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.

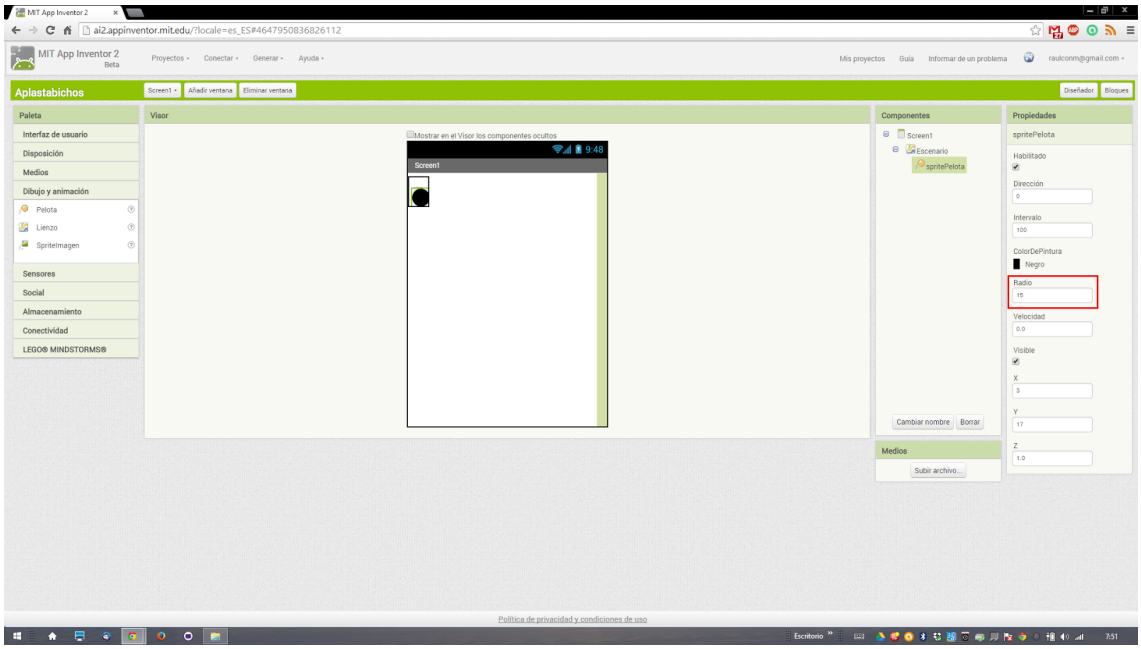
Este tipo de componentes deben "existir" siempre dentro de un componente **Lienzo**, que define el escenario donde se desenvuelve el sprite o sprites. Es decir, primero debemos definir un **Lienzo**, y a continuación incluiremos nuestro sprite dentro de él.

Definición del escenario de juego

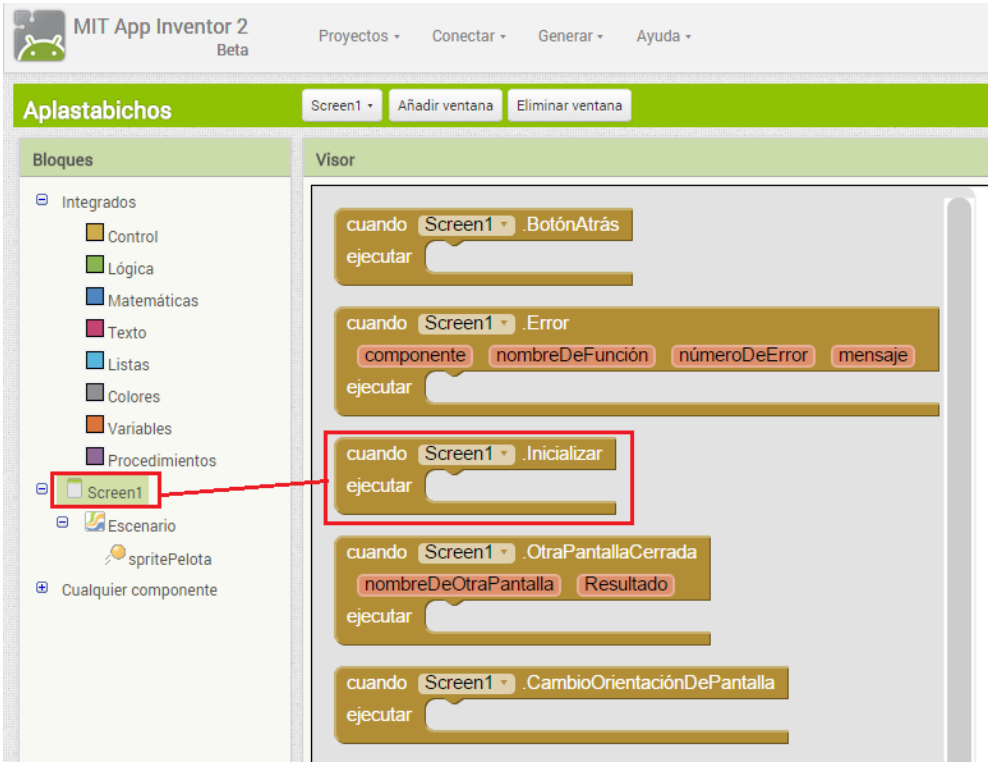
Vamos a programar un nuevo juego un poco más sofisticado, para seguir aprendiendo conceptos y componentes. Se trata de conducir una pelota a través de un escenario de juego para ir aplastando enemigos. Habrá que gestionar el movimiento de la pelota, la ubicación de los enemigos, el marcador, la duración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

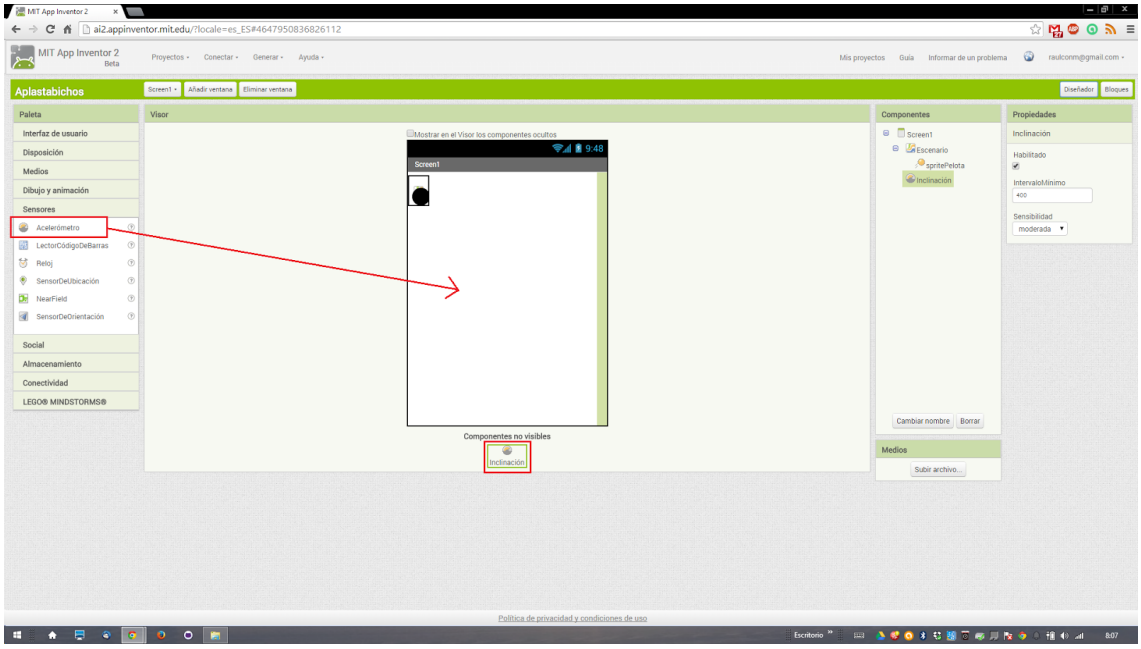
Definir el objetivo del juego

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.



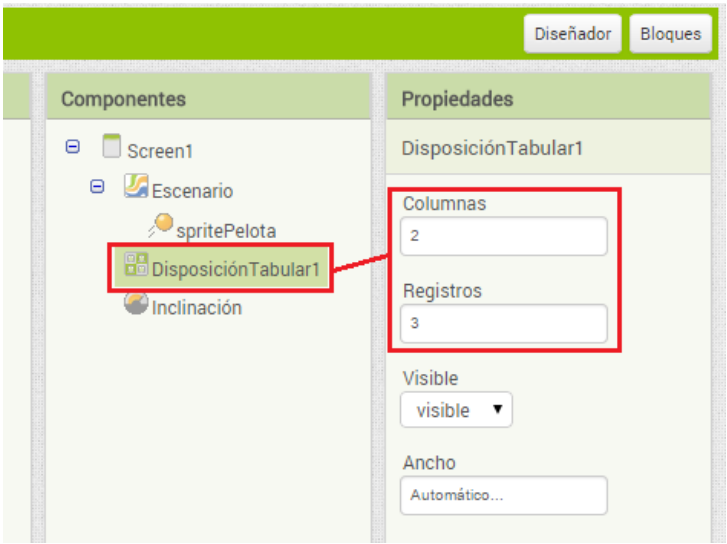
Ahora utilizaremos el bloque **cuando.Inclinación.CambioEnAceleración** para mover la pelota. Este bloque incluye tres variables, cada una de las cuales almacena la inclinación del objeto en uno de los ejes de coordenadas. Vamos a hacer que la pelota tenga una dirección dependiendo de los valores de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

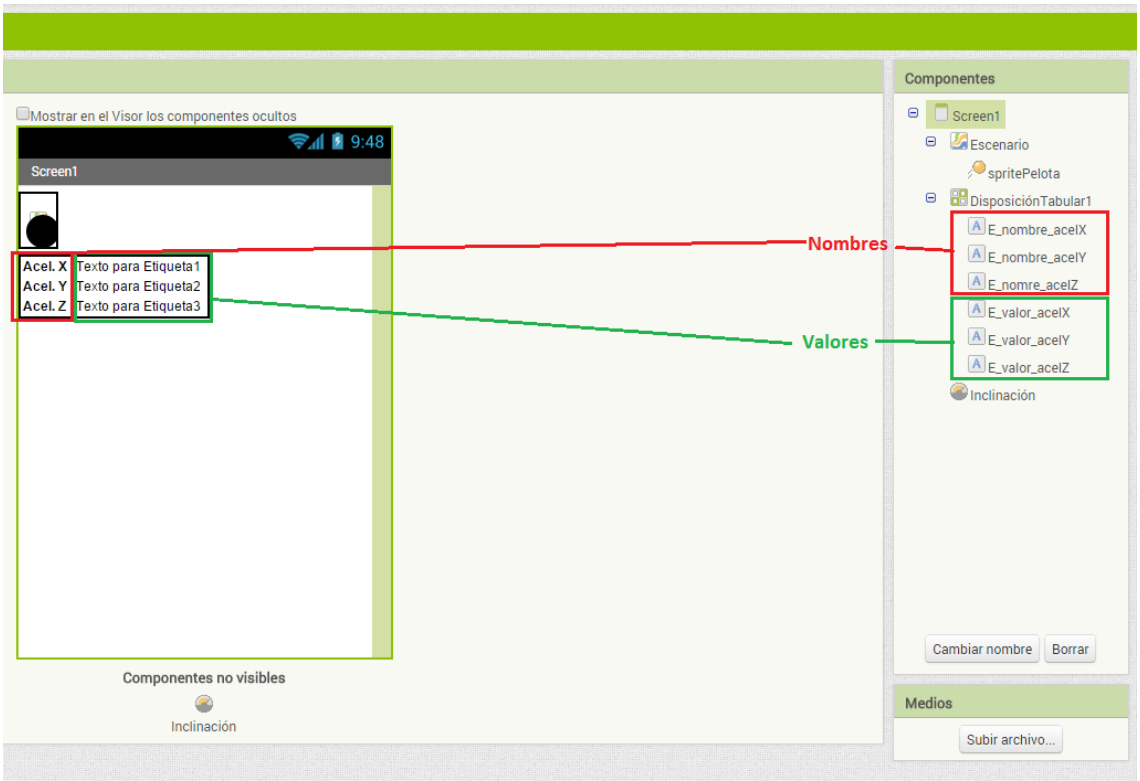
Para ello reduciremos un poco el tamaño del escenario, y colocaremos debajo tres campos **Etiqueta** para mostrar el valor de las variables.

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**.

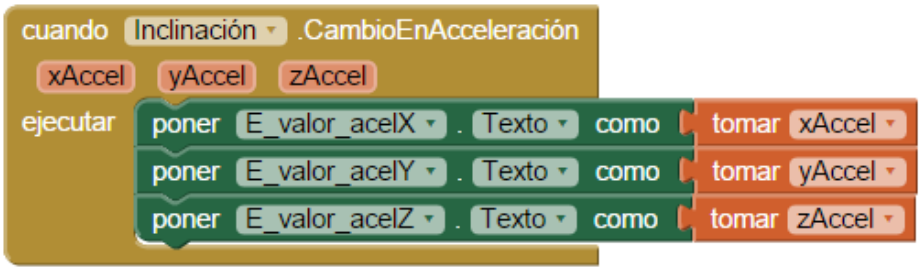


Dentro de la tabla, en la columna de izquierda, colocaremos tres etiquetas con la descripción de cada campo, **E_nombre_acelX**, **E_nombre_acelY** y **E_nombre_acelZ**. En la columna de la derecha pondremos tres etiquetas **E_valor_acelX**, **E_valor_acelY** y **E_valor_acelZ**, para mostrar las variables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



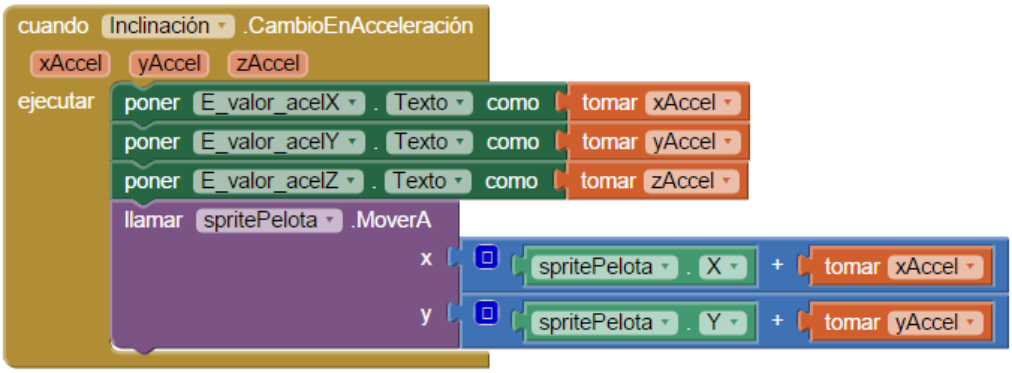
Es buen momento para experimentar qué sucede con cada una de las variables cuando inclinamos el dispositivo. Podemos invertir un poco de tiempo probando, hasta que entendamos cómo afecta la inclinación a cada una de estas variables. Veremos que para la Y los valores comprenderán de -10 (cuando el dispositivo 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).

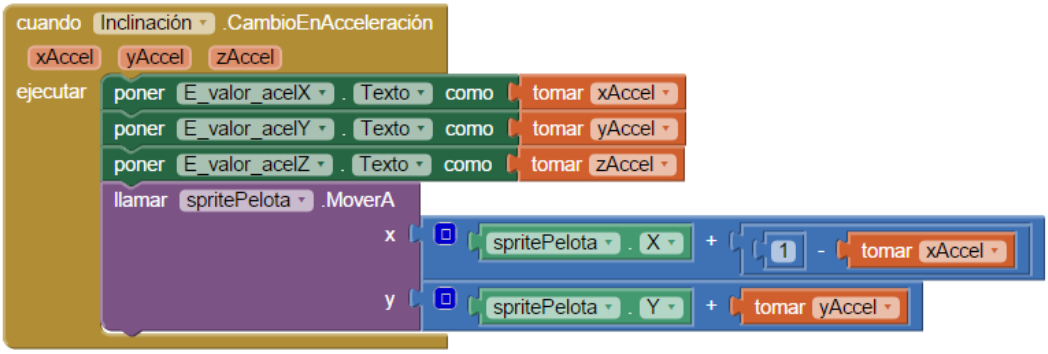
En cuanto a la Z, inicialmente sin uso en este juego, irá de 10 (cuando el dispositivo tiene la pantalla hacia arriba) a -10 (cuando el dispositivo tenga la pantalla hacia abajo, paralela al suelo).

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.

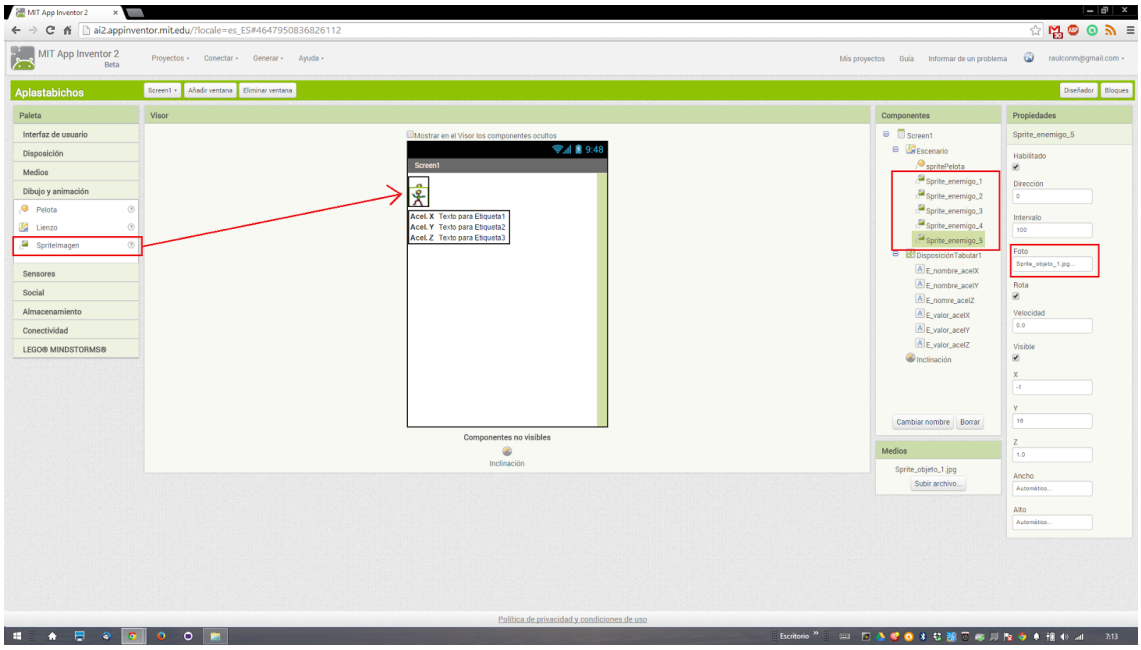
Tomémonos un tiempo para asimilar estos últimos párrafos porque no son sencillos de entender, y hagamos las pruebas que se nos ocurran, modificando el código que hemos generado para que el programa haga cualquier otra cosa que queramos.

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.

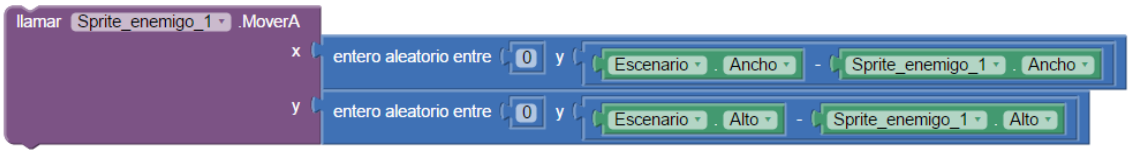
Para empezar habrá que dibujar el sprite que queremos utilizar en el juego para representar al bicho que queremos aplastar. En nuestro caso usaremos el pequeño sprite de 30x30 pixels “Sprite_objeto_1.jpg”, que nosotros mismos hemos creado, y que podemos encontrar en http://coderojo-medialabprado.4shared.com

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.**

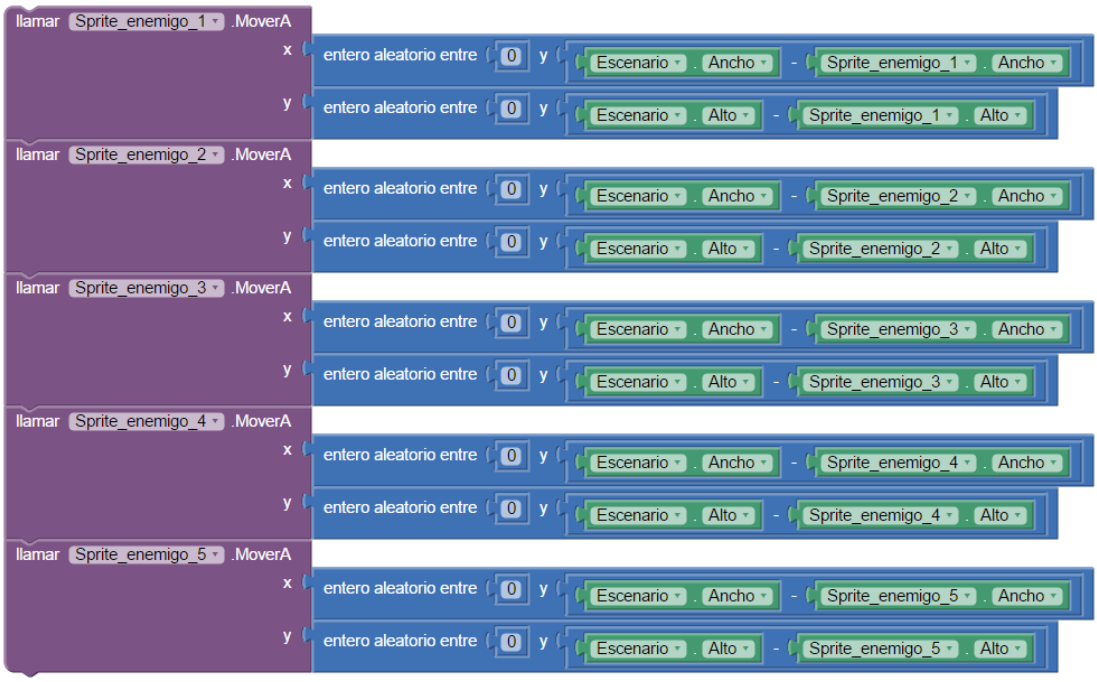


Manejo de los enemigos

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.

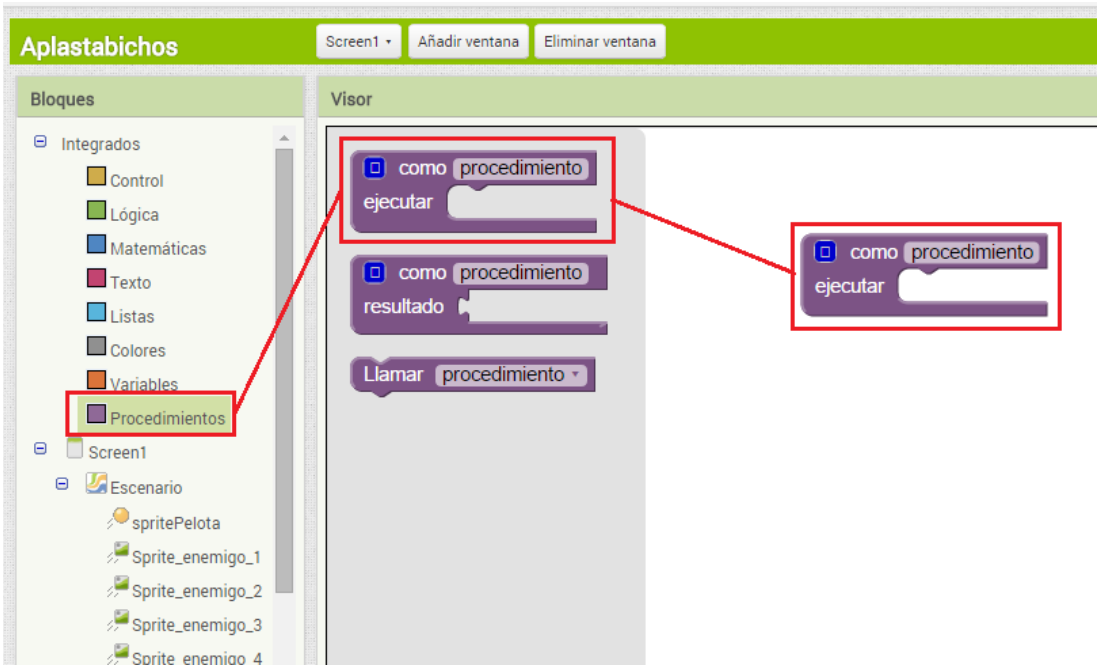
Un procedimiento es un conjunto de pasos bien definidos para ejecutar una tarea concreta que debe ser ejecutada muchas veces. Por ejemplo, para describir una tarea de nuestra vida diaria, como lavarnos las manos, podemos definir un procedimiento que incluya los siguientes pasos:

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

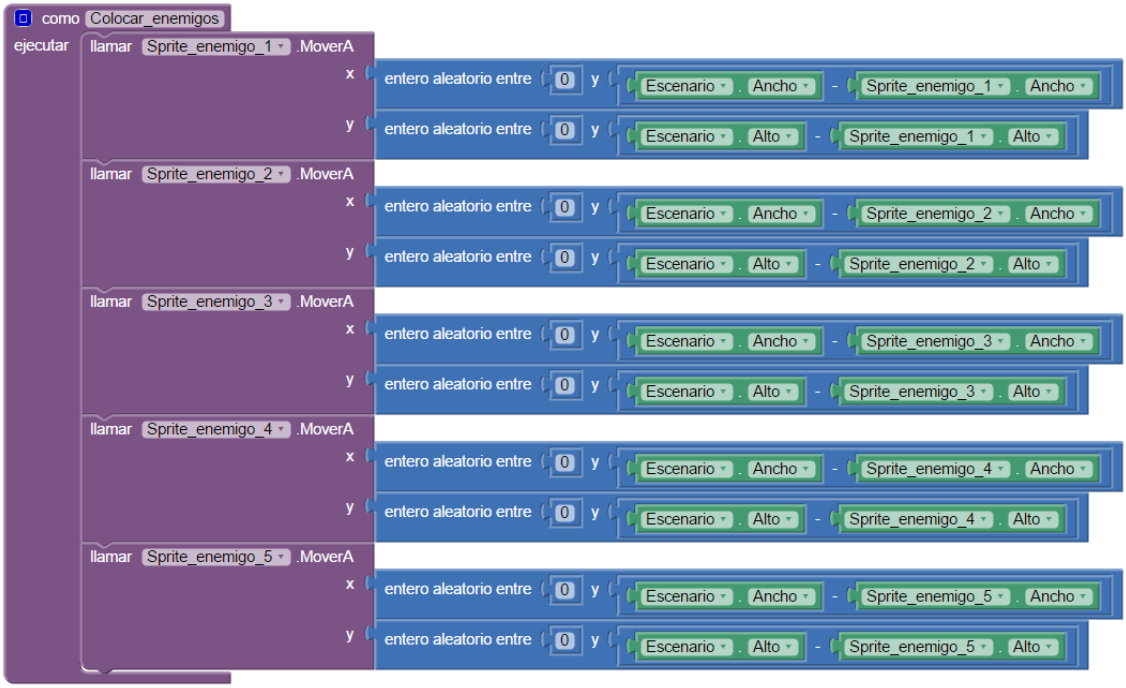
Estos pasos podrían incluirse dentro de un procedimiento llamado “Lavar manos”. Ya no tendríamos que enumerar cada uno de los siete pasos, sino referirnos al procedimiento por su nombre.

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.

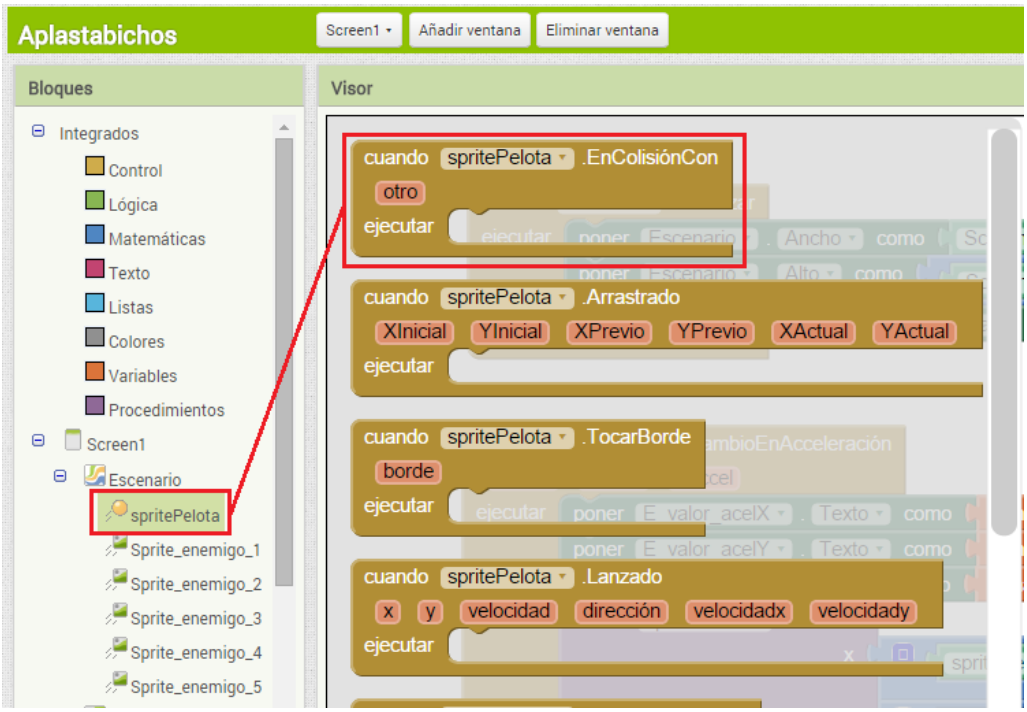


En este caso haremos que se ejecute el procedimiento dentro del bloque **cuando.Screen1.Inicializar ejecutar.**

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.

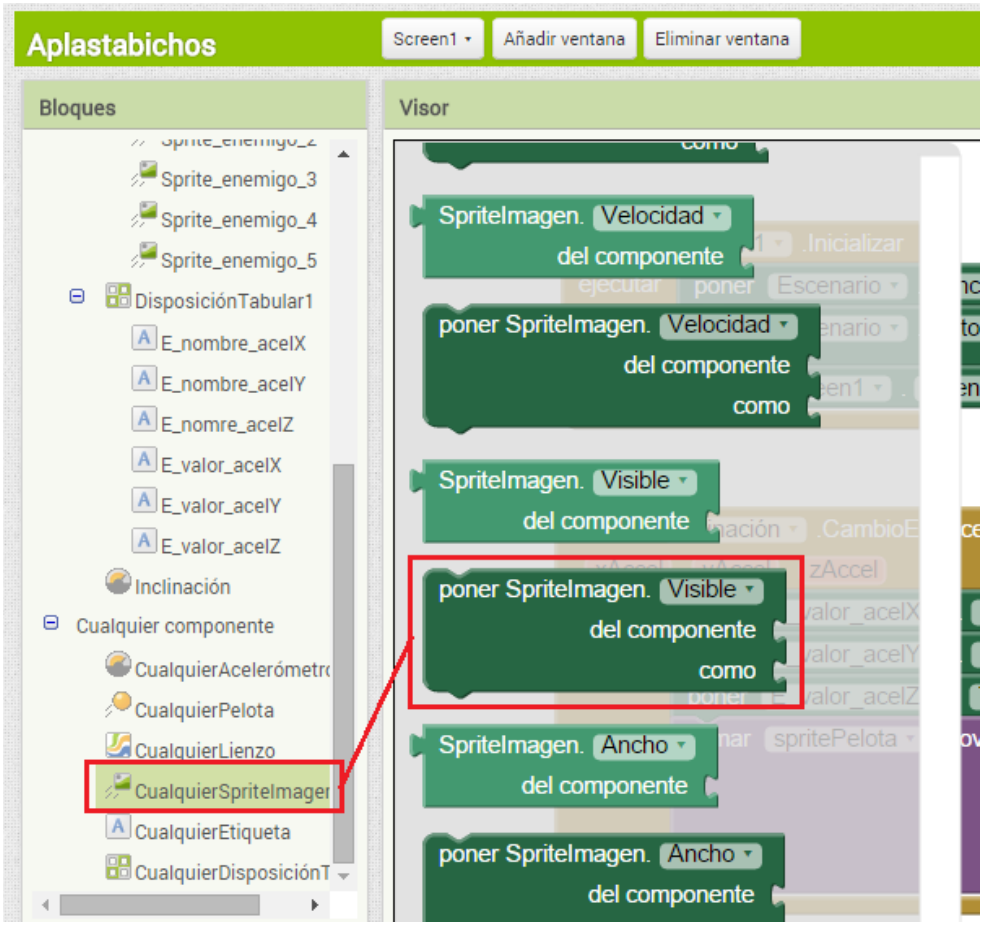
Para saber cuándo la pelota choca con un objeto utilizaremos el bloque **cuando spritePelota.EnColisiónCon otro ejecutar.** Este boque está dentro del cajón de recursos relacionados con el objeto **spritePelota**.



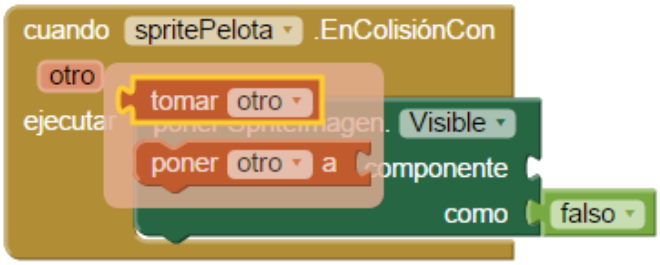
Lo que pongamos dentro de este bloque se ejecutará exactamente cuando eso suceda. En este caso, lo que queremos es que desaparezca el objeto contra el que ha chocado la pelota. Podremos hacer referencia a este objeto a través del parámetro **otro** incluido en el bloque **cuando spritePelota.EnColisiónCon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

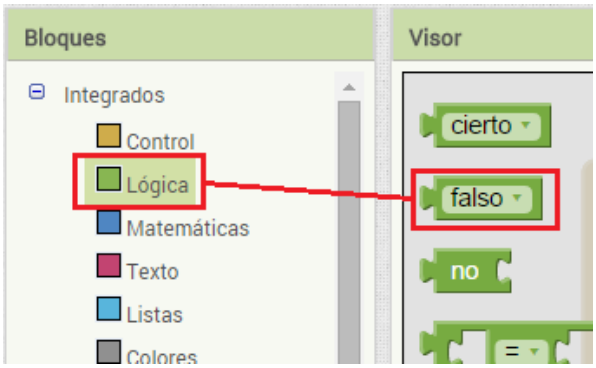
Lo que vamos a hacer es indicarle al programa que debe hacer desaparecer de la pantalla cada objeto cuando la pelota choca con él. Para no tener que repetir el código para cada objeto, usaremos el bloque genérico **poner SpriteImagen.Visible del componente como.** Este bloque nos permitirá cambian el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



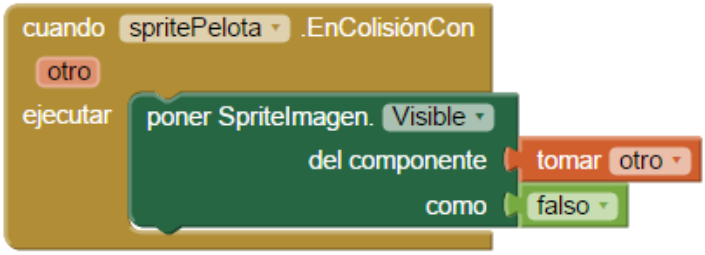
Usaremos el parámetro **otro** del bloque mostaza que hemos añadido antes para indicar a qué objeto concreto queremos referirnos, y estableceremos a **falso** el valor de la propiedad **Visible** del objeto, para que desaparezca del escenario.



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**.

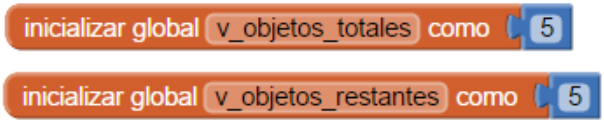


El bloque completo que define el comportamiento de cada objeto cuando la pelota choca con él quedará como en la siguiente figura.

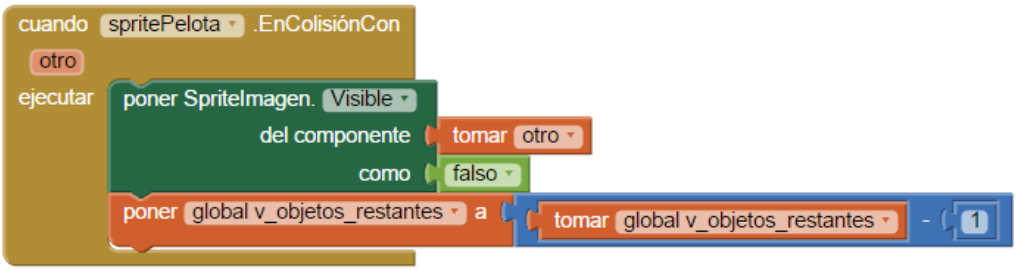


Gestión del marcador

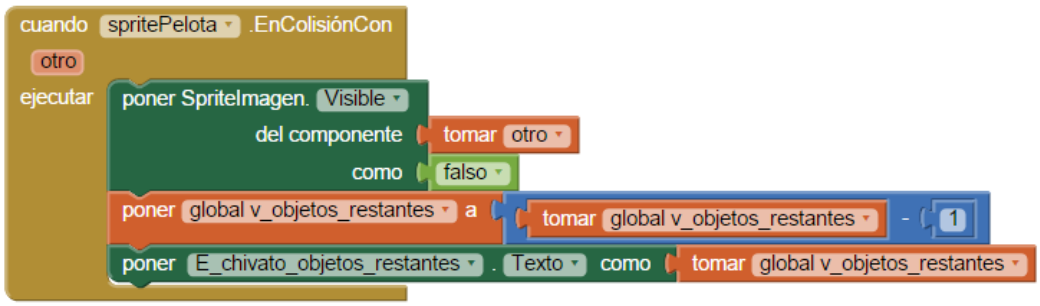
Tenemos también que mantener un contador de objetos restantes, para saber cuándo se han recogido todos y dar por completada la misión. Para ello usaremos dos variables, **v_objetos_totales**, y **v_objetos_restantes.** La primera define el número de objetos que vamos a manejar en cada partida, y la segunda 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.



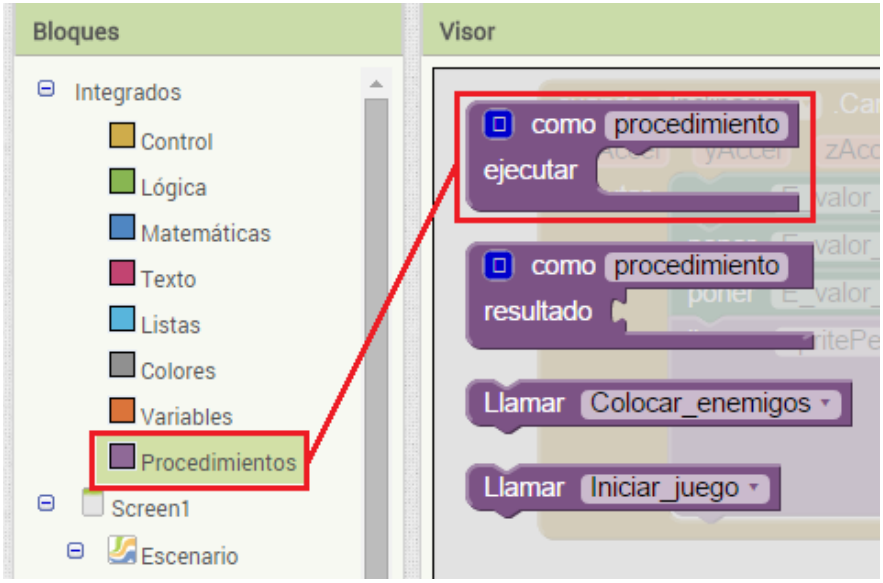
Podemos añadir un chivato debajo del escenario para saber cuál es el contenido de la variable **v_objetos_restantes**. Así sabremos si nuestro juego está gestionando correctamente esta variable tan importante.



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.

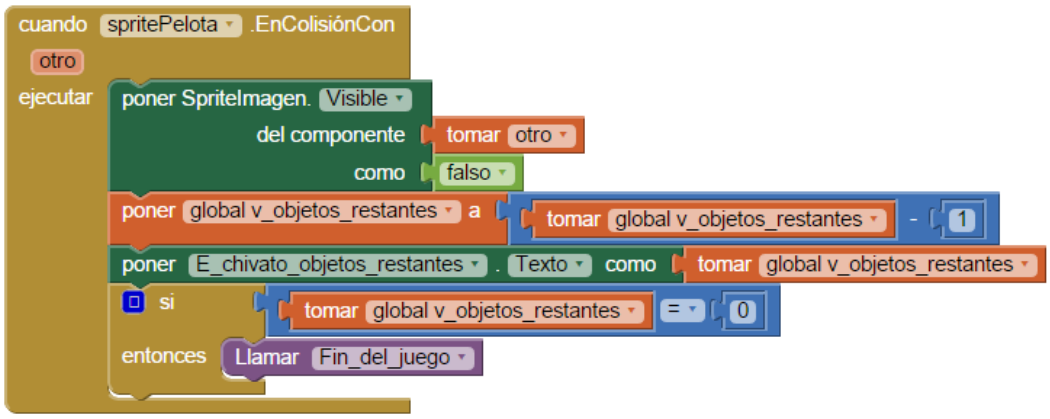
En el bloque **cuando.Cada_segundo.Temporizador ejecutar** indicaremos que reste **1** a la variable **v_segundos_restantes**, y que lo muestre en un nuevo objeto de tipo **Etiqueta** que llamaremos **E_segundos_restantes**.

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**.

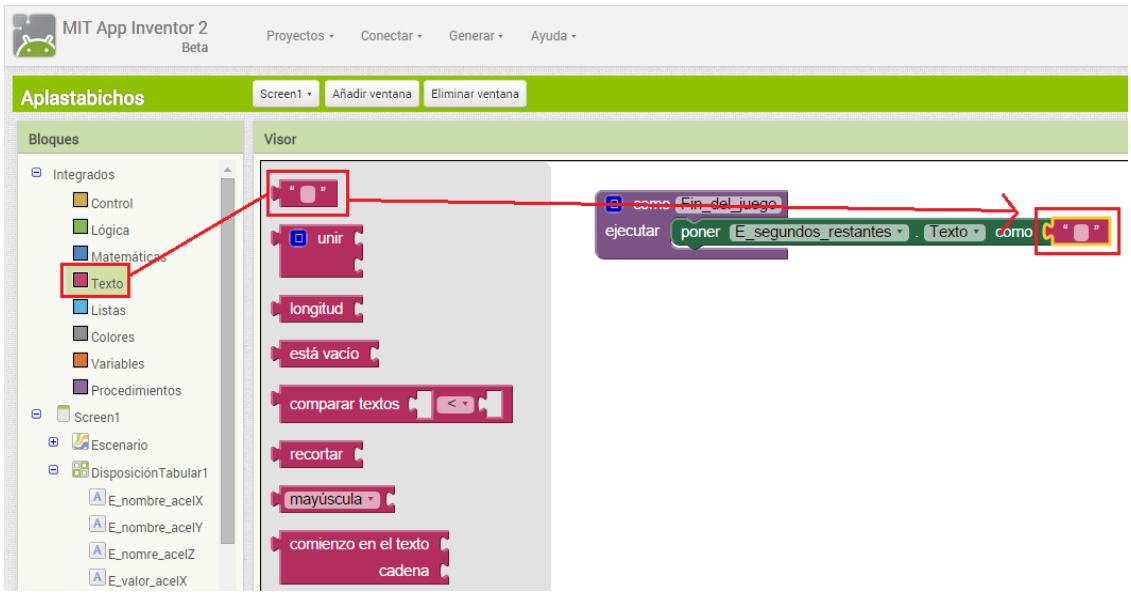
¿En qué otro caso deberá considerarse el juego terminado? Claro, el juego también deberá terminar cuando no queden más objetos por recoger. Lo indicaremos con un bloque **si-entonces** dentro del bloque **cuando.spritePelota.EnColisiónCon.ejecutar.**



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**.



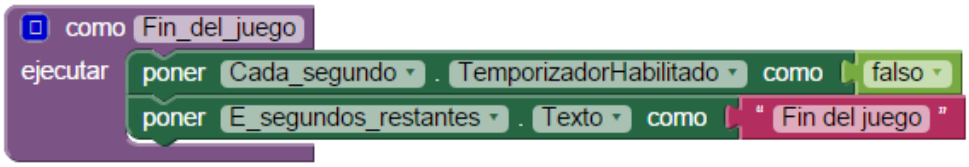
Haciendo clic entre las comillas podremos definir el texto que se mostrará.

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

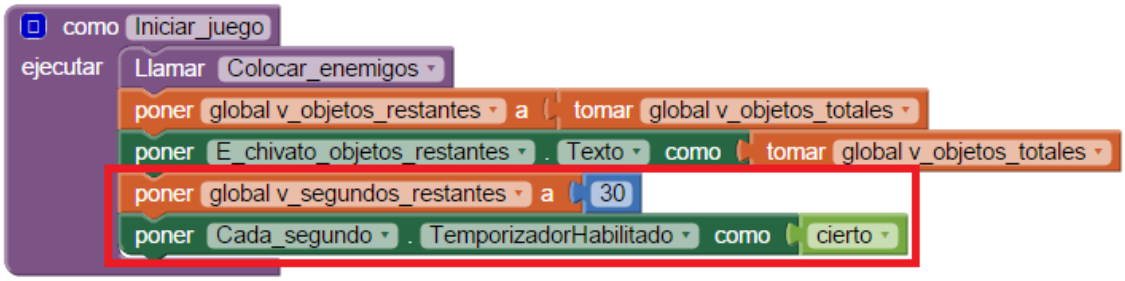
Si ponemos mucha atención veremos que el texto aparece, pero sólo durante un segundo. Esto se debe a que el programa comprueba en el bloque **cuando.Cada_segundo.Temporizador ejecutar** si la variable **Segundos_totales** vale 0, pero esto sólo se cumple durante un segundo, porque el tiempo sigue contando, 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**.

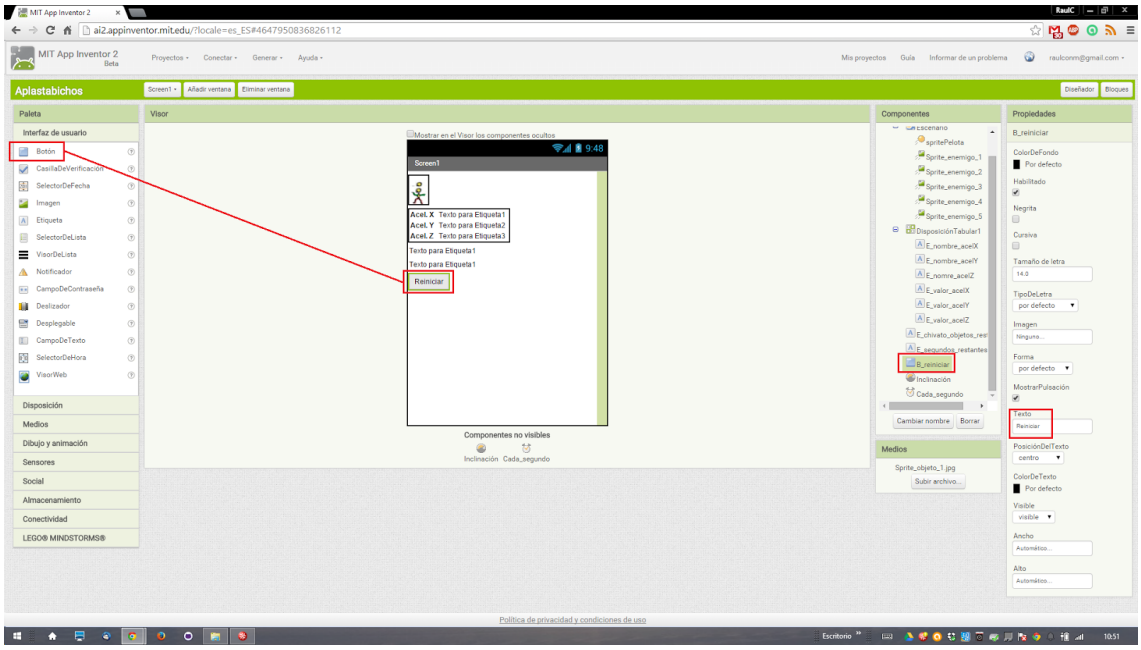


Pero atención, si queremos que el programa funcione correctamente la próxima vez, y ejecute el contenido de **cuando.Cada_segundo.Temporizador ejecutar**, tendremos que poner el valor de esa propiedad como **cierto** en el procedimiento **Iniciar_juego**. Además antes deberemos restablecer el número 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.

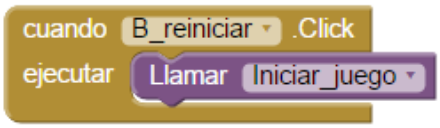


Añadir un botón para empezar de nuevo

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**.

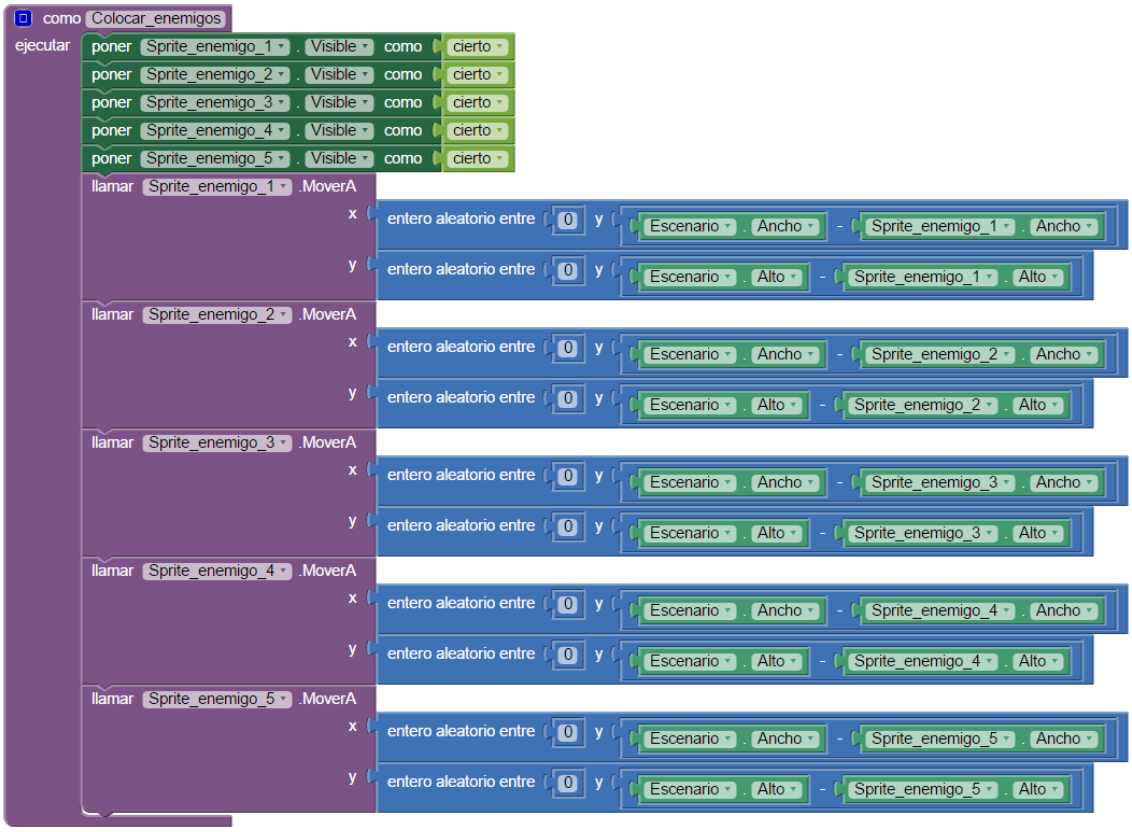


Un último detalle

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