

## Tutorial 10 - Creación de un juego (II)

### Paso 1 de 22

Continuando con el juego que comenzamos en el anterior tutorial, vamos a completar su desarrollo añadiendo obstáculos que la nave tiene que evitar.

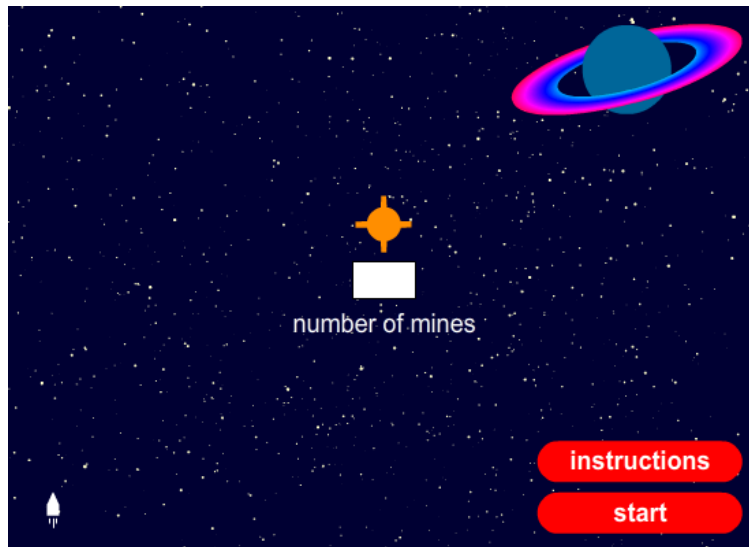
En primer lugar, crearemos para los obstáculos una animación utilizando una interpolación de forma.

Aprenderemos a añadir objetos dinámicamente desde la biblioteca, así como a posicionarlos en el escenario aleatoriamente.

También aprenderemos a generar movimiento aparentemente errático.

Utilizaremos sonidos desde la biblioteca sin necesidad de añadirlos al escenario.

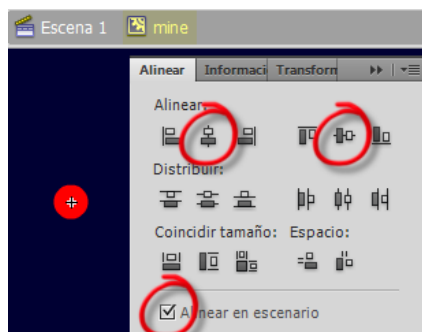
Finalmente, permitiremos al jugador elegir el número de obstáculos que aparecerán en el juego utilizando un campo de introducción de texto.



### Paso 2 de 22

Vamos a crear en primer lugar los obstáculos que tendrá que sortear la nave. Seleccionamos **Insertar > Nuevo símbolo**. Damos al símbolo el nombre *mine* y como tipo seleccionamos **Clip de película**.

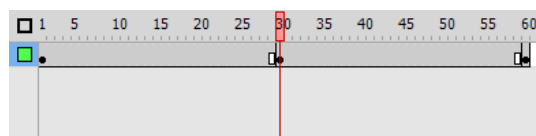
En la línea de tiempo de este nuevo clip, dibujaremos un pequeño círculo rojo de 25 x 25 px, sin trazo y con relleno rojo. Con el panel **Alinear**, centramos este círculo en su escenario.



Este clip tendrá una sencilla animación en la que cambiaremos su forma y su color, para después volver a su forma original.

La animación durará 60 fotogramas. Insertamos un **fotograma clave** pulsando **F6** en el *fotograma 60*. De esta forma se copiará el contenido del fotograma 1 en el fotograma 60.

Insertamos otro **fotograma clave** en el *fotograma 30*. También se copiará el contenido del fotograma 1, pero la intención en este caso es modificar en este fotograma la forma del círculo original.

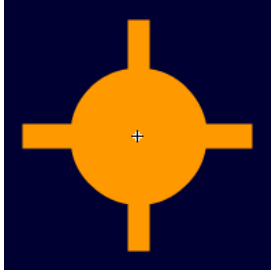


### Paso 3 de 22

Con el fotograma clave 30 seleccionado, dibujamos dos rectángulos, uno vertical y otro horizontal, que centraremos a medida que los dibujemos. Podemos trabajar con zoom para mayor comodidad.

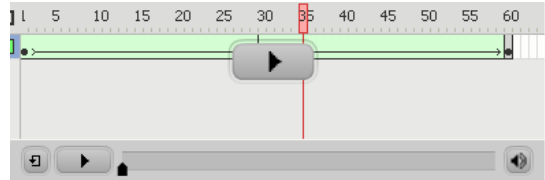
Estos rectángulos tampoco tendrán trazo y tendrán como relleno el mismo color rojo que el círculo. De esta manera, los rectángulos quedarán unidos al círculo en una sola forma.

Por último cambiamos el color de la forma compuesta a un tono anaranjado.



Para crear una transición desde la forma del círculo original al círculo con los rectángulos, y otra transición hasta volver de nuevo a la forma y color originales, tenemos que crear interpolaciones de forma.

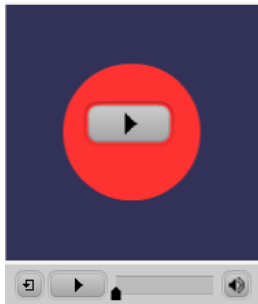
Para ello hacemos clic con el **botón derecho** del ratón sobre el área gris que hay entre dos fotogramas clave, y seleccionamos en el menú contextual **Crear interpolación de forma**. Repetimos el proceso para la transición comprendida entre los otros dos fotogramas clave.



Podemos crear interpolaciones de forma entre dos formas cualesquiera. Para controlar cambios de forma complejos, podemos utilizar los consejos de forma, aunque en nuestro caso no va a ser necesario. Consulta la ayuda de Flash a este respecto si necesitas crear interpolaciones de forma complejas.

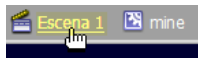
### Paso 4 de 22

Si pulsamos Intro o movemos la cabeza lectora, veremos que la forma cambia del círculo rojo inicial, a la forma compuesta anaranjada, para volver posteriormente al círculo rojo.



Podemos experimentar con diferentes formas o colores hasta crear una interpolación de forma que nos guste más.

Con esto damos por finalizada la creación de la mina, por lo que volvemos a la escena principal.

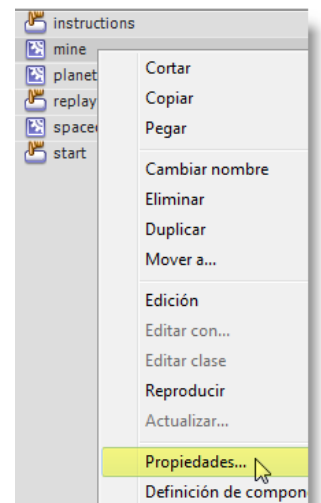


Vamos a añadir copias al escenario del clip *mine* utilizando ActionScript.

Hasta ahora, para poder hacer referencia a un objeto utilizábamos en la programación su nombre de instancia. Para asignar un nombre de instancia a un clip, seleccionábamos el clip en el escenario y escribíamos su nombre de instancia en el inspector de Propiedades.

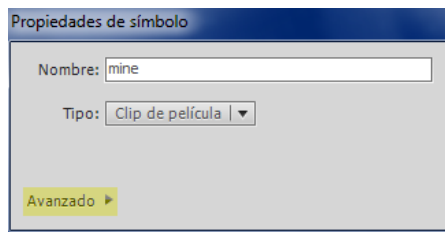
En el caso en el que el objeto no esté en el escenario, tenemos que exportarlo para ActionScript desde la biblioteca.

Hacemos clic con el **botón derecho** del ratón sobre el nombre del clip (*mine* en este caso), y seleccionamos **Propiedades** en el menú contextual.

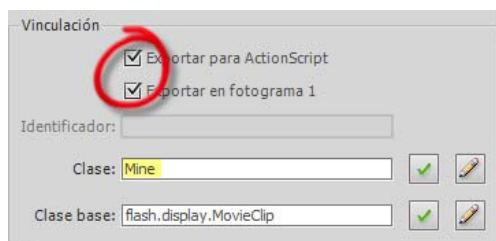


## Paso 5 de 22

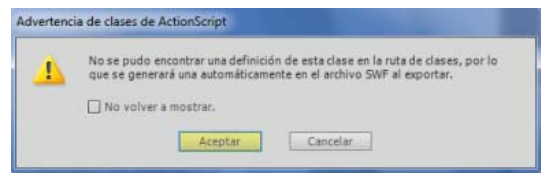
En la ventana **Propiedades de símbolo**, pulsamos sobre el botón **Avanzado**.



En el área **Vinculación** marcamos las casillas **Exportar para ActionScript** y **Exportar en fotograma 1**. Escribimos el nombre de clase *Mine* (por convención los nombres de las clases comienzan con mayúscula).



Al pulsar en **Aceptar** nos aparecerá una ventana de advertencia. Pulsamos de nuevo en **Aceptar** para que se genere de forma automática la definición de la clase *Mine*.



Podemos comprobar que en el panel **Biblioteca**, en el área de **Vinculación**, aparece el nombre de clase que hemos asignado al clip. Podríamos también haber puesto el nombre de vinculación directamente desde este panel, y el clip se exportaría igualmente.



Después de haber exportado el clip en una clase llamada *Mine*, ya podemos hacer referencia a esta clase en nuestra programación.

## Paso 6 de 22

Vamos a definir una nueva función a la que llamaremos `putMines`.

```
function putMines():void
{
    var mine:Mine;
    mine = new Mine;
    mine.x = 275;
    mine.y = 200;
    stage.addChild(mine);
}
```

En la primera línea de esta función, definimos que `mine` es un objeto de la clase *Mine*. Después creamos una instancia de esa clase, que será una nueva copia del clip que tenemos en la biblioteca. Después de esta definición, ya podemos hacer referencia al nuevo objeto instanciado.

En las siguientes líneas, definimos que la posición del objeto sea el centro del escenario (`x:275` e `y:200`). Por último, añadimos ese objeto al escenario (`stage`) utilizando el método `addChild`.

Para que esta función se ejecute, en algún momento tenemos que llamarla con la instrucción `putMines()`. Escribiremos esta instrucción dentro de la función `playGame`, ya que será el momento en el que queremos que se añadan las minas al escenario.

Si probamos la película, comprobaremos que al pulsar el botón `start` (que llama a la función `playGame`, que a su vez llama a la función `putMines`), aparece una mina en el centro del escenario.

Para que la posición en la que aparece la mina sea aleatoria, vamos a utilizar el método `random` de la clase *Math*.

`Math.random()` devuelve un número aleatorio comprendido entre 0 y 1. Si multiplicamos ese valor aleatorio por 550 (la anchura del escenario), el número devuelto estará comprendido entre 0 y 550.

Por lo tanto, para crear una posición aleatoria de la mina dentro del escenario, que mide 550 x 400, las líneas que determinan la posición del clip dentro de la función `putMines` quedarán como sigue:

```
mine.x = Math.random() * 550;
mine.y = Math.random() * 400;
```

La posición `x` del clip será un número comprendido entre 0 y 550, mientras que la posición `y` será un número comprendido entre 0 y 400. Cada vez que se ejecute la función `Math.random()`, el número devuelto será diferente.

## Paso 7 de 22

Por ahora sólo se añade una copia del clip al escenario. Para añadir varias copias, debemos hacer un bucle que cree varias minas y las añada al escenario.

Vamos a definir, junto con las variables `course` y `speed`, una nueva variable llamada `numMines`, cuyo valor será el número de minas que queremos crear (por ejemplo 10).

```
var numMines:Number = 10;
```

Ahora utilizaremos un bucle `for` para crear estas 10 copias de la mina.

Los bucles `for` tienen la siguiente estructura:

```
for (initial value; conditional statement;
expression that changes the value)
{
    //statements
}
```

Por ejemplo, en nuestro caso:

```
for (var i:Number = 0; i < numMines; i++)
{
    //statements
}
```

Este bucle funcionaría de la siguiente manera:

- Creamos una variable llamada `i` con un valor inicial de 0.
- Comprobamos si se cumple la condición, que en este caso es que el valor de `i` sea menor que el valor de `numMines`.
- Al cumplirse la condición de que `i < numMines`, ejecutaremos las sentencias que se encuentren entre las llaves del bucle `for`.
- Aumentamos el valor de `i` en una unidad (`i++` significa `i = i + 1`).
- Volvemos a comprobar la condición. Ahora `i` vale 1, que sigue siendo menor que 10 (valor de `numMines`).
- Como la condición se sigue cumpliendo, volvemos a ejecutar las sentencias, y sumamos otra unidad a `i`, que ahora valdrá 2.
- Cuando `i` tenga un valor de 10, momento en el que no se cumplirá que `i` sea menor que `numMines`, ya no se ejecutará el bucle. Al empezar con un valor de `i = 0`, el bucle se habrá ejecutado un total de 10 veces.

## Paso 8 de 22

Por lo tanto, para crear la cantidad de minas que hayamos indicado en la variable `numMines`, la función `putMines` quedará como sigue:

```
function putMines():void
{
    var mine:Mine;
    for (var i:Number = 0; i < numMines; i++)
    {
        mine = new Mine;
        mine.x = Math.random() * 550;
        mine.y = Math.random() * 400;
        stage.addChild(mine);
    }
}
```

Si probamos ahora la película, podemos comprobar que aparecen 10 minas en el escenario.

Todas las minas realizan su animación al mismo tiempo. Podemos hacer que cada mina comience su animación en un fotograma diferente, llevando su cabeza lectora a un fotograma aleatorio entre el 1 y el 60, que es el número de fotogramas que tiene la animación.

Esta vez necesitamos por tanto generar un número aleatorio entre 1 y 60. El número devuelto tiene en este caso que ser entero.

Si utilizamos `Math.random() * 60` obtendremos números decimales entre 0 y 60. Para asegurarnos de que el número resultante sea un entero comprendido entre 1 y 60 podemos utilizar el método `ceil`, que redondea al alza un número decimal.

Por lo tanto, dentro del bucle `for`, después de haber determinado una posición aleatoria para cada mina y antes de añadir la mina al escenario, escribiremos la instrucción:

```
mine.gotoAndPlay(Math.ceil(Math.random() * 60));
```

Probamos de nuevo la película. Ahora, al comenzar el juego, se crean 10 copias de la mina, comenzando la animación de cada una de ellas en momentos diferentes.

## Paso 9 de 22

Para convertir las minas en obstáculos contra los que debemos evitar chocar, añadiremos un detector a cada mina, para que evalúe en cada momento si está chocando con la nave.

Para ello añadimos en primer lugar un listener para cada mina en el mismo bucle que utilizamos para crearlas, dentro por tanto de la función `putMines` y dentro del bucle `for`, y antes de añadir las minas al escenario con `addChild`.

```
mine.addEventListener(Event.ENTER_FRAME, enemy);
```

La nueva función, a la que hemos llamado `enemy`, comprobará si cada mina a la que hemos añadido el listener choca con la nave en algún momento. Si choca, entonces ejecutaremos la función `gameOver`, con el parámetro `"lose"`.

```
function enemy(e:Event):void
{
    if (e.target.hitTestObject(spacecraft_mc))
    {
        gameOver("lose");
    }
}
```

Si probamos ahora nuestro juego, tanto si ganamos como si perdemos, nos aparecerá un error debido a que, a pesar de que no hay ninguna nave, la función `enemy` seguirá comprobando si cada mina choca con la nave. Al no encontrar ninguna nave en el escenario se mostrará el error.

Por tanto, lo primero que debemos hacer en la función `gameOver`, antes de la orden de ir a otro fotograma en el que no esté la nave, es eliminar los listeners que hemos añadido a las minas. A la vez que eliminamos cada listener, aprovecharemos para eliminar las minas del escenario. En el siguiente paso mostraremos cómo hacerlo.

El escenario (`stage`) funciona como un contenedor que contiene en primer lugar la línea de tiempo, y contiene también cada mina que le hemos ido añadiendo con `addChild`. Después de crear todas las minas, el escenario tendrá 11 elementos secundarios. El primero, en el nivel inferior de visualización (nivel 0), será la línea de tiempo. Después estarán todas las minas, cada una en un nivel superior a la anterior.

La cantidad de minas creadas depende del valor que hayamos dado a `numMines`. La última mina creada se mostrará por tanto en un nivel de visualización que coincide con `numMines`.

## Paso 10 de 22

Al comienzo de la función `gameOver`, antes de las sentencias en las que eliminábamos los listeners de la nave, podemos eliminar el listener de cada mina, y después eliminar cada mina en sí, con el siguiente código:

```
for (var i:Number = numMines; i > 0; i--)
{
    stage.getChildAt(i).removeEventListener(Event.ENTER_FRAME, enemy);
    stage.removeChildAt(i);
}
```

Analicemos lo que realiza este bucle. El bucle comenzará con un valor de `i` igual a `numMines`, que en nuestro caso es 10. Como la variable `i` es mayor que 0, se ejecutarán las sentencias que se encuentran entre las llaves.

Después reduciremos en una unidad el valor de `i` (`i--` significa `i = i - 1`). Por tanto, `i` valdrá 10 en la primera ejecución del bucle, 9 en la siguiente vuelta, etc., hasta el último valor que cumple la condición, es decir, hasta que `i` valga 1 (cumple `i > 0`).

En las sentencias que se encuentran dentro del bucle, en primer lugar eliminamos el listener del elemento que se encuentra contenido dentro del `stage`, en el nivel de visualización `i` (10, 9, 8, 7, ..., 1), que en el nuestro es cada mina que hemos creado. De esta forma ya no tratarán de detectar posibles colisiones con la nave. Con la siguiente línea eliminamos la mina a la que acabamos de eliminar su listener.

Ahora nuestro juego será funcional y no tendrá errores, pero vamos a introducir mejoras en los siguientes pasos:

- En primer lugar, nos aseguraremos de que la posición inicial de las minas no esté muy cerca de la nave, para que nos dé tiempo a empezar a jugar.
- Para hacer el juego más complejo, añadiremos un movimiento aleatorio de las minas por el escenario.
- Cambiaremos la imagen del planeta por la imagen de una mina en el fotograma `lose`.
- Después añadiremos un sonido para cuando ganemos y otro para cuando perdamos.
- Por último, permitiremos al usuario decidir el número de minas que quiere que aparezcan en el juego.

## Paso 11 de 22

Comencemos por la primera mejora.

Debido a la aleatoriedad de la posición de las minas, es posible que antes incluso de mover la nave, ya haya alguna mina que choque con ella.

Para solucionar este problema, sustituiremos el código en el que creábamos la posición `x` e `y` de cada `mine`, dentro de la función `putMines`, por este código:

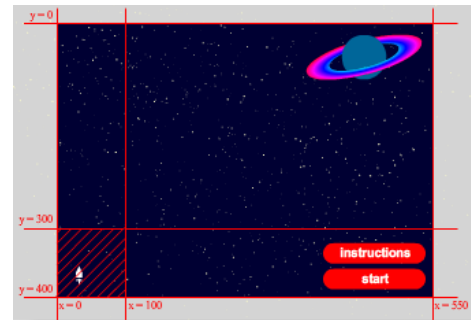
```
do
{
    mine.x = Math.random() * 550;
    mine.y = Math.random() * 400;
} while (mine.x < 100 && mine.y > 300);
```

La sentencia `do...while` lo que hace es ejecutar en primer lugar lo que se encuentre entre llaves. Mientras se cumpla la condición que está entre los paréntesis del `while`, entonces lo que está entre las llaves del `do` volverá a ejecutarse.

En este caso, creamos una posición al azar para la mina. Si la posición se encuentra cerca de la esquina inferior izquierda (si la posición horizontal de la mina es menor que 100, y además su posición vertical es mayor que 300), entonces volverá a generarse una posición aleatoria para la mina.

Este proceso se repetirá hasta que la condición no se cumpla, lo que significará que la posición generada para la mina ya no está cerca de la nave.

De un modo más gráfico, si la posición aleatoria de la mina está en la zona del cuadrado rayado, entonces cambiaremos la posición de la mina de forma aleatoria, hasta dar con una posición válida.



## Paso 12 de 22

Vamos a añadir el movimiento de vibración de las minas en el escenario. Creamos una nueva variable al inicio de la programación, junto con las variables `course`, `speed` y `numMines`, a la que llamaremos `vibration`. Le asignamos un valor de 5.

```
var vibration:Number = 5;
```

Vamos a programar que la posición de la mina varíe desde la posición en la que se encuentra en cada momento, a una nueva posición aleatoria que se encuentre a una distancia máxima de 5 píxeles de la posición actual, que es la cantidad que hemos asignado a la variable `vibration`.

Es decir, si la posición actual de una mina fuera `x=200` e `y=300`, al instante siguiente el valor de `x` estaría entre 195 y 205, mientras que el valor de `y` podría estar entre 295 y 305. De esta forma, la mina vibrará un máximo de 5 píxeles en cada sentido (horizontal y vertical).

Como hemos aprendido anteriormente, la sentencia

```
Math.random() * vibration
```

devolverá un valor que se encuentra entre 0 y 5 (que es el valor que hemos dado a la variable `vibration`).

Por lo tanto, la sentencia

```
Math.random() * vibration - Math.random() * vibration
```

devolverá un número entre -5 (0-5) y 5 (5-0).

Dentro de la función `enemy` (que se ejecuta llamada por un `ENTER_FRAME` de cada `mine`), pero fuera del condicional que evalúa si hay choque entre la mina y la nave, escribiremos el siguiente código:

```
e.target.x += Math.random() * vibration -
Math.random() * vibration;
e.target.y += Math.random() * vibration -
Math.random() * vibration;
```

Para evitar que con esta vibración aleatoria la mina pueda acabar fuera del escenario, añadiremos dentro de la misma función unas sentencias condicionales que evalúen si la mina está fuera de los límites, y de estarlo, la colocaremos de nuevo en el límite del escenario.

```
if (e.target.x < 0)
{
    e.target.x = 0;
}
```

Este condicional evitará que la mina salga por el lado izquierdo del escenario. Debemos añadir otros condicionales similares para el resto de los casos (si `x > 550` entonces que `x = 550`, si `y < 0` entonces `y = 0`, y si `y > 400` entonces `y = 400`).

## Paso 13 de 22

Probamos la película para comprobar si se efectúa la vibración de las minas. Veremos que aunque las minas nunca desaparecen completamente del escenario, puede que veamos media forma de la mina fuera del escenario en algunos momentos, porque el punto de registro de la mina se encuentra en su centro.

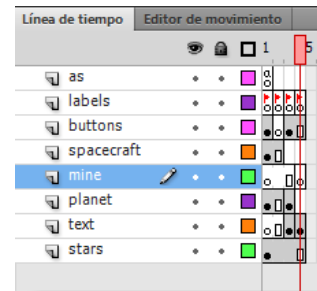
Si queremos, podemos modificar los límites de movimiento de la mina cambiando los valores de los condicionales del paso anterior. Por ejemplo, podemos reducir el espacio en el que pueden moverse las minas en 25 píxeles en cada lado.

La siguiente mejora va a ser eliminar el planeta en el cuarto fotograma (fotograma *lose*), y mostrar como sustitución una mina en el centro del escenario.

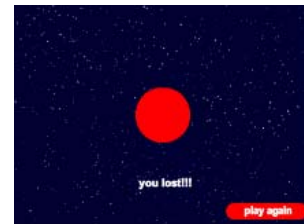
En primer lugar, eliminamos el cuarto fotograma de la capa *planet* pulsando sobre él con el **botón derecho** del ratón, y seleccionando **Quitar fotogramas**.

Añadimos una nueva capa a la que llamaremos *mine*. Situamos esta capa por encima de la capa *planet*.

Después pulsamos con el **botón derecho** del ratón el cuarto fotograma de la capa *mine*, y seleccionamos **Insertar fotograma clave**.



En este nuevo fotograma clave, arrastramos una instancia del clip *mine* desde la biblioteca al centro del escenario, y aumentamos su tamaño hasta por ejemplo *AN:100* y *AL:100*.

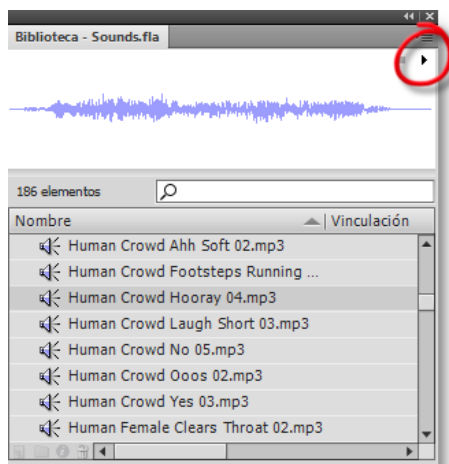


## Paso 14 de 22

Vamos a buscar sonidos que asociaremos al momento de ganar o de perder la partida.

En **Ventana > Bibliotecas comunes > Sonidos** podemos encontrar algunos ejemplos de sonidos que podemos utilizar en nuestros proyectos.

Para escuchar los diferentes sonidos, seleccionamos el sonido en la biblioteca y pulsamos sobre el pequeño botón para reproducir el que se encuentra en la ventana donde se muestra la onda del sonido.



Utilizaremos por ejemplo los sonidos *HumanCrowdHooray04.mp3* para el momento de ganar, y *Multimedia Internet CD-RomFlash Blast06.mp3* para el de perder.

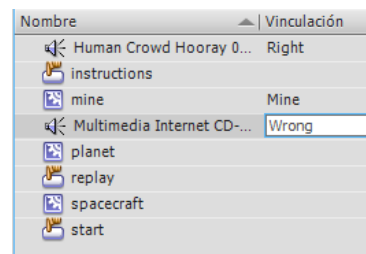
Podemos ampliar el tamaño de la biblioteca de sonidos o el área de los nombres para leer los nombres de los sonidos con más comodidad.

Para utilizar estos sonidos, los podemos arrastrar directamente desde la biblioteca de sonidos hasta la biblioteca de nuestro juego.

Una vez incorporados en nuestra biblioteca, podemos cerrar la biblioteca de sonidos.

Al igual que hicimos con el clip de la mina, vamos a exportar ambos sonidos para ActionScript.

Esta vez vamos a pulsar directamente sobre el **área de Vinculación de la Biblioteca**, asignando a un sonido el nombre de clase *Right* y al otro sonido el nombre de clase *Wrong*.



## Paso 15 de 22

Para utilizar estos sonidos añadiremos, junto con la definición de otras variables al inicio de la programación, las siguientes líneas:

```
var soundWin:Right = new Right();  
var soundLose:Wrong = new Wrong();
```

La variable `soundWin` pertenecerá a la clase `Right` y será una nueva instancia de ese sonido. De la misma forma, `soundLose` será un objeto de la clase `Wrong`.

Dentro de la función `gameOver` reproduciremos un sonido u otro dependiendo de si hemos ganado o hemos perdido.

En esta función, la variable que nos indica si hemos ganado o perdido es `frameLabel`. Por lo tanto, lo que haremos es comprobar si `frameLabel` tiene el valor "win". De ser así, reproduciremos `soundWin`. Si no fuera así, reproduciremos `soundLose`.

Por tanto, dentro de la función `gameOver` incluiremos este código:

```
if (frameLabel == "win")  
{  
    soundWin.play();  
}  
else  
{  
    soundLose.play();  
}
```

El doble signo de igual (==) compara si `frameLabel` es "win". Este signo es para comparar, mientras que un solo signo de igual (=) es para asignar.

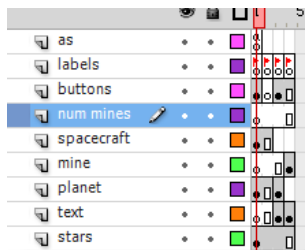
Si `frameLabel` es "win", se reproducirá `soundWin`. En caso de no serlo (else) se reproducirá `soundLose`.

## Paso 16 de 22

Ya tenemos el juego prácticamente finalizado. Podemos aumentar los valores a las variables `speed`, `numMines` y/o `vibration` para aumentar la dificultad del juego. Por ejemplo, una vibración de 15 complicaría considerablemente el juego, ya que el movimiento de las minas sería menos predecible.

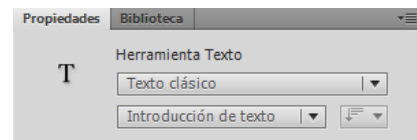
Podemos hacer que el propio jugador modifique estos valores mediante un campo de introducción de texto. Como ejemplo de ello, vamos a permitir que el jugador decida el número de minas con el que quiere jugar.

Para ello creamos en primer lugar una nueva capa llamada `num mines` y la situamos bajo la capa `buttons`.



Seleccionamos la herramienta **Texto**.

En el inspector de Propiedades seleccionamos **Texto clásico** e **Introducción de texto**.



Hacemos clic sobre cualquier punto en el escenario para crear el campo de introducción de texto en la capa `num mines`.

Seleccionamos este campo de texto recién creado. En el inspector de Propiedades, le asignamos el nombre de instancia `mines_txt`.

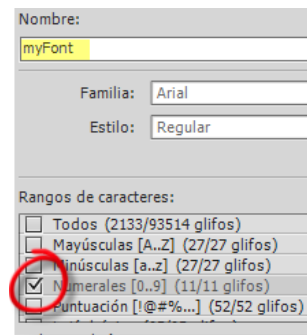


## Paso 17 de 22

Excepto en el caso en el que utilizemos fuentes del dispositivo, debemos incorporar las fuentes que vayamos a utilizar en los campos de texto dinámico o de introducción de texto.

En el área **Carácter** seleccionamos el tipo de letra y el tamaño que queramos, y hacemos clic sobre el botón **Incorporar**.

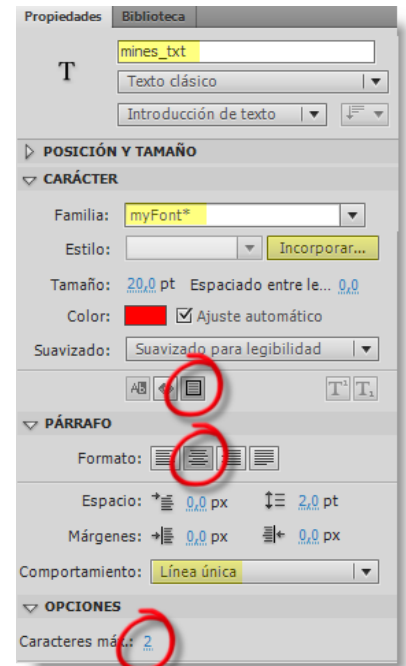
En la pantalla de incorporación de fuentes, escribimos el nombre *myFont* para esta fuente incorporada.



En Familia y Estilo aparecerá preseleccionado lo que teníamos en el inspector de Propiedades.

En **Rangos de caracteres** seleccionamos **Numerales**, ya que son los únicos caracteres que necesitamos.

Una vez aceptada la pantalla de Incorporación de fuentes, volvemos al inspector de Propiedades y en Familia seleccionamos en el desplegable la fuente que hemos incorporado. Su nombre aparecerá por encima de la lista del resto de las fuentes, e irá seguido de un asterisco para indicar que se trata de una fuente incorporada.



Seleccionamos un **color rojo** para que destaque sobre el fondo, y marcamos la casilla **Mostrar borde alrededor del texto**.

Esto hará que aparezca un recuadro blanco con un marco negro. El marco negro apenas lo veremos, ya que nuestro juego tiene un fondo muy oscuro, pero el color blanco del recuadro nos mostrará claramente dónde se encuentra nuestro campo de texto.

En el área **Párrafo**, alineamos el texto en el **centro** y elegimos un comportamiento de **Línea única**. Por último, en el área **Opciones** seleccionamos que se puedan escribir un máximo de 2 caracteres.

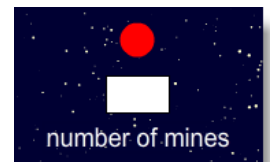
## Paso 18 de 22

Para que el bloque de texto no nos quede demasiado grande, podemos escribir en él dos números de prueba, y ajustar entonces el tamaño del campo.

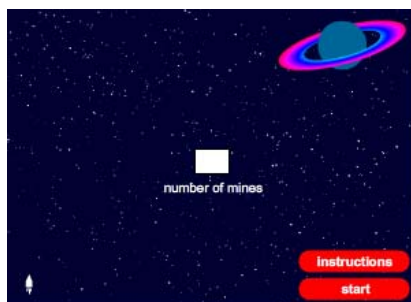
Una vez adaptado el tamaño, borramos los números que habíamos escrito como guía y centramos el campo de texto en el escenario.



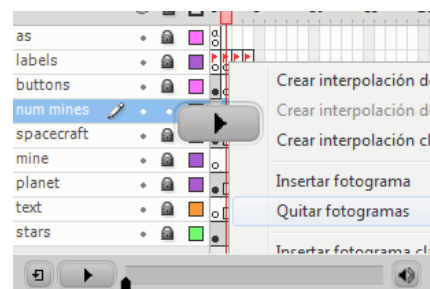
También podemos incluir en la misma capa una instancia de *mine*, para que se muestre sobre el campo de texto. Para ello la arrastramos directamente desde la biblioteca.



En la misma capa, creamos un campo de **texto clásico estático** bajo el campo de introducción de texto, y escribimos el texto *number of mines* en color blanco.



Para que estos elementos sólo se muestren en el primer fotograma, seleccionamos el resto de fotogramas de esa capa, hacemos clic con el **botón derecho** del ratón, y seleccionamos **Quitar fotogramas** del menú contextual.



## Paso 19 de 22

Volviendo a la programación, podemos indicar en primer lugar que, antes de pulsar ningún botón, el foco del escenario se encuentre en el campo de introducción de texto, al que habíamos llamado *mines\_txt*. De esta forma podremos escribir directamente en el campo de texto, sin necesidad de hacer clic previamente sobre él.

Para ello añadiremos, junto a la definición inicial de las variables de nuestro juego, la siguiente instrucción:

```
stage.focus = mines_txt;
```

Después, cuando pulsemos sobre el botón *start*, el foco pasará a la nave, ya que es lo que habíamos indicado en la función *playGame*. Por ello no hay ningún problema en establecer el foco inicial del juego en el campo de texto.

Si probamos ahora el juego, comprobaremos que si escribimos un número con el teclado, éste aparecerá en el campo de texto si necesidad de haberlo seleccionado previamente.

Cambiamos el valor inicial de la variable *numMines* a 0 en vez de 10:

```
var numMines:Number = 0;
```

Una vez que se pulse el botón *start*, lo primero que se debe hacer es asignar a la variable *numMines* el valor que hayamos escrito en el campo *mines\_txt*.

Por lo tanto, la primera instrucción dentro de la función *playGame*, será:

```
numMines = Number(mines_txt.text);
```

Para acceder al contenido de un campo de texto, tenemos que acceder a la propiedad *text* del campo. El contenido de un campo de texto es siempre de tipo *String*. Sin embargo, la variable *numMines* es de tipo numérico. Por ello es necesario convertir el contenido textual del campo de texto *mines\_txt* en número utilizando *Number*.

Probamos nuevamente nuestro juego. El número de minas se corresponderá al que indiquemos en el campo *mines\_txt*. Sin embargo, el juego también comenzará si dejamos el campo en blanco o si introducimos otros caracteres no numéricos. El siguiente paso será controlar estas posibles circunstancias.

## Paso 20 de 22

En la función *playGame* podemos indicar que, una vez que tengamos el *numMines* a partir del texto introducido en el campo de texto, se evalúe el número de minas.

Si se introduce un número de minas que se encuentre entre 1 (que haya al menos una mina) y 50 (número con el que es muy difícil llegar al planeta), entonces que se ejecuten todas las demás instrucciones de la función *playGame*.

Por lo tanto, en la función *playGame* primero estará la línea

```
numMines = Number(mines_txt.text);
```

Después habrá un condicional que evaluará si *numMines* está dentro del margen que consideramos válido, y, si es así, se ejecutará el resto de la función y el juego comenzará.

```
if (numMines > 0 && numMines < 51)
{
    /*put here the rest of the statements we
    have in PlayGame function and start the game*/
}
else
{
    /*what do we do if numMines is out of bounds?
    will display an error message*/
}
```

Para el mensaje de error, podemos dibujar un rectángulo rojo similar al de los botones, y escribir sobre él un texto de aviso, como por ejemplo *enter a number of mines between 1 and 50*.



Seleccionamos el rectángulo y el texto de aviso, y pulsamos **F8** para convertirlo en un **clip de película** llamado *alert*. Le damos el nombre de instancia *alert\_mc*.

Al inicio de la película, este clip debe estar invisible por lo que, junto con la definición de variables al inicio de la programación, escribiremos

```
alert_mc.visible = false;
```

Si *numMines* no está dentro de los límites queremos que se muestre el aviso. Además podemos añadir instrucciones para borrar el número erróneo que hemos introducido, así como para devolver el foco al campo de texto, ya que se habrá perdido al pulsar el botón *start*.

Para todo ello escribiremos entre las llaves del *else*

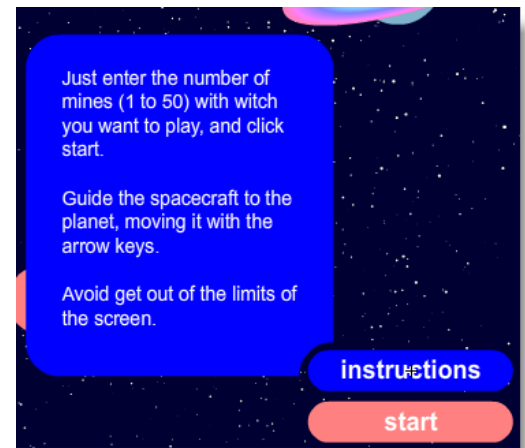
```
alert_mc.visible = true;
mines_txt.text = "";
stage.focus = mines_txt;
```

## Paso 21 de 22

En el caso de que `numMines` esté dentro del margen válido, la película se desarrollará en el fotograma *game* (fotograma 2), donde ya no se encuentra el aviso, por ello no es necesario indicar en este caso que `numMines` no esté visible.

Por último nos queda modificar la información que muestra el botón *instructions*, para que aparezca también información sobre las minas. Posiblemente tendremos que aumentar el tamaño del rectángulo sobre el que aparecen las instrucciones.

Con esto ya habremos completado la programación de nuestro juego, con el que hemos aprendido a programar muchas tareas habituales en los juegos, tales como controlar objetos con el teclado, detectar colisiones, añadir elementos desde la biblioteca, generar azar, etc.



## Paso 22 de 22

Para complementar los conceptos desarrollados en este tutorial, se recomienda hacer las siguientes actividades:

1. Permite al jugador seleccionar la velocidad de la nave y la vibración de las minas.
2. Crea niveles para el juego, de tal forma que al llegar al planeta se comience un nuevo juego con más minas, mayor velocidad y mayor vibración.
3. Añade un nuevo control para el movimiento de la nave, haciendo que la barra espaciadora detenga y reanude su movimiento.

