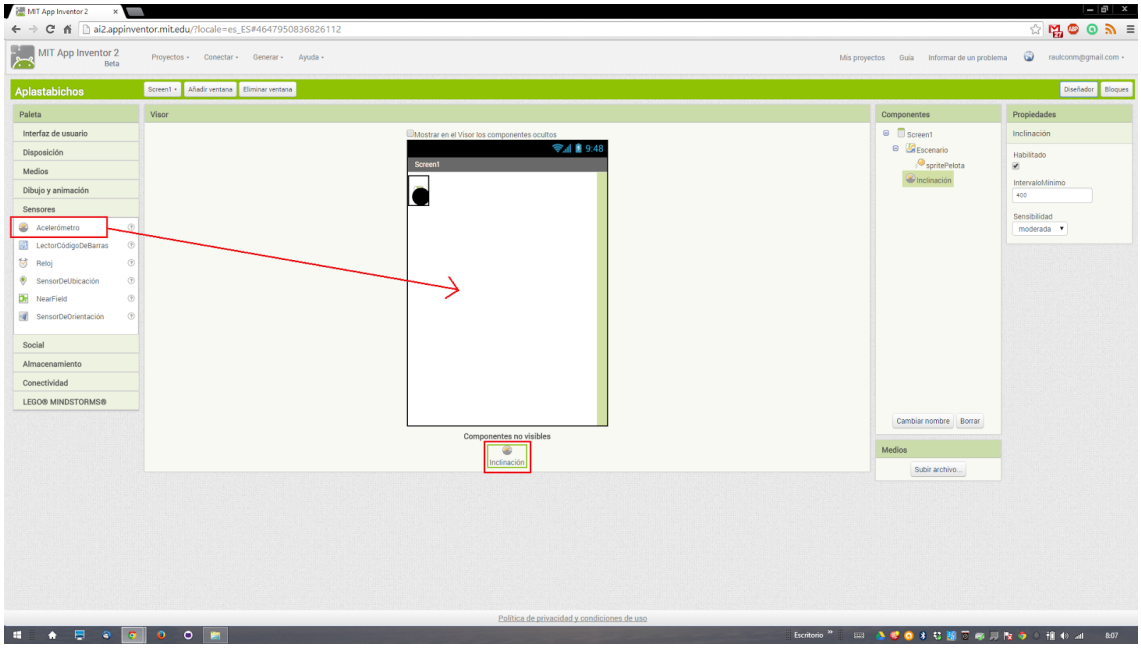
Definir el objetivo del juego

Como decíamos antes, este juego consistirá en ir aplastando con nuestra pelota todos los objetos que aparezcan a lo largo y ancho del escenario. Para que la pelota ruede por el escenario tendremos que inclinar el dispositivo. La pelota rodará siempre hacia la parte del escenario que se encuentre más cerca del suelo. Recogeremos cada objeto cuando la pelota choque con él.

El movimiento de la pelota

Esta parte es muy importante, ya que de ella depende que el juego funcione correctamente y su uso sea agradable para el usuario. Aunque parezca complicado, una vez entendida la lógica, no será difícil crear los bloques que permitan este movimiento.

Lo primero que debemos hacer es incluir en nuestro visor un componente **Acelerómetro**. Lo llamaremos **Inclinación**. Este componente no es visible, así que aparecerá en la parte inferior del Visor, en la zona destinada a componentes no visibles.



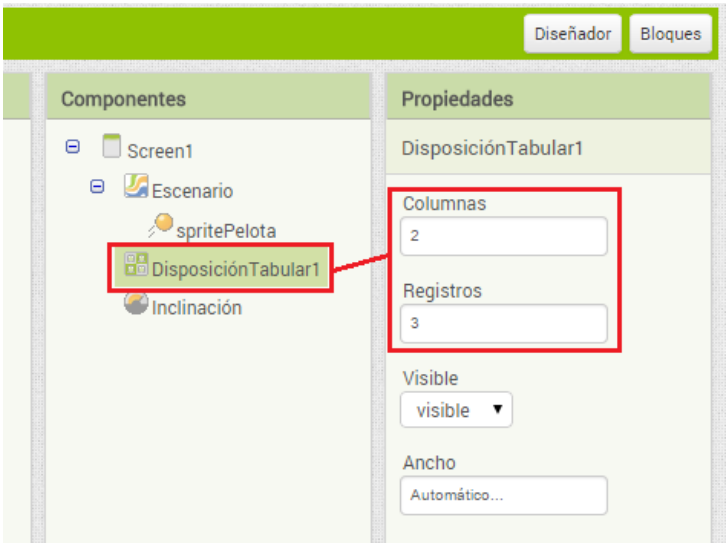
Ahora utilizaremos el bloque **cuando.Inclinación.CambioEnAceleración** para mover la pelota. Este bloque incluye tres variables, cada una de las cuales almacena la inclinación del objeto en uno de los ejes de coordenadas. Vamos a hacer que la pelota tenga una dirección dependiendo de los valores de estas tres variables.

Como vimos en la sesión anterior, es conveniente utilizar “chivatos” durante la fase de programación de nuestras aplicaciones. En este caso mostraremos en pantalla el contenido de estas variables, para conocer cómo están funcionando.

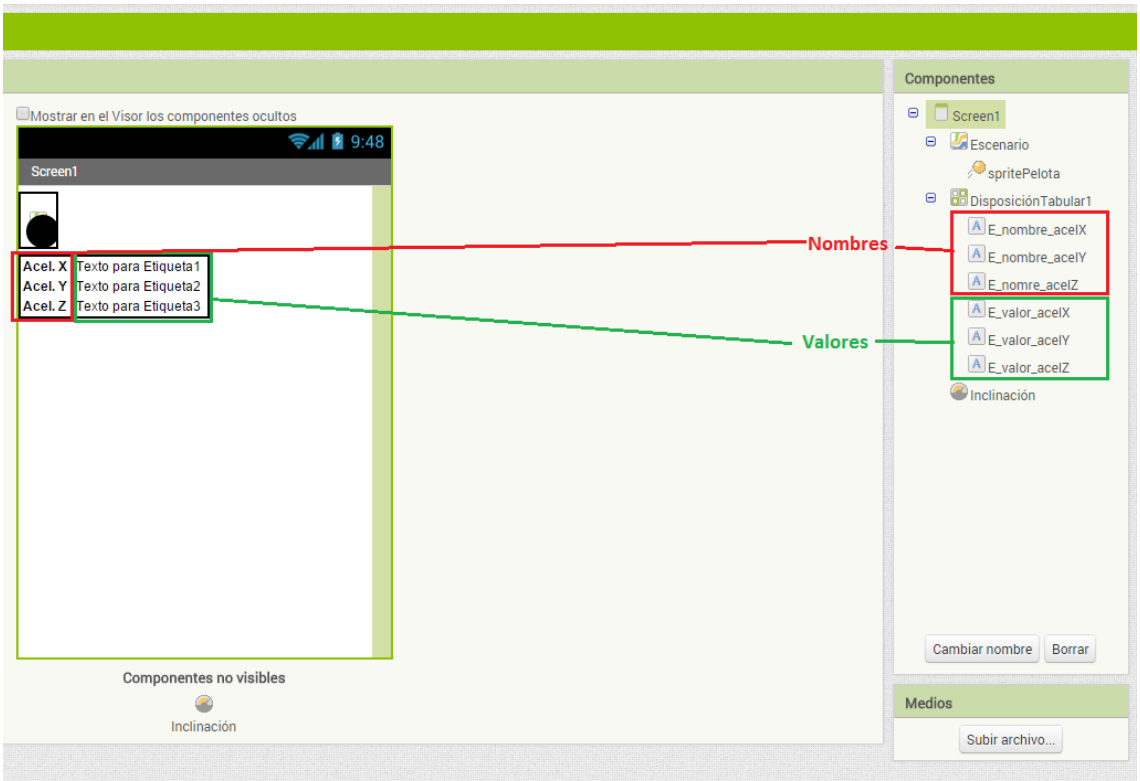
Para ello reduciremos un poco el tamaño del escenario, y colocaremos debajo tres campos **Etiqueta** para mostrar el valor de las variables.

Como la definición de las dimensiones del escenario se hace dentro del **cuando.Screen1. Inicializar**, y este sólo se ejecuta al abrir la aplicación, tendremos que hacer varias pruebas conectando y desconectando el dispositivo a App Inventor hasta encontrar las dimensiones más apropiadas. En nuestro caso hemos reducido la altura del escenario en 150 unidades, para dar espacio suficiente a la tabla, y a otros objetos que añadiremos más adelante.

Facilitaremos la disposición de la información usando un componente **DisposiciónTabular** del cajón **Disposición** de la Paleta de componentes. Como queremos mostrar los datos de tres etiquetas con sus tres valores correspondientes, tendremos que indicar que la **DisposiciónTabular** tenga dos **Columnas** con tres **Registros**.

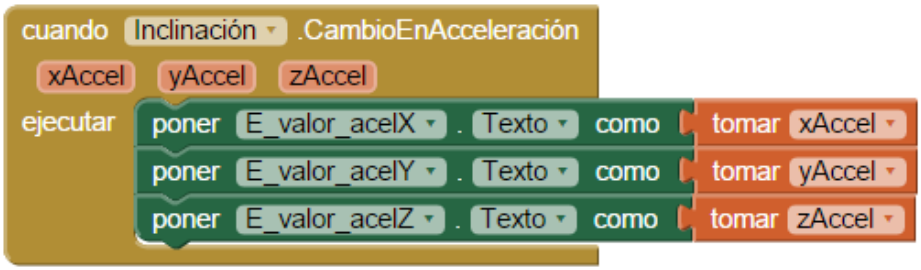


Dentro de la tabla, en la columna de izquierda, colocaremos tres etiquetas con la descripción de cada campo, **E_nombre_acelX**, **E_nombre_acelY** y **E_nombre_acelZ**. En la columna de la derecha pondremos tres etiquetas **E_valor_acelX**, **E_valor_acelY** y **E_valor_acelZ**, para mostrar las variables cuyo valor queremos conocer.



Ya podemos indicarle al programa que muestre los valores de las tres variables dentro de las etiquetas que hemos creado. Recordemos que podemos duplicar bloques y modificarlos cuando vamos a crear varias instrucciones similares.

El dispositivo siempre sabe cuál es la aceleración en cualquiera de los tres ejes. Para saberlo nosotros y utilizarlo en nuestro juego tenemos que utilizar el bloque mostaza **cuando.Inclinación.CambioEnAceleración**. Para conocer la aceleración en el eje X, por ejemplo, tenemos que dejar el puntero del ratón inmóvil durante un segundo sobre el campo **xAccel** color naranja que hay dentro del bloque mostaza. Una vez aparezcan las opciones **tomar** y **poner** para esa variable, podremos arrastrar el bloque **tomar** hasta el hueco disponible a la derecha del bloque color verde oscuro correspondiente.



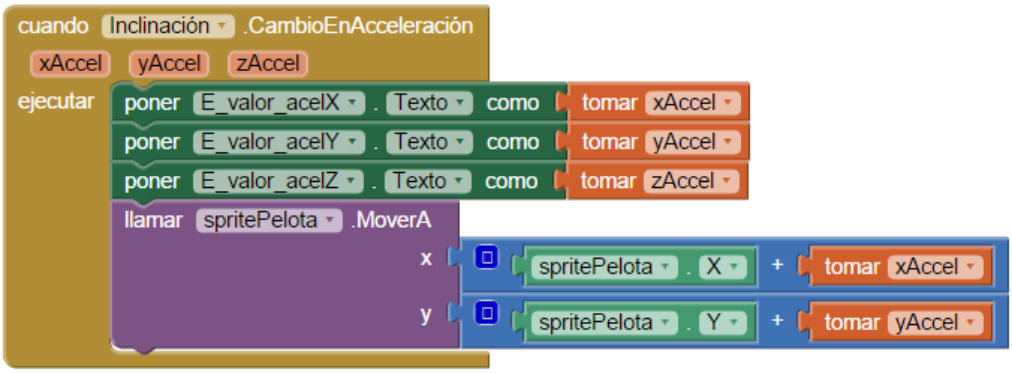
Es buen momento para experimentar qué sucede con cada una de las variables cuando inclinamos el dispositivo. Podemos invertir un poco de tiempo probando, hasta que entendamos cómo afecta la inclinación a cada una de estas variables. Veremos que para la Y los valores comprenderán de -10 (cuando el dispositivo está vertical y apuntando hacia el suelo) a 10 (cuando el dispositivo está vertical, y hacia arriba). Para la X, los valores comprenderán también entre 10 (cuando está completamente

inclinado con la pantalla hacia la izquierda) y -10 cuando está completamente inclinado con la pantalla hacia la derecha).

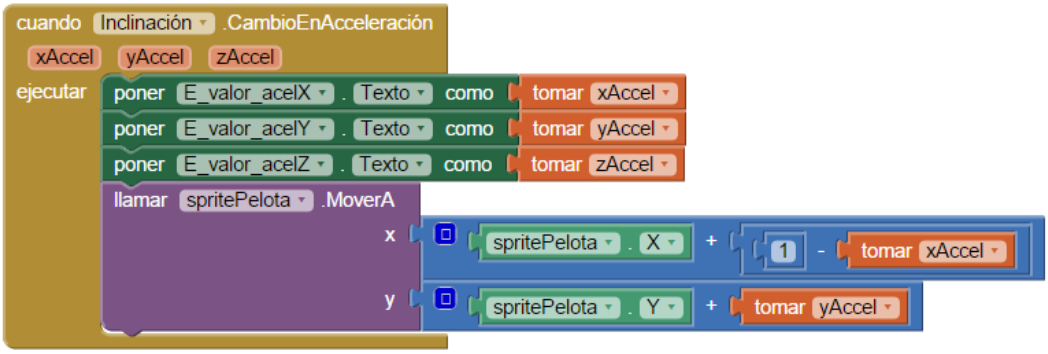
En cuanto a la Z, inicialmente sin uso en este juego, irá de 10 (cuando el dispositivo tiene la pantalla hacia arriba) a -10 (cuando el dispositivo tenga la pantalla hacia abajo, paralela al suelo).

Una vez está claro qué sucede cuando inclinamos el dispositivo, es hora de programar el comportamiento de la pelota. Tendrá que moverse en la dirección en que inclinemos el dispositivo. ¿Cómo? Una pista: hay que utilizar el mismo bloque mostaza del sensor de inclinación que hemos usado antes.

Dentro del mismo bloque mostaza que detecta la variación del sensor de inclinación incluiremos el movimiento de la pelota. Para ello usaremos el bloque violeta **llamar.spritePelota.MoverA**, que sirve para colocar el objeto en cualquier punto del escenario que queramos. Este bloque acepta dos parámetros de entrada, X e Y, que definen las coordenadas donde se colocará la esquina superior izquierda del sprite de la pelota. Haremos que la coordenada X y la coordenada Y de la pelota que definen su colocación en el escenario vayan variando cuando inclinemos el dispositivo en cualquiera de los dos ejes, o en los dos al mismo tiempo. Lo haremos simplemente sumando el valor de la variable **xAccel** al valor de la coordenada X actual de la pelota, y sumando **yAccel** al valor de la coordenada Y. Los bloques de color verde claro están dentro del cajón de propiedades de **spritePelota**.



Cuando hagamos esto veremos que el comportamiento vertical, el de la coordenada Y, será el esperado, es decir, que la pelota caerá hacia nosotros cuando inclinemos el dispositivo hacia nuestro lado, y se alejará de nosotros cuando inclinemos el móvil en la dirección contraria. Sin embargo, cuando inclinemos el dispositivo hacia la derecha, en el eje X, la pelota caerá hacia la izquierda, y viceversa. Para solucionar este problema, y adaptar el funcionamiento al comportamiento natural de una pelota, tendremos que modificar el bloque que indica el posicionamiento en la componente X de la coordenada. En lugar de sumarle el valor de **xAccel** le sumaremos el valor de restar **xAccel** a 1.



Ya está listo. Para la coordenada Y utilizaremos el bloque de sumar, y dependiendo de si **yAccel** tiene un valor positivo o negativo la pelota se moverá en una dirección u otra, porque sumar un número negativo es lo mismo que restar. En el caso de X tendremos que añadir un bloque de resta para corregir el comportamiento contrario, pero básicamente la lógica es la misma que con la Y.

¡Las matemáticas funcionan, y tienen aplicación en la vida real!

Veremos también que la velocidad a la que se mueve la pelota es mayor si inclinamos mucho el dispositivo. Esto es debido a que el valor de **xAccel** e **yAccel** que estamos sumándole a la posición de la pelota también es mayor o menor en función de cuánto inclinamos el dispositivo.

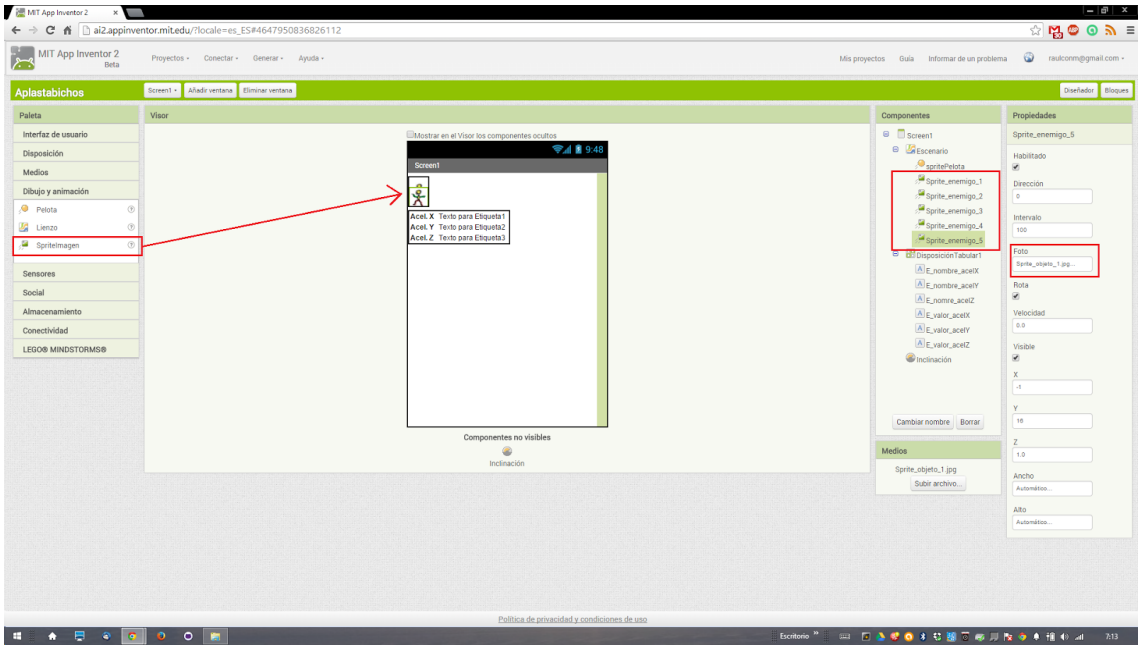
Tomémonos un tiempo para asimilar estos últimos párrafos porque no son sencillos de entender, y hagamos las pruebas que se nos ocurran, modificando el código que hemos generado para que el programa haga cualquier otra cosa que queramos.

Crear los bichos a aplastar

A continuación hay que crear cada objeto que vaya a aparecer en el escenario. Tendremos que almacenar la posición de todos y cada uno de ellos, ya que son objetos distintos entre sí, para que la aplicación sepa cuándo la pelota está en contacto con alguno, y el juego actúe en consecuencia.

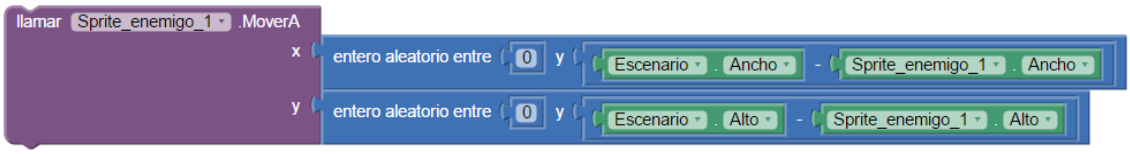
Para empezar habrá que dibujar el sprite que queremos utilizar en el juego para representar al bicho que queremos aplastar. En nuestro caso usaremos el pequeño sprite de 30x30 pixels “Sprite_objeto_1.jpg”, que nosotros mismos hemos creado, y que podemos encontrar en http://coderojo-medialabprado.4shared.com

A continuación, y desde la ventana del Diseñador, incluiremos en nuestro escenario cinco objetos **SpriteImagen** iguales. Le daremos un nombre diferente a cada uno, y a todos le asignaremos el aspecto de nuestro sprite. Podemos llamarles **Sprite_enemigo_1** a **Sprite_enemigo_5.**

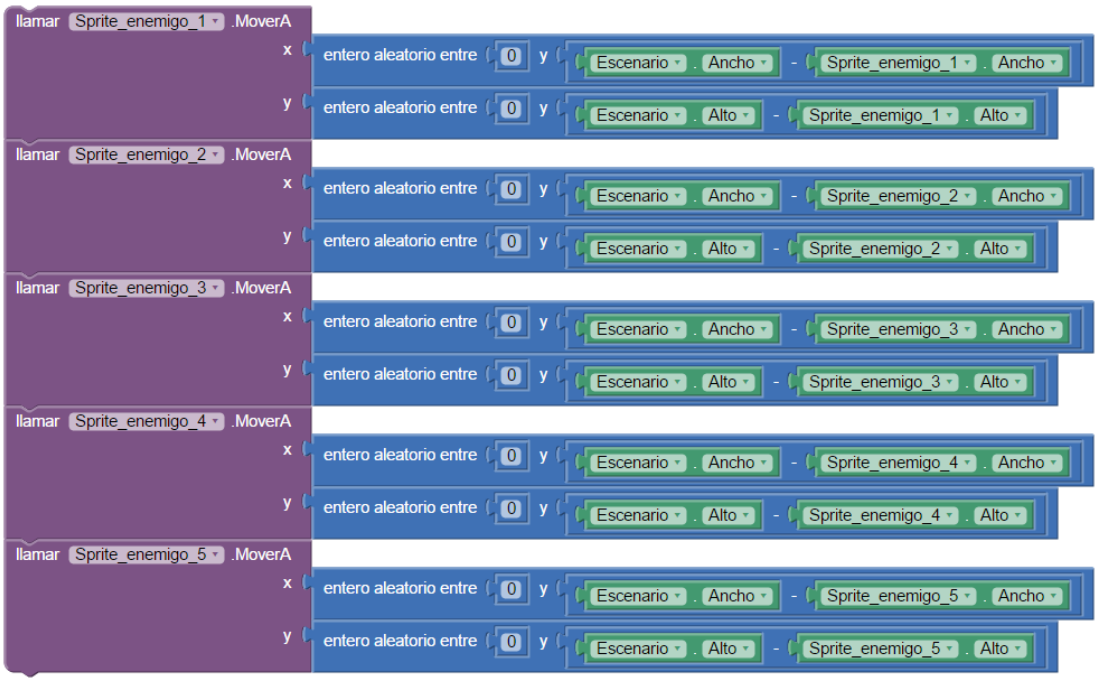


Manejo de los enemigos

Una vez creados los cinco objetos tenemos que colocarlos en el escenario. Lo haremos con el bloque **llamar.Sprite_enemigo_1.MoverA**. Especificaremos para cada objeto una posición aleatoria. Con los bloques azules **entero aleatorio entre** definiremos en qué coordenada X aparecerá la esquina superior del sprite dentro del escenario. Especificaremos un 0 para indicar que el objeto puede aparecer desde el margen izquierdo del escenario. A continuación le diremos con el bloque verde **Escenario.Ancho** que el límite máximo es el límite derecho del escenario. Pero, atención, si el bloque azul entero aleatorio entre nos devolviera precisamente un valor de X muy cercano al límite derecho la mayor parte del enemigo sobrepasaría el límite derecho del escenario, y no sería visible. Para solucionar esto se resta a **Escenario.Ancho** el ancho del sprite del enemigo, **Sprite_enemigo_1.Ancho**.



Haremos lo mismo con la coordenada vertical Y, y repetiremos los mismos bloques cinco veces, porque hay cinco objetos iguales.



Procedimiento para tareas definidas y repetitivas

Cada vez que el juego se reinicie tendremos que colocar los objetos en el escenario, y deberemos volver a usar todos estos bloques, así que lo mejor, para no tener que escribir el mismo código varias veces cuando es tan grande, será definir un procedimiento, donde incluiremos todos los bloques que sirven para colocar los objetos.

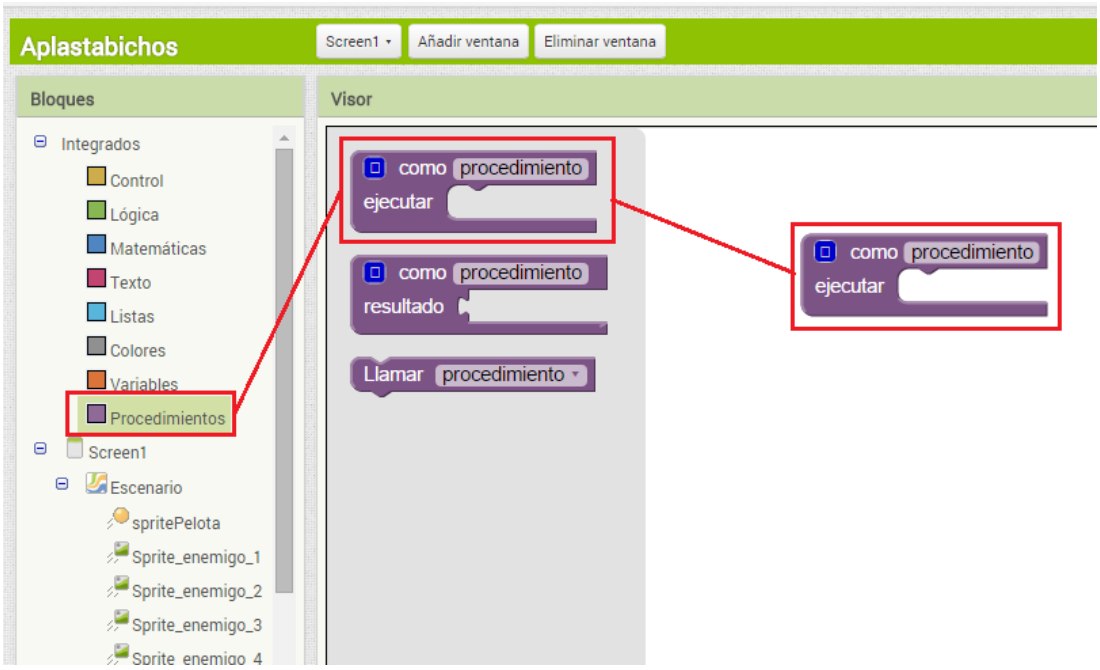
Un procedimiento es un conjunto de pasos bien definidos para ejecutar una tarea concreta que debe ser ejecutada muchas veces. Por ejemplo, para describir una tarea de nuestra vida diaria, como lavarnos las manos, podemos definir un procedimiento que incluya los siguientes pasos:

1. Abrir el grifo
2. Mojar las manos
3. Poner jabón
4. Frotar manos
5. Aclarar
6. Cerrar el grifo
7. Secar las manos con la toalla

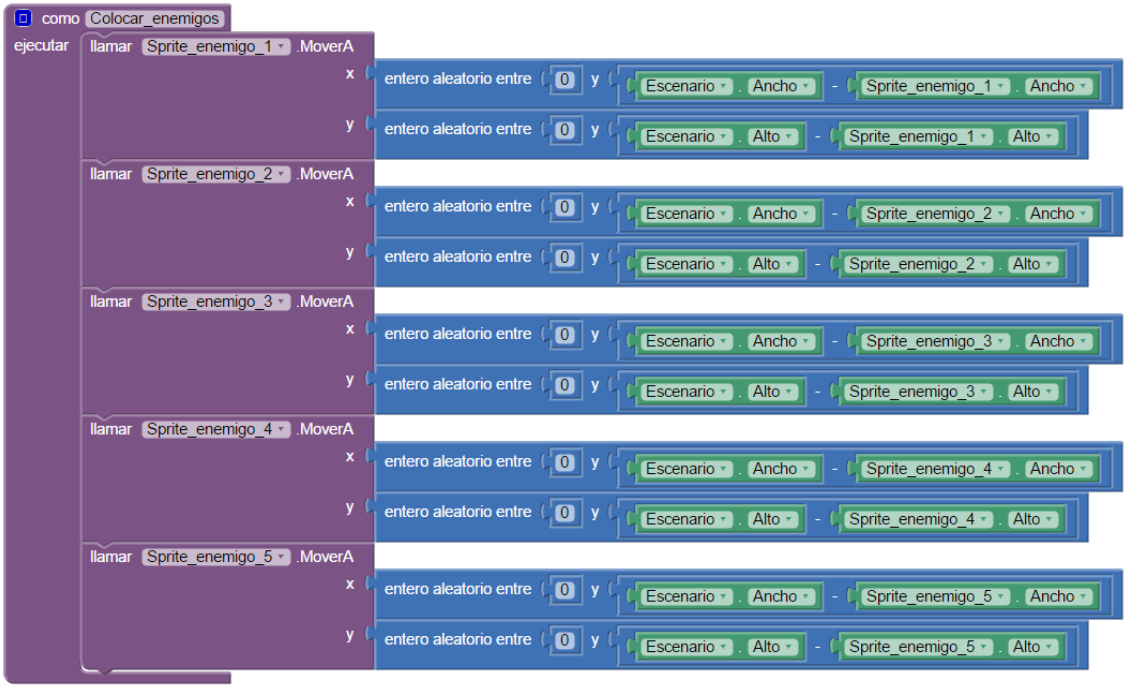
Estos pasos podrían incluirse dentro de un procedimiento llamado “Lavar manos”. Ya no tendríamos que enumerar cada uno de los siete pasos, sino referirnos al procedimiento por su nombre.

Los procedimientos son muy importantes, porque permiten organizar mejor el código y ahorrar esfuerzo a la hora de programar. En nuestro caso, podremos invocar en cualquier momento al procedimiento para hacer que la aplicación vuelva a colocar los objetos aleatoriamente en el escenario.

El bloque para definir procedimientos se encuentra dentro del cajón **Procedures**.



Una vez definido el procedimiento, y modificado su nombre, los bloques quedarán así.

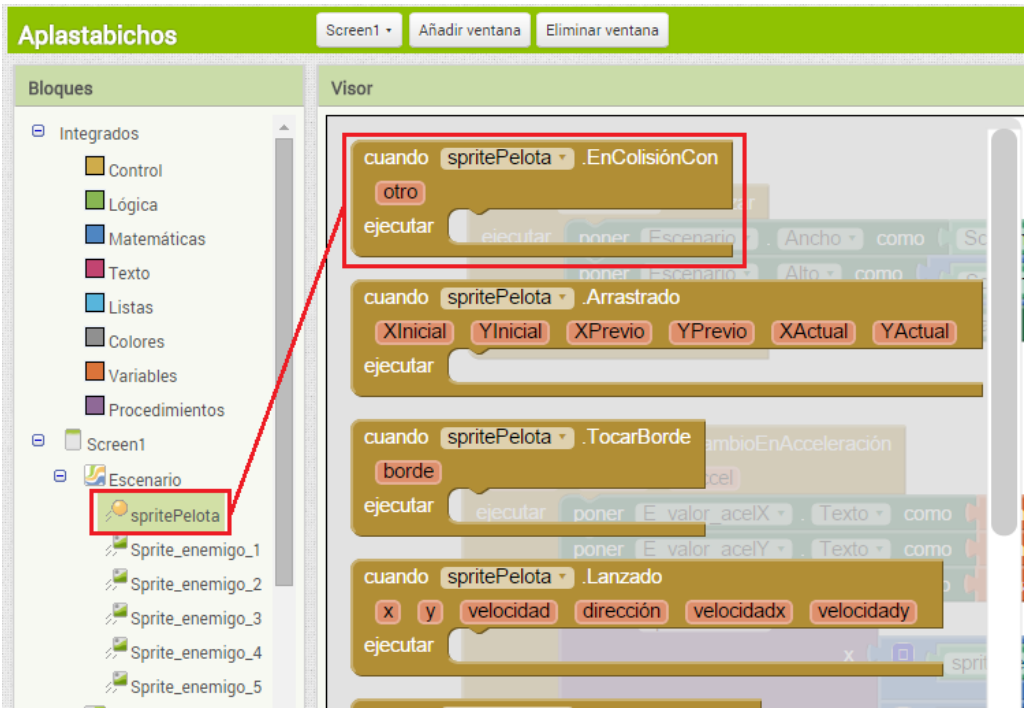


En este caso haremos que se ejecute el procedimiento dentro del bloque **cuando.Screen1.Inicializar ejecutar.**

Implementar la mecánica del juego

El jugador deberá inclinar el dispositivo para guiar a la pelota hacia cada uno de los objetos y chocar con ellos. Cada vez que eso suceda deberemos retirar el objeto del escenario. El juego acabará cuando el jugador haya hecho desaparecer todos los objetos.

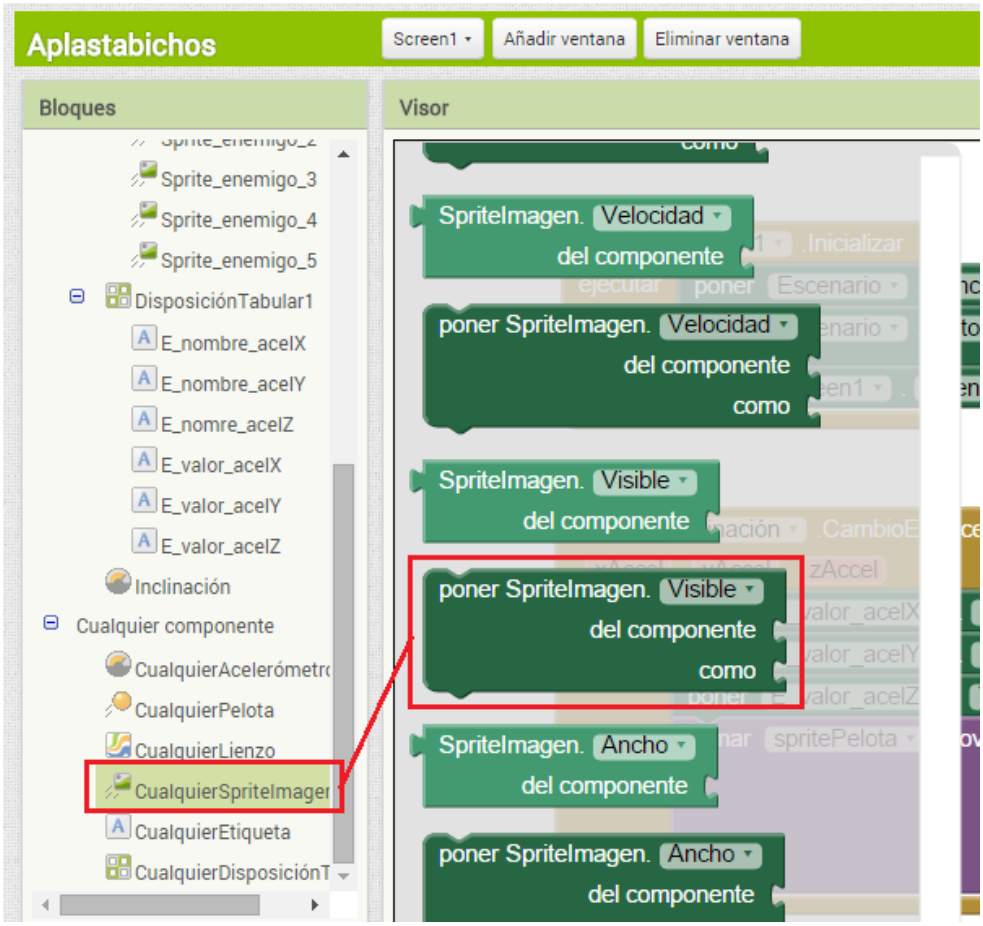
Para saber cuándo la pelota choca con un objeto utilizaremos el bloque **cuando spritePelota.EnColisiónCon otro ejecutar.** Este boque está dentro del cajón de recursos relacionados con el objeto **spritePelota**.



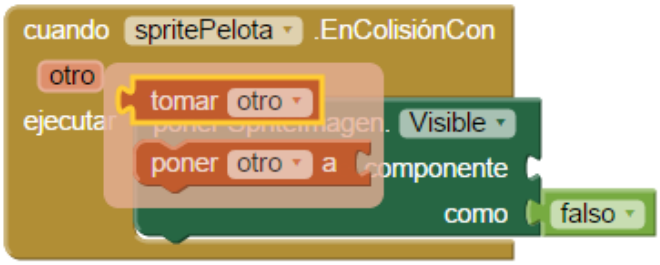
Lo que pongamos dentro de este bloque se ejecutará exactamente cuando eso suceda. En este caso, lo que queremos es que desaparezca el objeto contra el que ha chocado la pelota. Podremos hacer referencia a este objeto a través del parámetro **otro** incluido en el bloque **cuando spritePelota.EnColisiónCon otro ejecutar** que hemos definido.

Ahora vamos a utilizar un nuevo tipo de recurso, un bloque que no habíamos usado todavía. Este bloque se encuentra dentro del cajón **Cualquier componente / CualquierSpriteImagen**. La diferencia entre este cajón y los que hemos abierto antes es que los bloques contenidos aquí nos permitirán definir acciones que harán referencia a diferentes objetos del mismo tipo. En este caso hará referencia a objetos del tipo **SpriteImagen**, como son nuestros cinco “enemigos”.

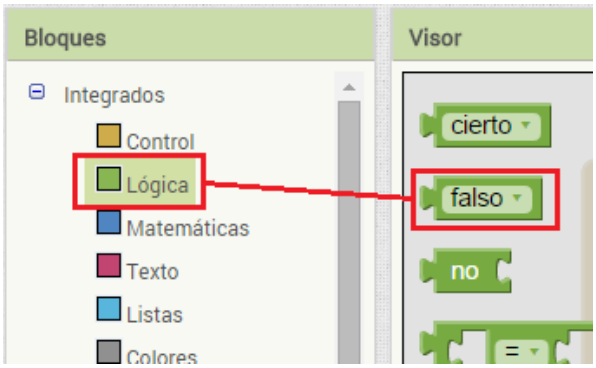
Lo que vamos a hacer es indicarle al programa que debe hacer desaparecer de la pantalla cada objeto cuando la pelota choca con él. Para no tener que repetir el código para cada objeto, usaremos el bloque genérico **poner SpriteImagen.Visible del componente como.** Este bloque nos permitirá cambian el valor de la propiedad **Visible** de cualquier componente del tipo **SpriteImagen**, que especificaremos en el hueco **del componente**.



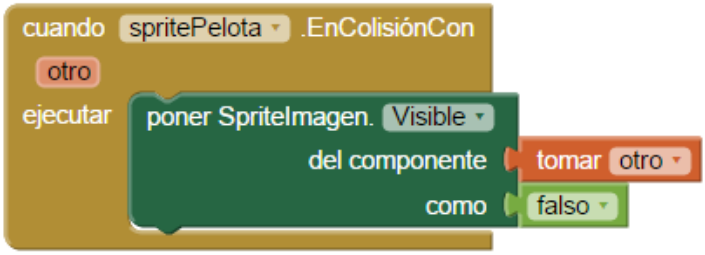
Usaremos el parámetro **otro** del bloque mostaza que hemos añadido antes para indicar a qué objeto concreto queremos referirnos, y estableceremos a **falso** el valor de la propiedad **Visible** del objeto, para que desaparezca del escenario.



Muchas propiedades de los objetos tienen dos posibles valores, **cierto** o **falso**. Podemos definir el valor para este tipo de variables tomándolo del cajón **Lógica**.

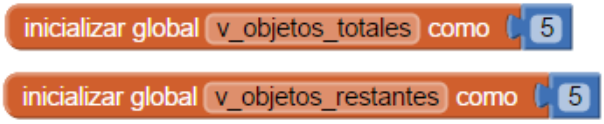


El bloque completo que define el comportamiento de cada objeto cuando la pelota choca con él quedará como en la siguiente figura.

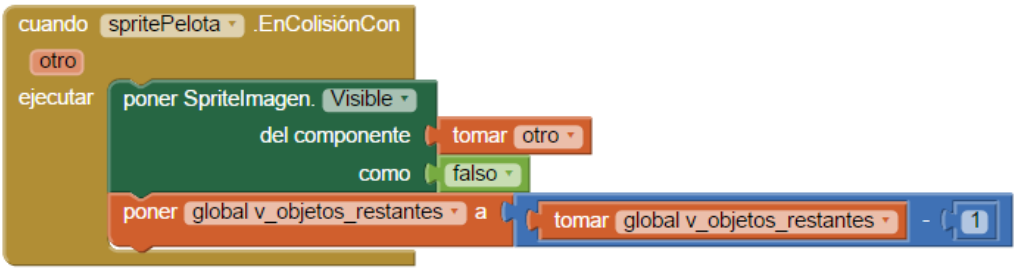


Gestión del marcador

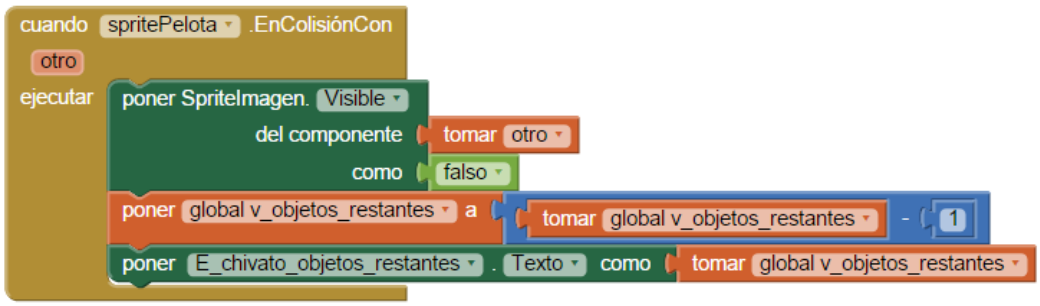
Tenemos también que mantener un contador de objetos restantes, para saber cuándo se han recogido todos y dar por completada la misión. Para ello usaremos dos variables, **v_objetos_totales**, y **v_objetos_restantes.** La primera define el número de objetos que vamos a manejar en cada partida, y la segunda define el número de objetos que quedan por hacer desaparecer. Como siempre, par indicar que estamos definiendo una variable, y que luego sea más fácil identificarla como tal, comenzaremos los nombres con el prefijo **v_**.



Cada vez que la pelota choque con un objeto, tendremos que restar **1** al número de objetos restantes. Cuando la variable **v_objetos_restantes** sea **0**, significará que el jugador ha recogido todos los objetos, y el juego habrá terminado.



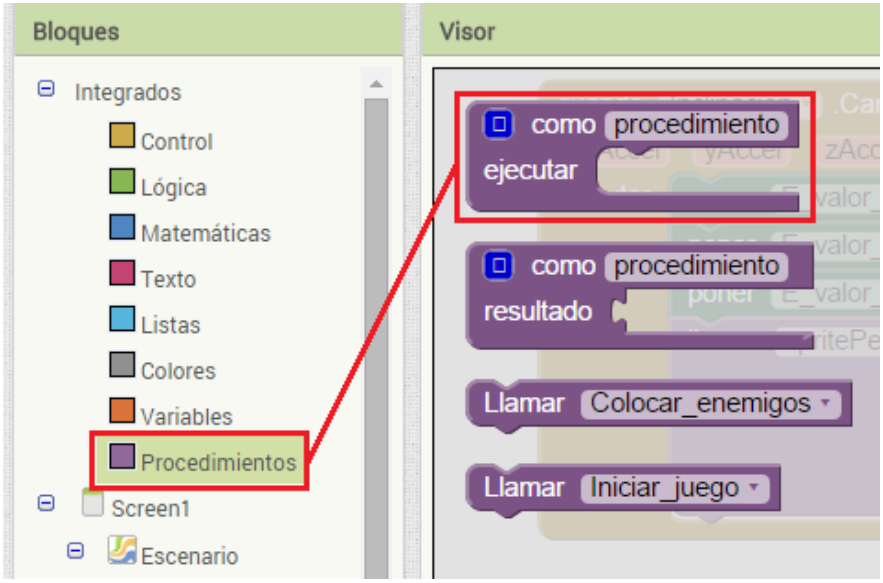
Podemos añadir un chivato debajo del escenario para saber cuál es el contenido de la variable **v_objetos_restantes**. Así sabremos si nuestro juego está gestionando correctamente esta variable tan importante.



Tenemos que acordarnos de eliminar cualquier chivato que hayamos utilizado antes de hacer la versión final del programa.

Reorganizando los bloques de código

Cuando una aplicación está empezando a crecer, como la nuestra, conviene mirarla un poco "desde lejos" y pensar de qué manera podemos hacer que sea más fácil de manejar, cómo organizarla mejor. Así pues, para facilitar el mantenimiento de nuestro programa, y su lectura, vamos a definir dos procedimientos: **Iniciar_juego** y **Fin_del_juego**. Para ellos abriremos el cajón **Procedimiento** y arrastraremos dos veces el bloque como **procedimiento ejecutar**.



Nombraremos los nuevos bloques quedarán de la siguiente manera.

En el primero de ellos tendremos que reubicar los objetos, y reiniciar todas las variables que tiene que manejar el juego. Es decir, dejar las cosas listas para empezar a jugar.

Vemos que el procedimiento **Fin_del_juego** aún está vacío, pero ya sabemos que incluiremos en él los bloques que deberán ejecutarse cuando el juego termine.

Habrá que hacer una llamada al procedimiento **Iniciar_juego** dentro del bloquec **cuando Screen1.Inicializar ejecutar**, que ahora quedará así.

Limitar el tiempo para crear tensión

Para dar mayor interés al juego usaremos un temporizador, un componente **Reloj**, que limitará el tiempo que tiene el jugador para aplastar a los enemigos. Cuando el tiempo termine, el jugador no podrá eliminar más objetos.

Tenemos que empezar por crear un objeto **Reloj** en el Diseñador. El componente **Reloj** se encuentra dentro del cajón **Sensores**. Le daremos el nombre **Cada_Segundo**. Se pretende conocer cuándo pasa cada segundo, así que su propiedad **IntervaloDelTemporizador** contendrá el valor 1000 (1000 milisegundos es igual a 1 segundo).

La idea es mostrar siempre en la pantalla el número de segundos restantes, y para eso necesitamos restar un segundo cada vez respecto de la cantidad que queda disponible. Así pues, definiremos el número de segundos disponibles con la variable **v_segundos_restantes**, y le daremos el valor inicial 30.

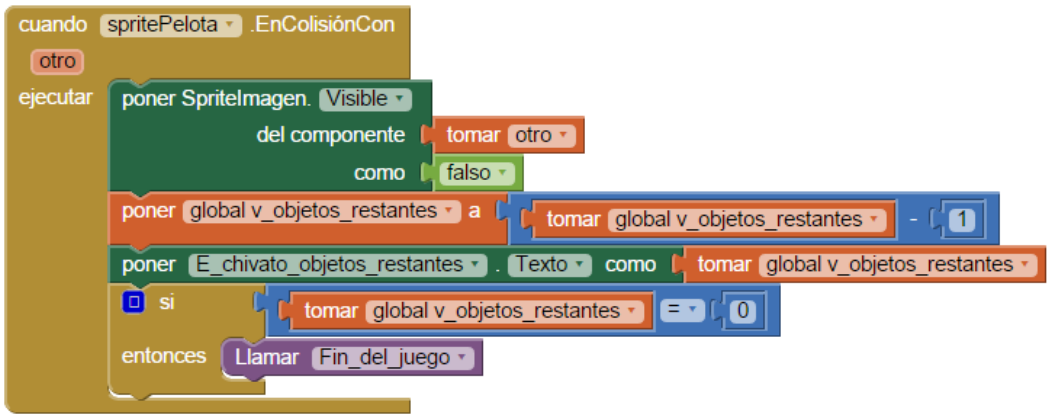
En el bloque **cuando.Cada_segundo.Temporizador ejecutar** indicaremos que reste **1** a la variable **v_segundos_restantes**, y que lo muestre en un nuevo objeto de tipo **Etiqueta** que llamaremos **E_segundos_restantes**.

Con estos componentes que hemos definido controlaremos el tiempo de juego.

¿Cuándo termina el juego?

Cuando esta variable **v_segundos_restantes** alcance el valor 0 (cero) la partida habrá terminado, porque ya no habrá segundos restantes. Podemos implementar esta comprobación añadiendo un bloque mostaza **si-entonces** dentro del bloque **cuando.Cada_segundo.Temporizador ejecutar**. Cuando se cumpla la condición del **si-entonces** habrá llegado el momento de hacer una llamada al procedimiento **Fin_del_juego**.

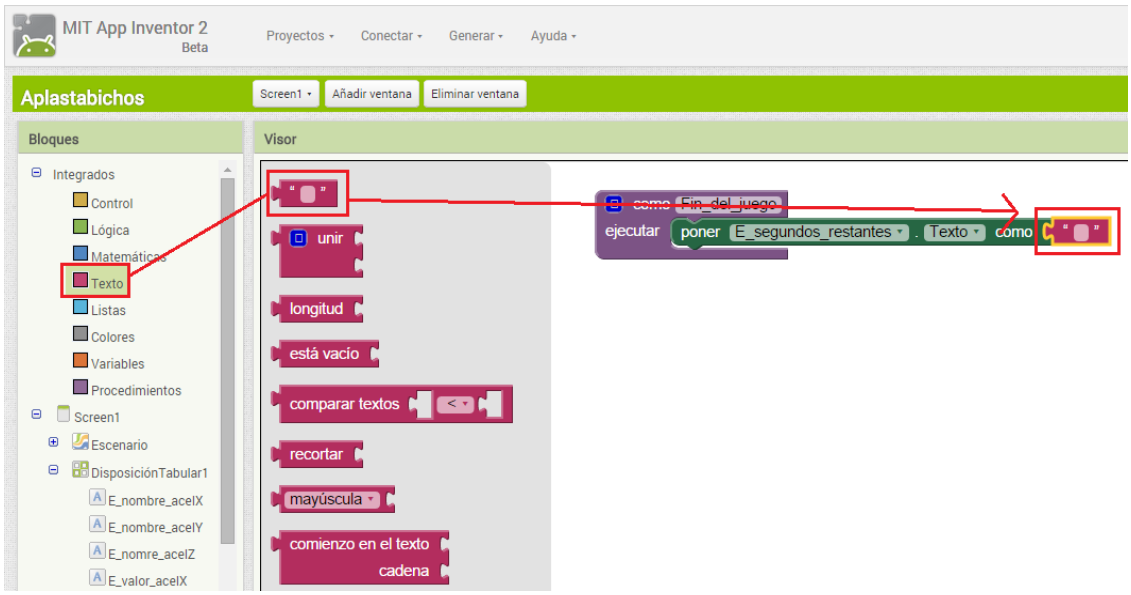
¿En qué otro caso deberá considerarse el juego terminado? Claro, el juego también deberá terminar cuando no queden más objetos por recoger. Lo indicaremos con un bloque **si-entonces** dentro del bloque **cuando.spritePelota.EnColisiónCon.ejecutar.**



Fin del juego

Tenemos entonces que definir qué hacer cuando el juego termine. Una opción sencilla y clara puede ser informar al usuario a través de un texto, y tal vez un sonido.

Por ejemplo, podemos escribir el texto "Fin del juego" en la etiqueta **E_segundos_restantes**. Para ello abriremos el cajón **Texto** del editor de bloques, y arrastraremos el componente de texto vacío al hueco verde que asigna un valor al texto de la etiqueta **E_segundos_restantes**.



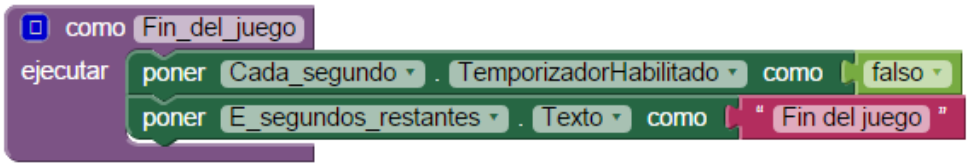
Haciendo clic entre las comillas podremos definir el texto que se mostrará.

Extrañamente, ese texto no aparece por mucho tiempo en la pantalla. ¿Por qué?

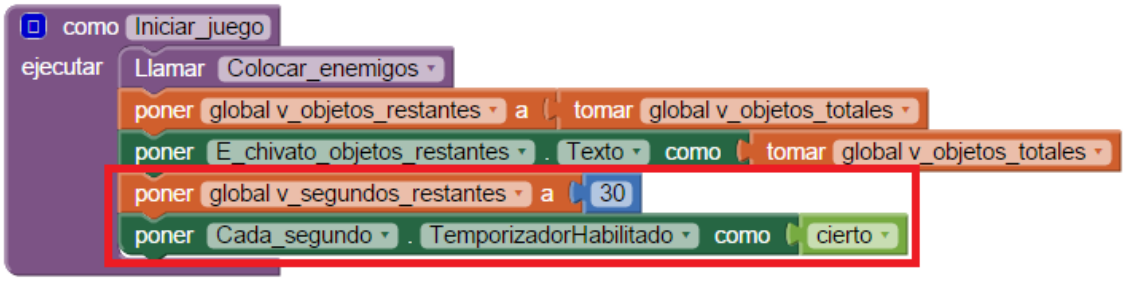
Si ponemos mucha atención veremos que el texto aparece, pero sólo durante un segundo. Esto se debe a que el programa comprueba en el bloque **cuando.Cada_segundo.Temporizador ejecutar** si la variable **Segundos_totales** vale 0, pero esto sólo se cumple durante un segundo, porque el tiempo sigue contando, así que rápidamente el valor de **v_segundos_restantes** pasa a ser **-1**, que es el valor que se obtiene cuando a 0 le restamos 1. El programa sigue ejecutándose, y en la siguiente ejecución del bloque **cuando.Cada_segundo.Temporizador ejecutar** volverá a escribir el valor del contador de segundos encima del texto.

Congelando el tiempo

Para evitar esto tenemos que hacer que **Cada_segundo** deje de contar, es decir, que el bloque **cuando.Cada_segundo.Temporizador ejecutar** deje de ejecutarse. Esto se consigue dándole el valor **falso** a la propiedad **TemporizadorHabilitado** del componente **Cada_segundo**.

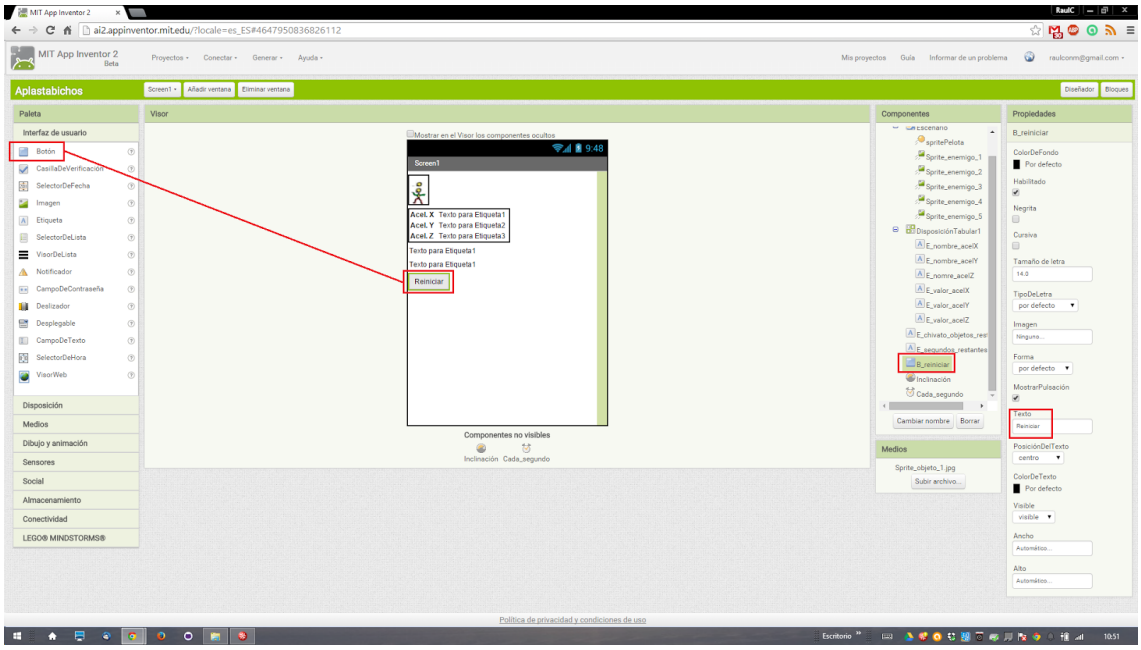


Pero atención, si queremos que el programa funcione correctamente la próxima vez, y ejecute el contenido de **cuando.Cada_segundo.Temporizador ejecutar**, tendremos que poner el valor de esa propiedad como **cierto** en el procedimiento **Iniciar_juego**. Además antes deberemos restablecer el número de segundos restantes a **30**, su valor inicial. En caso de olvidarnos de dar a nuestro programa alguna de estas dos instrucciones, el juego se ejecutaría para siempre.

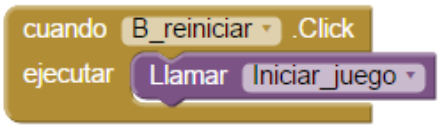


Añadir un botón para empezar de nuevo

Bueno, casi está todo, pero no podemos olvidar incluir un botón “Reiniciar” para que el juego comience de nuevo una vez terminado.



No será difícil crear el código necesario para poner el juego de nuevo a funcionar, porque hemos sido organizados, y hemos creado los procedimientos adecuados. Sólo tendremos que indicarle al programa que ejecute el procedimiento **Iniciar_juego** cuando el jugador pulse el botón **Reiniciar**.

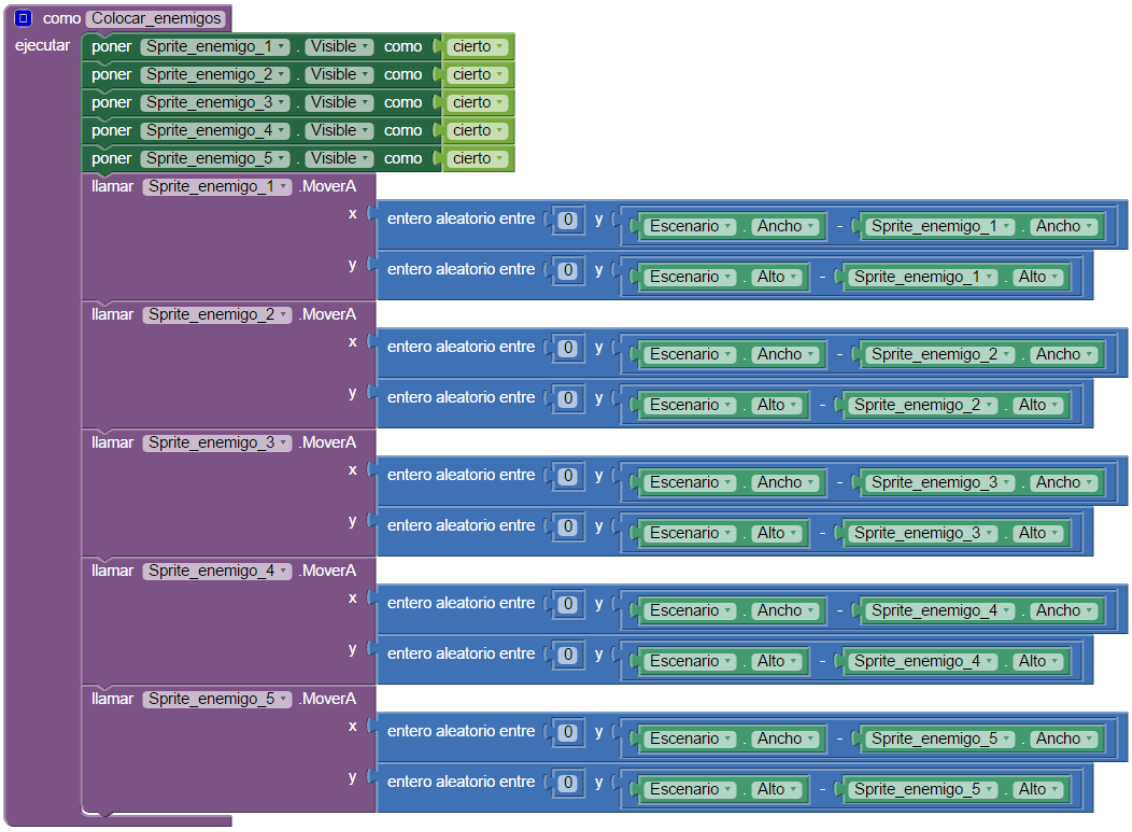


Un último detalle

Veremos que cuando iniciamos de nuevo el juego ya no aparecen los objetivos que hay que "aplastar", a pesar de que sí vuelve a tener el valor **5** la variable **v_objetos_restantes**. ¿Por qué? Por favor, revisa el programa intenta averiguarlo antes de ver la explicación.

Bueno, lo que ocurre es hemos olvidado algo. En el procedimiento **cuando.spritePelota.EnColisiónCon** hacemos desaparecer cada objeto cuando la pelota choca con él, usando el bloque **poner SpriteImagen.Visible del componente como.** Para que los objetos vuelvan a ser visibles al pulsar el botón **Reiniciar** tendremos que poner este atributo como **cierto** de nuevo. Podemos incluir esta instrucción, por ejemplo, dentro el procedimiento **Colocar_enemigos**.

Vamos a hacer esto utilizando un bloque disponible dentro de los cajones de cada enemigo. Usaremos el bloque verde oscuro que sirve para establecer el estado del atributo **Visible**. Podemos hacer lo mismo para todos los enemigos copiando el bloque del primero y cambiando el nombre del enemigo al que se refiere en cada caso.



Ideas para mejorar el juego

A partir de este punto cada uno puede mejorar el juego como mejor le parezca, pensando qué quiere que suceda en cada momento, e implementándolo en el programa. Se proponen algunas ideas:

- Agregar sonido al movimiento de la pelota
- Añadir una imagen de fondo más profesional
- Incluir un sonido cada vez que se aplaste un enemigo
- Añadir sonido al paso de los segundos

¡Y atención, no olvidemos quitar los chivatos antes terminar el juego!

Comentarios finales

Para un creador de aplicaciones, el ordenador es una herramienta, un recurso, que le permite desarrollar sus ideas, hacerlas realidad, para que otros las utilicen. Para conseguirlo tan solo se necesita aprender uno de los lenguajes que el ordenador entiende. Si practicas con continuidad, antes de lo que imaginas podrás desarrollar programas que hagan casi todo aquello que se te ocurra.

Recuerda que solo tu imaginación y creatividad son el límite.

Raúl C. (@raulconm)
raulconm@gmail.com