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!

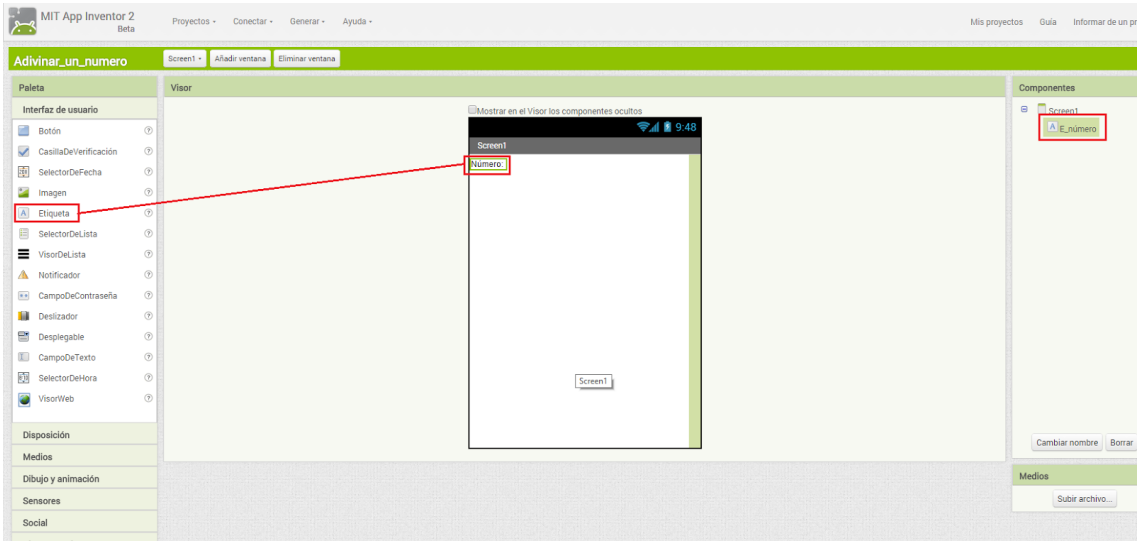
Para que el dispositivo pueda hacer esto correctamente, tenemos que enseñarle cómo hacerlo, paso a paso, añadiendo bloques de código. Se trata de enseñarle al programa la lógica que seguimos nosotros, los seres humanos, cuando jugamos a este juego.

Definimos la interfaz del juego

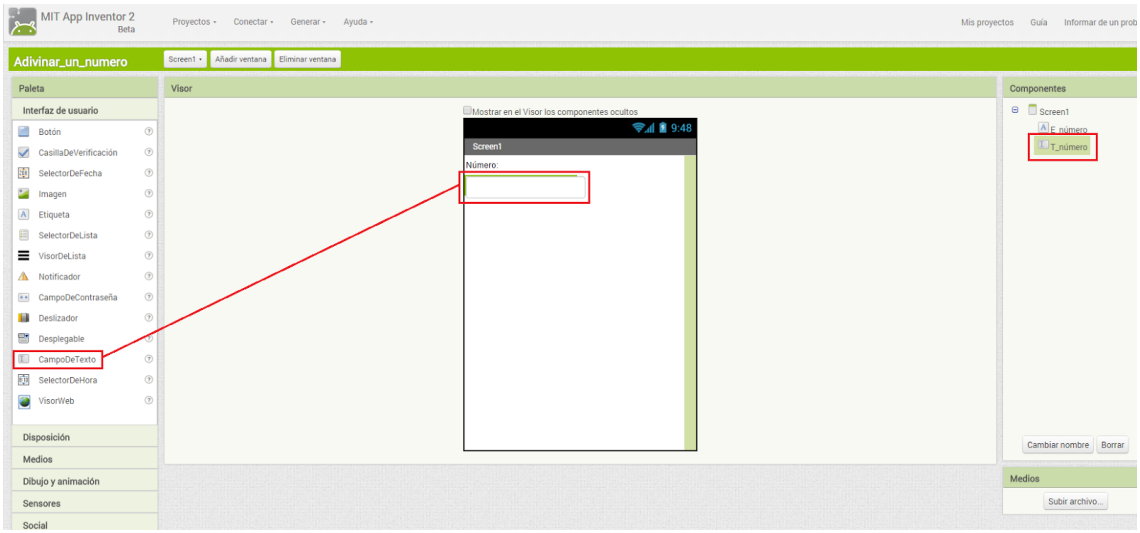
Tenemos que crear un interfaz para para que la aplicación se relacione con el usuario, es decir, le pregunte un número, y le vaya dando pistas, diciendo si es demasiado alto, demasiado bajo, o si finalmente ha acertado el número secreto.

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**.

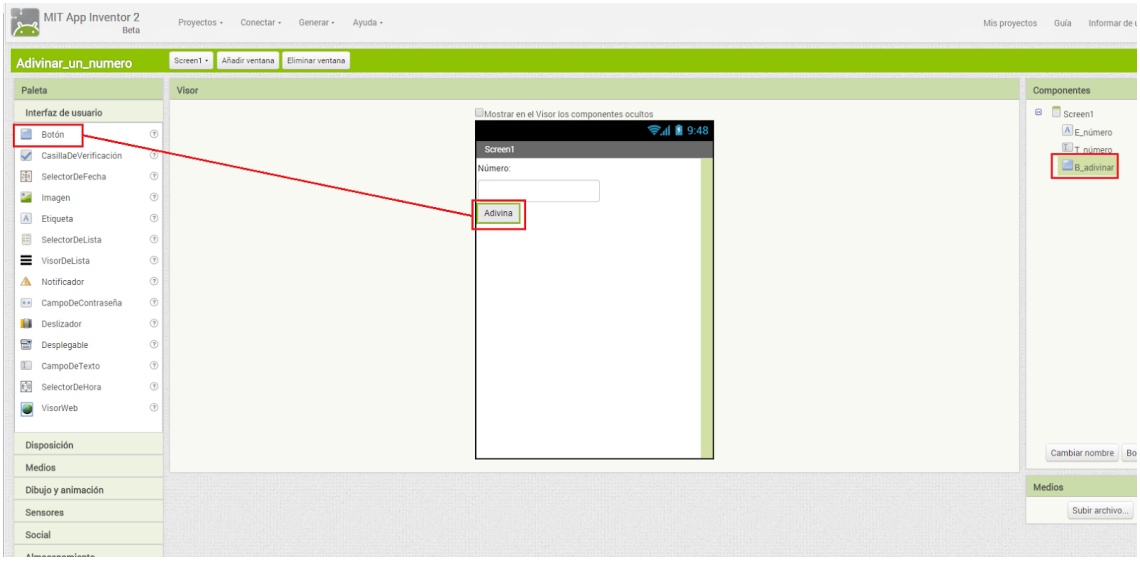


Además el dispositivo va a tener que "escuchar" cada número que le digamos, para decidir si hemos acertado, así que añadiremos en nuestra ventana del visor un campo de texto. Servirá para que el jugador indique qué número cree que ha pensado el programa.



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

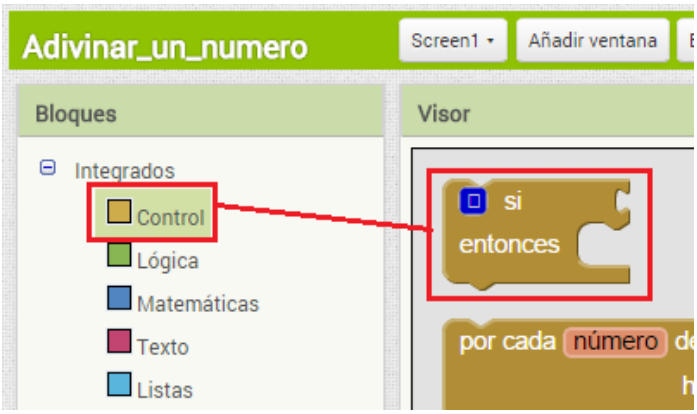
Para comenzar a jugar deberíamos pedirle al jugador que nos diga un número. Esto lo conseguiremos por medio del campo de texto **T_número** que hemos creado en el Visor. El jugador irá escribiendo números en este campo y el programa le irá indicando si el número aleatorio generado secretamente es mayor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

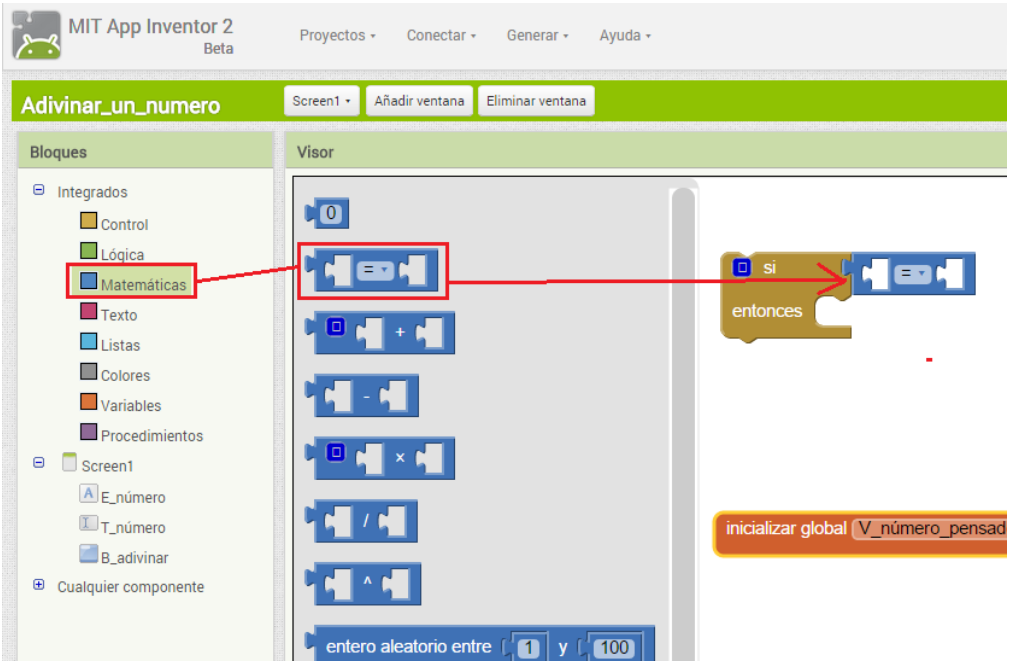
Al bloque **si-entonces** se le tienen que encajar dos bloques más por el lado derecho. El primer encajador, el que sigue a **si**, sirve para indicar la expresión que se va a evaluar. En el caso de nuestro programa, vamos a comparar dos números, el pensado y el que ha dicho el jugador.

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

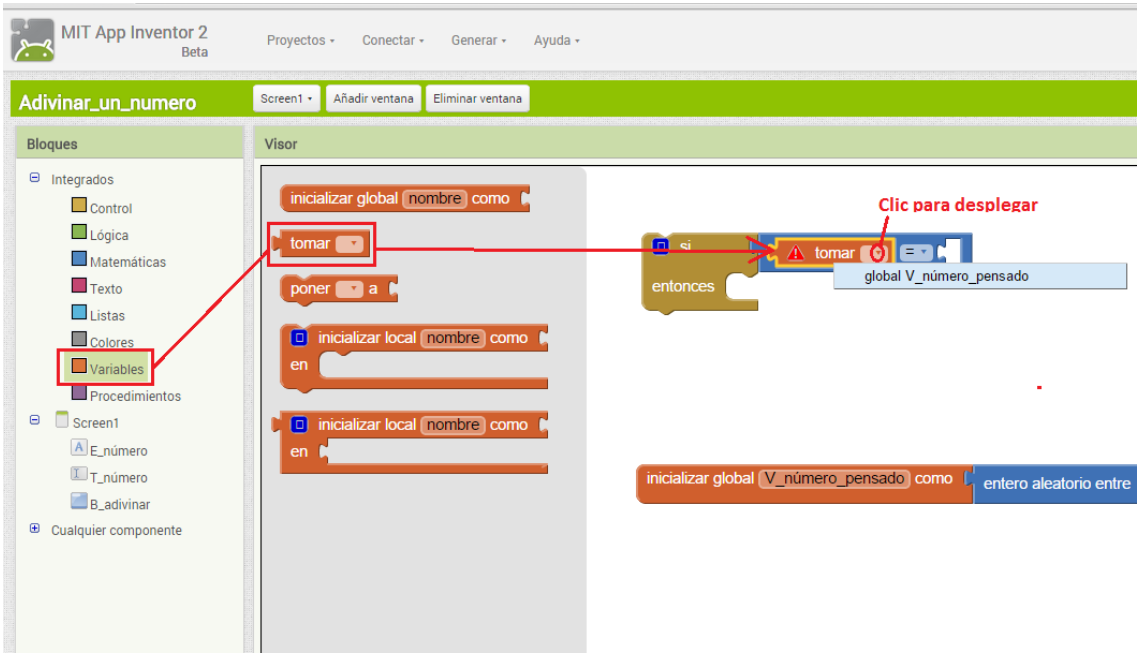
El bloque que sirve para hacer una comparación entre dos números está en el cajón **Matemáticas**. Lo arrastraremos al editor de bloques, hasta encajarlo con el primero de los huecos del bloque **si-entonces**.



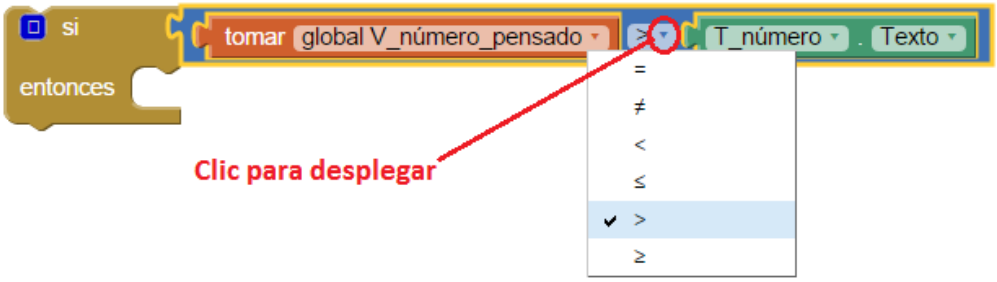
En el primer hueco del bloque azul pondremos el número secreto que almacenamos antes en la variable, y en el otro lado indicaremos cuál es el número que ha escrito el jugador en el campo **T_número**. Antes de hacer esto tenemos que saber cómo manejar el contenido de una variable.

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

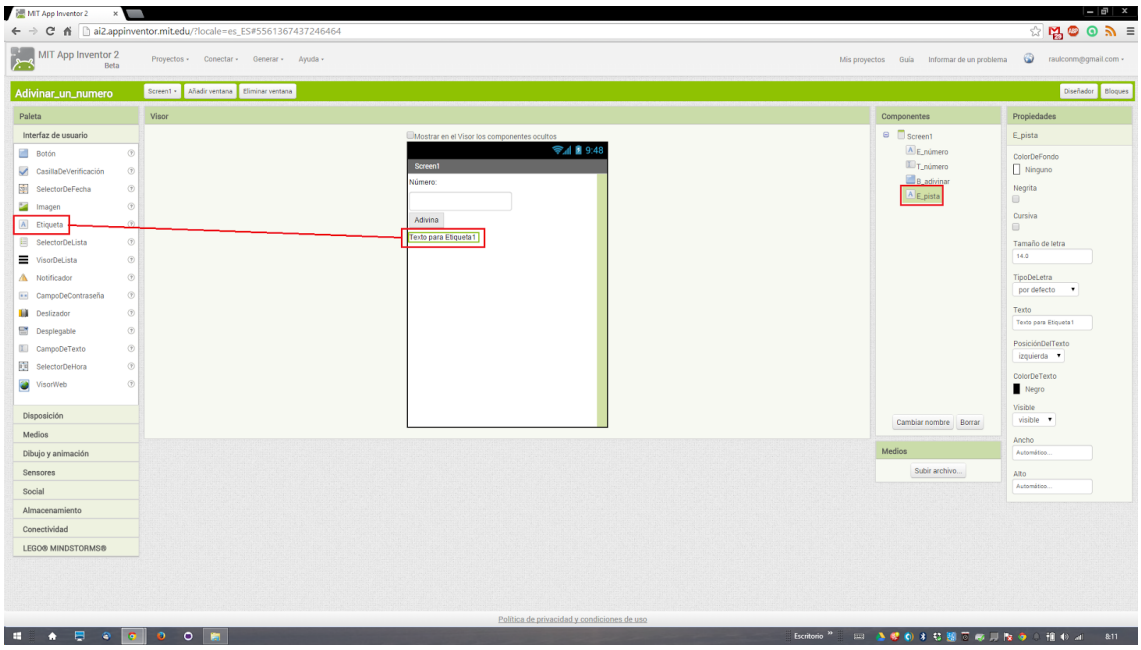
Podemos saber cuál es el contenido de una variable con el bloque **tomar**. Arrastramos al editor el bloque **tomar** que se encuentra dentro del cajón **Variables** hasta el primer hueco del bloque azul de comparación, y una vez colocado elegimos qué variable es la que queremos usar, desplegando la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

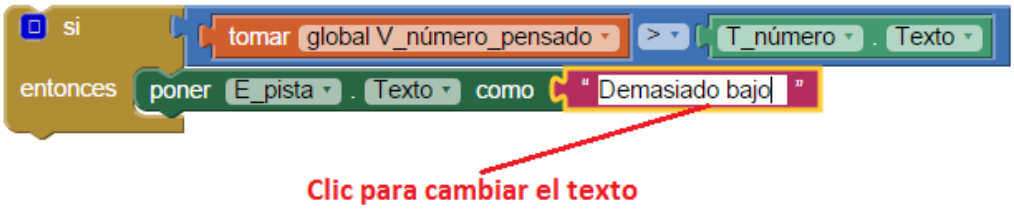
Igual que con el bloque **tomar** tomábamos el valor de la variable, con el bloque **poner** vamos a asignar el valor "Demasiado bajo" al texto de la etiqueta **E_pista,** para que el jugador lo vea en la pantalla del juego.

ATENCIÓN

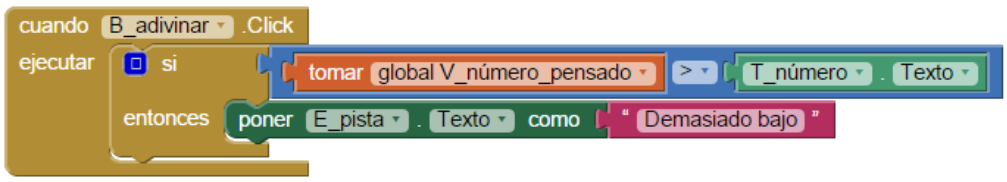
Utilizaremos el componente **Etiqueta** cuando el valor de su propiedad **Texto** sólo va a ser modificada por el programa, es decir, cuando el usuario de la aplicación no tiene que escribir sobre ella para modificarlo. En caso de que el usuario pudiera modificarlo utilizaríamos un componente **CampoDeTexto**.

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.

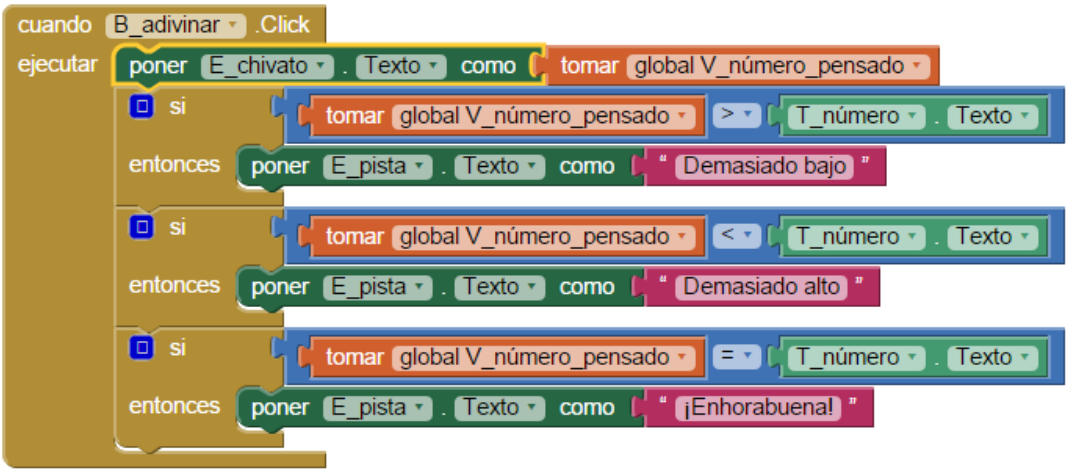
Sucede que como programadores, cuando estamos probando la aplicación, no sabemos cuál es el contenido de la variable **V_número_pensado**, así que no tenemos certeza de si el programa está ejecutándose bien, es decir, si está informando correctamente al jugador si el número es demasiado alto, o demasiado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

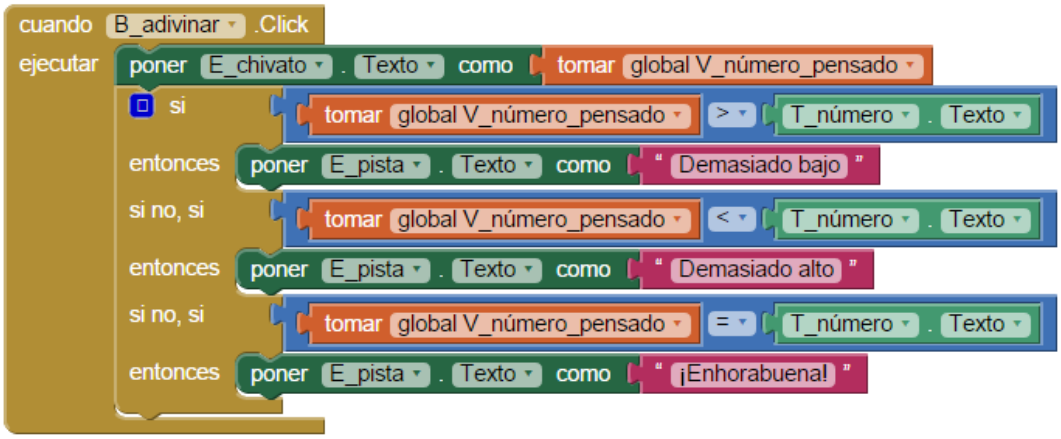
En nuestro programa, para evitar las ejecuciones de código innecesarias, vamos a utilizar sentencias **si-entonces** anidadas, o **si-entonces-si no**. Sólo se ejecutarán los bloques del **si no** cuando NO se cumpla la condición del **si-entonces** anterior.

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.



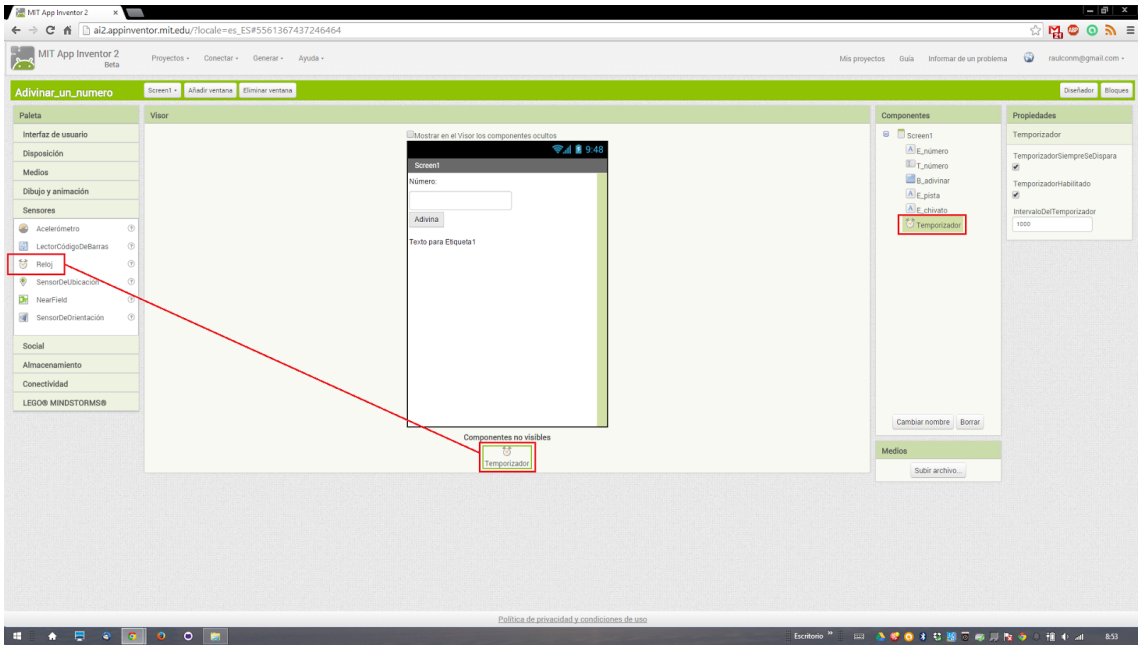
De esta forma el bloque **cuando.B_adivinar_Clic** se ejecutará mucho más rápido, al evitar pasar por bloques sin necesidad.

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.

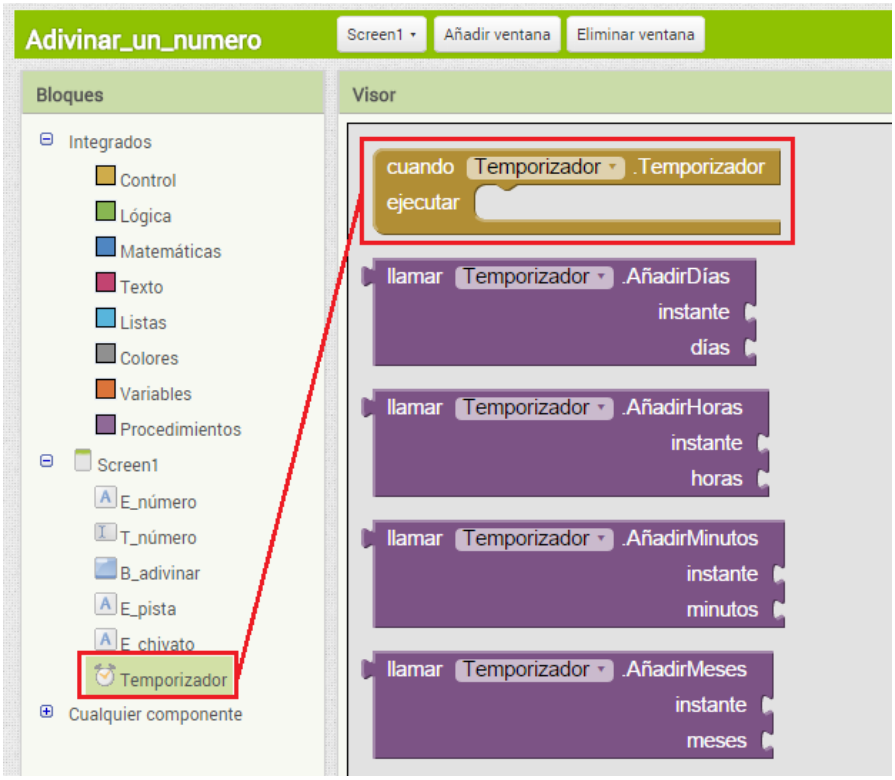
Empezamos añadiendo desde la Paleta al Visor un componente del tipo **Reloj**. Como el reloj es un componente no visible, aparecerá fuera de la zona visible del Visor. Lo llamaremos **Temporizador**.



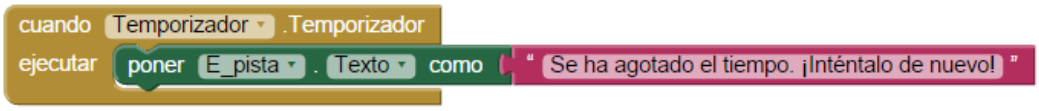
Una de las propiedades más importantes del reloj es **IntervaloDelTemporizador**, que controla cada cuánto tiempo se "dispara" el reloj. Se expresa en milisegundos, y su valor inicial es 1000, o lo que es lo mismo, un segundo. Esto significa que cada segundo el reloj dirá "¡Ha pasado una unidad de tiempo!". 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.

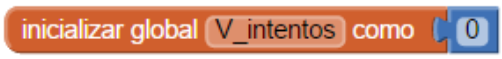


Ahora pondremos los bloques para que el juego nos informe de que han transcurrido los treinta segundos a través de un mensaje de texto. Como llegados a este punto ya no vamos a utilizar más la etiqueta **E_pista**, podemos reutilizarla para mostrar al jugador el texto de tiempo agotado.



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.