

Tutorial 9 - Creación de un juego (I)

Paso 1 de 26

En este tutorial vamos a crear la primera parte de un juego en el que tenemos que llevar una nave hasta un planeta, con ayuda de las flechas del teclado.

Comenzaremos creando los elementos gráficos necesarios para el juego, utilizando herramientas como el óvalo y rectángulo simple, y el pincel rociador.

En lo referente a programación, aprenderemos cómo controlar un objeto con el teclado, así como a detectar su posición o si choca con algún otro objeto.

También aprenderemos a crear botones que nos muestren información al situar el puntero sobre ellos.

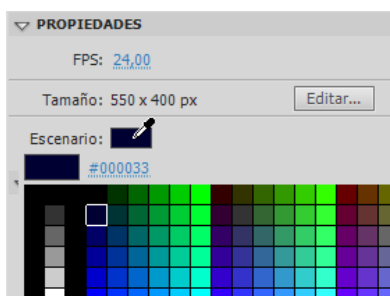
En el siguiente tutorial añadiremos complejidad a nuestro juego añadiendo obstáculos con movimientos aleatorios.



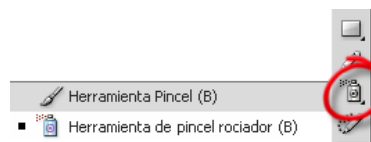
Paso 2 de 26

Creemos un nuevo **ActionScript 3.0** desde el área **Crear nuevo** de la pantalla de bienvenida o bien seleccionando **Archivo > Nuevo**. Guardamos el archivo como *tutorial9 fla*.

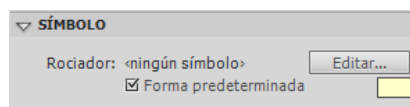
Cambiamos el **color de fondo** en el inspector de Propiedades. Le asignamos un color azul oscuro (#000033).



Vamos a crear en primer lugar los elementos gráficos de nuestro juego, comenzando por el fondo de estrellas. Para crear este fondo utilizaremos la **herramienta de pincel rociador** que se encuentra agrupada junto con la herramienta Pincel.



Esta herramienta emite de forma predeterminada un spray de puntos de partículas con el color que seleccionemos. Con la herramienta de pincel rociador seleccionada, cambiamos en el inspector de Propiedades el **color** de la partícula a un amarillo claro (#FFFFCC).



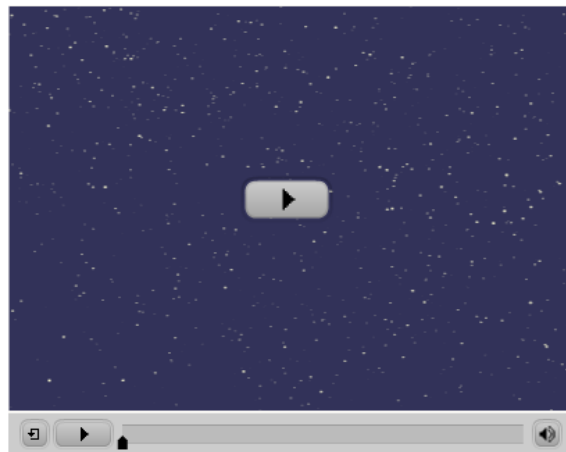
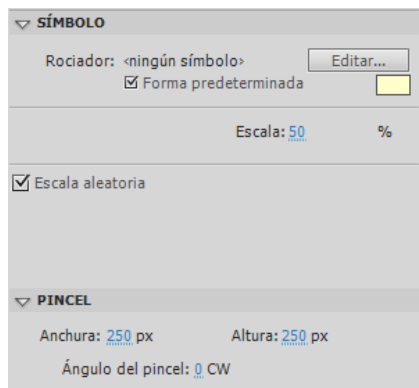
Con el botón *Editar* podríamos seleccionar como partícula emitida un símbolo que tuviéramos en la biblioteca. En este caso mantendremos la forma predeterminada.

Paso 3 de 26

Continuando la edición de las propiedades del pincel rociador, cambiamos la **Escala** a 50%. La escala se refiere al tamaño de las partículas emitidas.

Seleccionamos la casilla **Escala aleatoria**. Esto hace que las partículas sean de diferentes tamaños, aunque el tamaño máximo será el que hemos indicado previamente en la casilla *Escala*.

Por último, en el área *Pincel*, seleccionamos una **anchura** y **altura** de 250 px. Esto hará que las partículas estén más distanciadas entre sí.



Pulsamos y arrastramos sobre el escenario con el **pincel rociador** hasta conseguir un fondo estrellado que nos guste.

Damos a esta capa el nombre *stars*. Después bloqueamos la capa para evitar cambios accidentales.

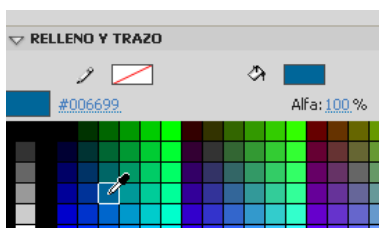
Paso 4 de 26

El siguiente paso va a ser la creación del planeta. Nuestro planeta estará formado por la parte interior y los anillos, es decir, que contendrá más de una capa. Para que todas las capas formen parte del mismo objeto, lo más cómodo es crearlas directamente dentro de un mismo símbolo.

Seleccionamos **Insertar > Nuevo símbolo**, le damos el nombre *planet* y como **Tipo** seleccionamos **Clip de película**.

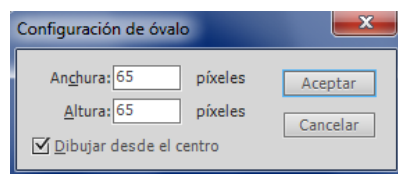
Dentro del símbolo *planet*, dibujaremos en primer lugar la parte interior del planeta. Para ello seleccionamos la herramienta **Óvalo**.

En el inspector de Propiedades **eliminamos el color del trazo** y seleccionamos como **color de relleno** el color #006699.



Con la herramienta **Óvalo** seleccionada, pulsamos la tecla **Alt** y, sin soltarla, hacemos clic sobre el escenario.

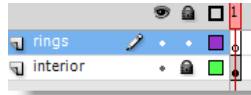
Se abrirá la pantalla **Configuración de óvalo**, que permite definir directamente el tamaño del óvalo. En este caso le vamos a asignar una **anchura** y **altura** de 65 px.



Seleccionamos el óvalo que hemos creado y lo centramos dentro de su propio escenario con ayuda del panel **Alinear**.

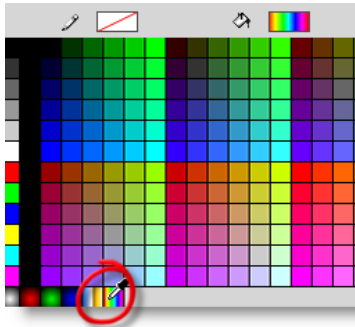
Paso 5 de 26

Llamamos *interior* a la capa en la que hemos creado el óvalo anterior, y añadimos una nueva capa a la que llamaremos *rings*.



Seleccionamos esta vez la herramienta **Óvalo simple** para poder modificar algunas características del óvalo después de dibujarlo.

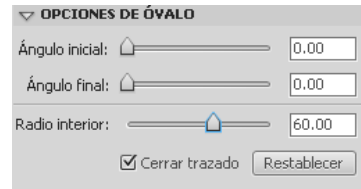
Esta vez tampoco seleccionaremos un trazo en el inspector de Propiedades. Como **color de relleno** seleccionaremos el **degradado multicolor** que aparece en la parte inferior de la paleta.



Al igual que en el paso anterior, pulsamos la tecla **Alt** y, sin soltarla, hacemos clic sobre cualquier punto del escenario. De momento no es necesario que el óvalo de los anillos esté centrado.

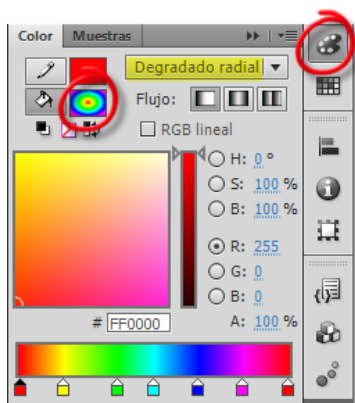
En la pantalla **Configuración de óvalo** asignamos una **anchura** y **altura** de **150 px**. Aceptamos para crear el óvalo.

En **Opciones de óvalo** establecemos un **Radio interior** de **60**. Recomendamos probar las opciones de ángulo inicial y final para conocer las posibilidades que esto ofrece, aunque finalmente dejaremos los ángulos a 0.

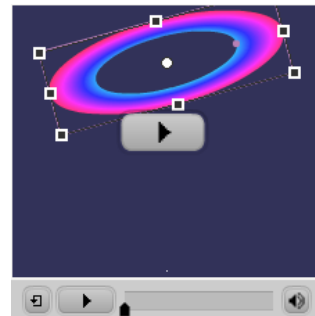


Paso 6 de 26

El relleno multicolor que hemos utilizado es por defecto un degradado lineal. Para convertir el degradado en radial, abrimos el panel **Color** y, con el óvalo de los anillos seleccionado, cambiamos el **Tipo de color** a **Degradado radial**.

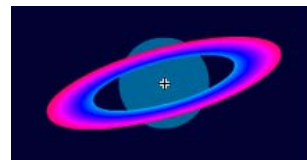
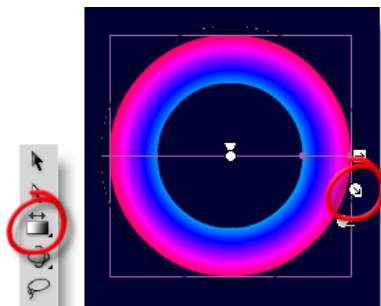


Con la herramienta **Transformación libre** modificamos la forma de los anillos, tal y como se muestra en el vídeo.



Centramos los anillos en el escenario con ayuda del panel **Alinear**. La parte interior del planeta deberá asomar ligeramente tanto por encima como por debajo de los anillos. De no ser así, haremos las rectificaciones pertinentes en los anillos con la herramienta transformación libre.

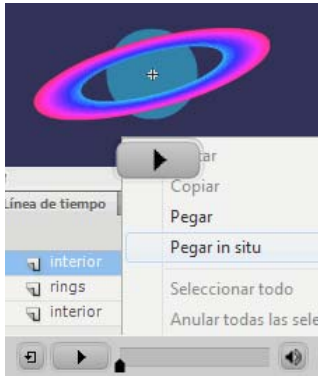
Tenemos varias formas de cambiar los colores de los anillos. Una de ellas es utilizar la herramienta **Transformación de degradado** y cambiar la amplitud del degradado para que se muestren diferentes colores.



Paso 7 de 26

La parte interior del planeta queda completamente por detrás de los anillos. Sin embargo, para que parezca que esa parte se encuentra dentro de los anillos, una parte del planeta debería verse por delante de los anillos, y otra parte por detrás.

Para conseguir este efecto, duplicaremos la parte interior del planeta en una nueva capa superior, a la que también podemos llamar *interior*.



Copiamos el óvalo de la parte interior del planeta haciendo clic sobre él con el **botón derecho** del ratón y seleccionando **Copiar**.

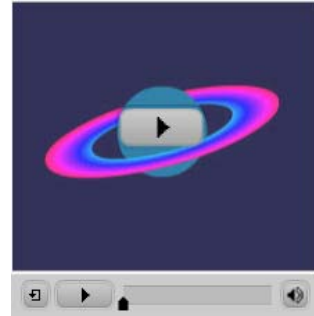
Bloqueamos las dos capas inferiores y, con la capa superior activa, hacemos clic con el **botón derecho** sobre el escenario y seleccionamos **Pegar in situ**.

Esto copiará el dibujo de la parte interior del planeta en la misma posición que tenía en la capa inferior.

Si ahora borramos de la capa superior una parte del planeta, podremos ver lo que se encuentra bajo esa parte, es decir, los anillos en primer lugar, y tras ellos la parte interior del planeta en la capa inferior.

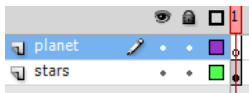
Probamos a eliminar la parte superior o inferior de la parte interior del planeta en la capa superior, según la perspectiva que queramos dar a nuestro planeta (visto desde arriba o visto desde abajo).

Observamos el vídeo para entender mejor lo que ocurre al borrar una zona del planeta en la capa superior. Elegimos la perspectiva que más nos guste.



Paso 8 de 26

Volviendo a la escena principal, creamos una nueva capa llamada *planet* por encima de la capa *stars*.



Con la capa *planet* activa, arrastramos desde la biblioteca una instancia del clip *planet*.

Situamos el clip en la parte superior derecha del escenario, y le damos el nombre de instancia *planet_mc*.



Creamos otra capa llamada *spacecraft* en la que situaremos la pequeña nave que controlará el usuario.

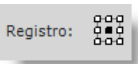
Esta nave tendrá una sola capa, así que podemos crearla en la capa *spacecraft* de la escena principal.

Dibujamos una nave de aspecto similar al de esta imagen. Para ello podemos utilizar herramientas como el rectángulo y la línea.



La imagen se encuentra muy ampliada. La altura total de la nave debe ser aproximadamente de 25 px, y la anchura unos 10 px.

Una vez dibujada, seleccionamos **Modificar > Convertir en símbolo** o pulsamos **F8**. Le damos al clip el nombre *spacecraft*, y como **Registro** le asignamos el **punto central**. Esto hará que el clip quede centrado en su propio escenario.



Situamos la nave en la parte inferior izquierda del escenario. En el inspector de Propiedades le asignamos el nombre *spacecraft_mc*.

Con estos dos clips ya podemos comenzar la programación de nuestro juego. Creamos una nueva capa a la que llamaremos *as*, y con el primer fotograma de esta capa seleccionado, pulsamos **Ventana > Acciones** o **F9**.

Paso 9 de 26

Queremos que la nave gire y se desplace al pulsar las flechas del teclado, así que el primer paso será convertir la nave en un detector de eventos del teclado.

Programamos que cuando se detecte una pulsación de tecla, se ejecute una función a la que hemos llamado `arrows`. La forma de añadir un evento de teclado es similar a lo que hemos visto hasta ahora.

```
spacecraft_mc.addEventListener(KeyboardEvent.KEY_DOWN
, arrows);

function arrows(e:KeyboardEvent):void
{
    trace("He pressed a key");
}
```

Con esta programación le estamos diciendo a la nave que, cuando detecte que se ha pulsado una tecla, ejecute una función que se llama `arrows`. Esta función, que recibe un evento de teclado, tiene a su vez la orden de escribir en el panel Salida la frase `I pressed a key`.

Sin embargo, si probamos la película y pulsamos las flechas del teclado no ocurrirá nada.

Esto se debe a que para que funcione un detector de eventos de teclado, el objeto encargado de detectarlo debe recibir el foco de la película, es decir, debe estar seleccionado. Para seleccionarlo añadimos al inicio de la programación el siguiente código:

```
stage.focus = spacecraft_mc;
```

Ahora que la nave tiene el foco, detectará las pulsaciones de nuestro teclado, escribiendo tras cada pulsación la frase que habíamos puesto en el `trace`. Cuando estamos en fase de pruebas, es posible que esta detección no funcione con las teclas que tengamos en Flash como métodos abreviados de teclado. Por ello en la fase de pruebas nos limitaremos también a las flechas del teclado.

Al tener ahora la nave el foco, podemos ver que aparece un rectángulo alrededor de la nave para indicar que tiene el foco. Para que no aparezca este rectángulo añadiremos, antes de dar el foco a la nave, una sentencia que especifique que, aunque nuestra nave tenga el foco, no muestre un rectángulo. Las dos primeras líneas de nuestra programación quedarán así:

```
spacecraft_mc.focusRect = false;
stage.focus = spacecraft_mc;
```

Paso 10 de 26

Ahora que hemos probado que el `trace` funciona, y que la nave no muestra el rectángulo del foco, procederemos a sustituir el `trace` por las sentencias que queremos ejecutar.

Queremos evaluar qué flecha se ha pulsado, y ejecutar acciones diferentes según sea el caso de haber pulsado una flecha u otra. Para este tipo de problemas es muy útil una sentencia llamada `switch`.

Para comprender esta sentencia, supongamos el caso de un test con respuestas a, b y c:

```
switch (answer)
{
    case a :
        trace("the user has chosen a");
        break;

    case b :
        trace("the user has chosen b");
        break;

    case c :
        trace("the user has chosen c");
        break;

    default :
        trace("ha ocurrido un error");
}
```

Lo que haría en este caso la sentencia `switch` es evaluar `answer` (variable que está entre paréntesis).

En el caso de que la respuesta fuera por ejemplo b, se analizarán los casos hasta encontrar uno que sea correcto (`case b`), y entonces se ejecutarán la sentencias que se encuentren dentro de ese `case` (en este caso un `trace` que informa de que se ha respondido b), y ya no se revisarán los demás casos (`break`).

Cada `case` debe terminar siempre con una sentencia `break` para que se omita el resto de la sentencia `switch` cuando encontremos un `case` correcto. Así evitamos que vaya comprobando el resto de los casos.

Si ningún `case` fuera correcto, entonces se ejecutará lo que se indique en `default`. Aquí las únicas respuestas posibles son a, b o c, por lo que si `answer` es cualquier otra cosa, hemos programado un `trace` que indique que ha habido un error. Aquí no haría falta un `break`.

Siempre es conveniente incluir `default`, aunque no preveamos que vaya a ocurrir nada fuera de los `case`.

Es importante no olvidar que esta sentencia tiene también sus propias llaves al inicio y al final.

Paso 11 de 26

Vamos a evaluar qué tecla ha detectado nuestro *listener* que se ha pulsado (`e.keyCode`). Según la tecla pulsada, rotaremos el clip hacia un lado o hacia otro. Si se ha pulsado cualquier otra tecla, no haremos nada, salvo salir de la sentencia.

Nuestra sentencia, que estará dentro de la función `arrows` (sustituyendo al `trace`), quedará como sigue:

```
switch (e.keyCode)
{
    case Keyboard.RIGHT :
        e.target.rotation = 90;
        break;

    case Keyboard.LEFT :
        e.target.rotation = -90;
        break;

    case Keyboard.UP :
        e.target.rotation = 0;
        break;

    case Keyboard.DOWN :
        e.target.rotation = 180;
        break;

    default :
        break;
}
```

La rotación se mide en grados (360 grados sería una vuelta completa). El clip rotará sobre su punto de registro, que este caso está en el centro del clip.

Probamos la película (**Ctrl+Intro**). La nave ya se orienta hacia un lado u otro dependiendo de la flecha que hayamos pulsado.

El paso siguiente será añadir sentencias para que la nave, además de rotar, se desplace. Para conseguirlo podemos utilizar sentencias similares a las que utilizamos para desplazar la nube en el tutorial 6.

Por ejemplo, podemos añadir las siguientes sentencias en cada `case`, según corresponda:

```
e.target.x += 3; //3px right
e.target.x -= 3; //3px left
e.target.y -= 3; //3px up
e.target.y += 3; //3px down
```

El primer `case` quedaría por tanto de la siguiente forma:

```
case Keyboard.RIGHT :
    e.target.rotation = 90;
    e.target.x += 3;
    break;
```

Paso 12 de 26

Completamos los cuatro `case` con las sentencias correspondientes y probamos de nuevo la película. Es conveniente probar la película con cada pequeño paso que hagamos, para poder comprobar qué hace exactamente cada parte del código que vamos añadiendo.

La nave ya se mueve hacia los cuatro lados, dependiendo de la flecha que pulsemos. Sin embargo, el movimiento no es muy fluido.

Si queremos que al pulsar una flecha, el movimiento hacia ese lado se mantenga de forma continua hasta pulsar otra flecha, entonces necesitaremos añadir un `ENTER_FRAME`.

Añadimos a la nave un detector del evento `ENTER_FRAME`, que llame a una función a la que llamaremos `moveSpacecraft`:

```
spacecraft_mc.addEventListener(Event.ENTER_FRAME,
moveSpacecraft);
```

La función `moveSpacecraft` tiene que detectar el último movimiento que se ha activado, y continuar en la misma dirección. Por ejemplo, en el caso de que el movimiento sea hacia la derecha, entonces continuar avanzando hacia la derecha.

Vamos a crear una variable llamada `course`, que almacene un texto con la dirección del movimiento.

Las variables se crean con la siguiente estructura (asignar un valor inicial es opcional):

```
var identifier:DataType = value;
```

Por legibilidad del código, es conveniente declarar las variables que vayamos a utilizar en la parte superior del código. Por lo tanto, añadiremos esta primera línea a nuestro código:

```
var course:String = "";
```

El tipo de variable `String` representa cadenas de caracteres. Las cadenas de caracteres siempre van entre comillas.

Como valor inicial hemos creado una cadena vacía (sin ningún contenido).

Paso 13 de 26

Como ahora queremos que los desplazamientos de la nave los haga otra función que recibe un `ENTER_FRAME` (la función `moveSpacecraft`), en la función `arrows` sólo guardaremos el dato sobre la rotación y hacia dónde se debe dirigir el movimiento.

Por lo tanto, sustituimos las sentencias `case` del desplazamiento en la función `arrows` de la siguiente manera:

```
case Keyboard.RIGHT :
    e.target.rotation = 90;
    course = "right";
    break;
```

En los demás casos, asignamos a la variable `course` los valores "left", "up" y "down" respectivamente.

La función `moveSpacecraft`, que crearemos a continuación, moverá de forma continua el clip hacia un lado u otro dependiendo del último valor asignado a la variable `course`, ya que se ejecutará continuamente al ser llamada por un evento `ENTER_FRAME`.

La función `moveSpacecraft` evaluará el contenido de la variable `course`, y en base a eso moverá la nave.

```
function moveSpacecraft(e:Event):void
{
    switch (course)
    {
        case "right" :
            e.target.x += 3;
            break;

        case "left" :
            e.target.x -= 3;
            break;

        case "up" :
            e.target.y -= 3;
            break;

        case "down" :
            e.target.y += 3;
            break;

        default :
            break;
    }
}
```

Paso 14 de 26

El valor 3 es la cantidad de píxeles que se desplaza la nave en cada `ENTER_FRAME`. Es, por tanto, la velocidad de la nave. En vez de escribir este valor, podríamos por tanto crear una variable llamada `speed`, que especifique el número de píxeles en que varía la posición de la nave en cada momento.

Podemos crear esta variable en la parte superior de la programación, tras la declaración de la variable `course`:

```
var course:String = "";
var speed:Number = 3;
```

Dentro de la función `moveSpacecraft`, sustituimos el número de píxeles por la variable `speed` en los cuatro `case`, que tomará el valor que le hemos asignado cuando creamos la variable:

```
case "right" :
    e.target.x += speed;
    break;
```

De esta forma, si cambiamos el valor de la variable `speed`, cambiará la velocidad de los desplazamientos en las cuatro direcciones. Podemos probar diferentes valores hasta que nos parezca una velocidad adecuada.

El siguiente paso va a ser detectar si la nave llega al planeta, que será la meta del juego. Para ello, mientras la nave se desplaza debemos ir comprobando si choca con el planeta.

Dentro de la función `moveSpacecraft`, por debajo de la llave de cierre de la sentencia `switch`, añadiremos este código:

```
if (e.target.hitTestObject(planet_mc))
{
    trace("Targed achieved. I win.");
}
```

Esto es un condicional que comprueba si la nave (`e.target`) choca (`hitTestObject`) con el planeta (`planet_mc`). Si choca, entonces se mostrará `Targed achieved. I win.` en el panel Salida.

Vamos a añadir otro condicional, esta vez para detectar si la nave se sale fuera de los límites del escenario, que mide 550 x 400. Por lo tanto deberemos comprobar si la propiedad `x` (posición horizontal) de la nave tiene un valor inferior a 0 o superior a 550, y si la propiedad `y` (posición vertical) tiene un valor por debajo de 0 o por encima de 400. En cualquiera de los cuatro casos, la nave estará posicionada fuera del escenario.

Paso 15 de 26

Este nuevo condicional, escrito también dentro de la función `moveSpacecraft`, quedará de la siguiente manera (las barras verticales `|`, que se escriben con `AltGr + 1`, equivalen al OR lógico):

```
if (e.target.x < 0 || e.target.x > 550 || e.target.y < 0 || e.target.y > 400)
{
    trace("I'm outside the stage. I lost.");
}
```

Probamos la película para comprobar su correcto funcionamiento. La nave se desplazará por el escenario dependiendo de las flechas que pulsemos en el teclado. Si llegamos al planeta, aparecerá la frase de que hemos ganado en el panel Salida, y si estamos fuera de los límites del escenario, aparecerá la frase de que hemos perdido.

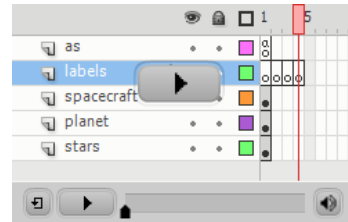
Ahora el juego comienza directamente, pero sería adecuado disponer de una pantalla previa con las instrucciones del juego, y que al llegar al planeta (ganar) o salirnos del escenario (perder) se mostrara una pantalla diferente y el juego terminara, dando opción a volver a jugar.

Vamos por tanto a crear cuatro fotogramas diferentes:

- Un fotograma inicial con dos botones, uno con las instrucciones del juego y otro para comenzar a jugar.
- Un segundo fotograma en el que se se desarrollará el juego.
- Un tercer fotograma indicando que hemos ganado.
- Un último fotograma indicando que hemos perdido.

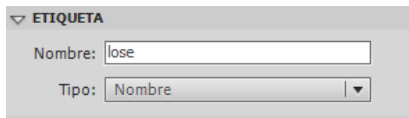
Estos dos últimos fotogramas tendrán un botón para volver a jugar.

Insertamos una nueva capa a la que llamaremos *labels*. Insertamos **fotogramas clave** en los fotogramas 2, 3 y 4. Podemos crear estos fotogramas clave fácilmente si hacemos clic sobre cada fotograma y pulsamos la tecla **F6**.



Paso 16 de 26

Pulsamos sobre cada fotograma de la capa *labels*, y asignamos a cada uno un **nombre de etiqueta** en el inspector de **Propiedades**. Al fotograma número 1 le llamaremos *start*, al 2 *game*, al 3 *win*, y al 4 *lose*.

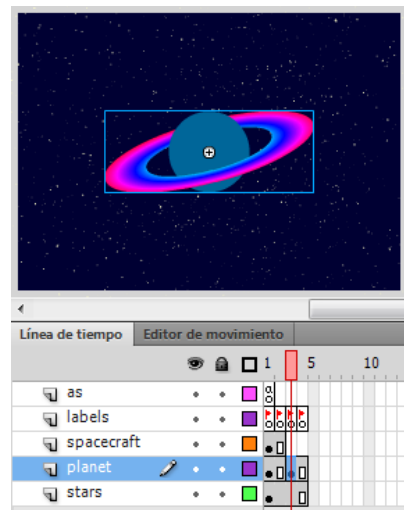
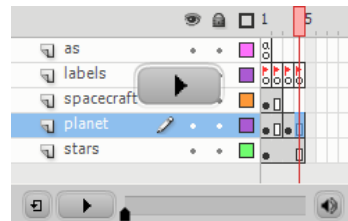


Los fotogramas que tengan una etiqueta mostrarán una pequeña bandera roja en la línea de tiempo.

La nave y el planeta tendrán que mantenerse visibles y sin cambios en los fotogramas *start* y *game*, mientras que las estrellas se verán como fondo en los 4 fotogramas.

En los fotogramas *win* y *lose* utilizaremos de nuevo una imagen del planeta, pero esta vez lo situaremos a tamaño mayor y en el centro de la pantalla.

Por lo tanto insertamos los fotogramas correspondientes pulsando **F5**, y después insertaremos un fotograma clave con **F6** en el tercer fotograma de la capa *planet*, donde haremos un cambio en el tamaño y posición del planeta que se mantendrá en los fotogramas 3 y 4.



Paso 17 de 26

Cambiamos la posición y tamaño del planeta en el fotograma 3, posicionándolo en el centro del escenario y a mayor tamaño.

Creamos una **nueva capa** llamada *text*.

En el fotograma 3 de esta nueva capa creamos un **fotograma clave** (F6).

Con la herramienta **Texto** creamos en este fotograma un campo de texto clásico y estático con el texto *you won!!!*.

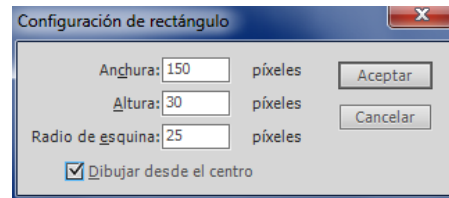
En el fotograma 4 de la capa *text* creamos un nuevo fotograma clave (F6), y cambiamos el texto a *you lost!!!*



Creamos una **nueva capa** llamada *buttons*.

Para crear un rectángulo que después será el fondo del botón, seleccionamos la herramienta **Rectángulo, sin trazo** y con color de **relleno rojo**.

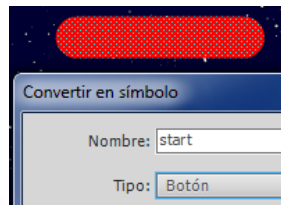
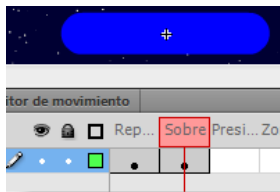
Con la tecla **Alt** pulsada hacemos clic sobre el escenario. En la configuración del rectángulo, establecemos una **anchura** de 150 px, **altura** de 30 px, y **radio de esquina** de 25 px, para que aparezcan las esquinas redondeadas.



Paso 18 de 26

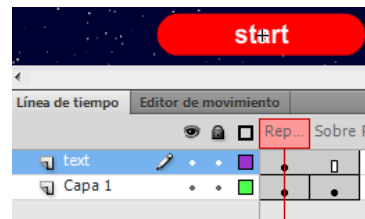
Seleccionamos el rectángulo rojo que hemos creado, y pulsamos **F8** para convertirlo en **símbolo**.

Le damos el nombre *start*, y seleccionamos que sea tipo **Botón**.



Añadimos una nueva capa sobre el fondo del botón, en la que situaremos el texto del botón. Con la herramienta **Texto** creamos un campo de texto clásico estático, y escribiremos *start* en color blanco. Centramos el campo de texto en el rectángulo de fondo.

Este texto se mantendrá tanto en el fotograma *Reposo* como en el fotograma *Sobre*.



Hacemos doble clic sobre el botón en el escenario para editarlo.

Insertamos un **fotograma clave** pulsando **F6** en el estado *Sobre*, y cambiamos el **color** del rectángulo a **azul** en el inspector de Propiedades.

Volvemos a la escena principal, y situamos el botón en la parte inferior derecha del escenario. En el inspector de Propiedades le asignamos el **nombre de instancia** *start_btn*.

Paso 19 de 26

Creemos un nuevo botón llamado *instructions* con las mismas características que el anterior, pero esta vez con el texto *instructions*. Lo situamos en el escenario sobre el botón *start*.

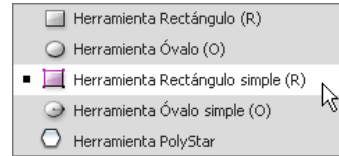


Vamos a hacer que cuando situemos el puntero sobre el botón *instructions*, aparezcan directamente las instrucciones del juego. La forma más sencilla de conseguir este efecto es añadiendo el contenido que queremos mostrar en el estado *Sobre* del botón.

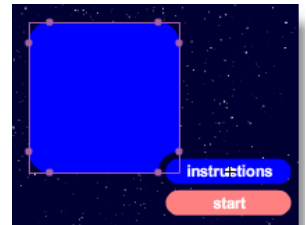
Hacemos doble clic sobre el botón *instructions* para editarlo y añadir algunos cambios a su fotograma *Sobre*.

Añadimos una **nueva capa** en la línea de tiempo del botón, e insertamos un **fotograma clave** en el estado *Sobre*.

Seleccionamos la herramienta **Rectángulo simple**, para poder efectuar modificaciones en las características del rectángulo después de dibujarlo.



En el fotograma *Sobre* de esta nueva capa que hemos creado, dibujamos un rectángulo con el mismo color azul que habíamos utilizado para el botón.

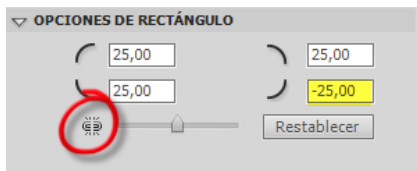


Paso 20 de 26

Con el rectángulo recién creado seleccionado en el escenario, cambiamos el valor de sus esquinas en el inspector de Propiedades.

Para poder asignar un valor diferente a una esquina, debemos en primer lugar desactivar el bloqueo del radio de las esquinas.

Después asignamos un valor de 25 a todas las esquinas excepto a la esquina inferior derecha, a la que asignaremos un valor de -25. De esta forma conseguiremos una esquina con el radio inverso.



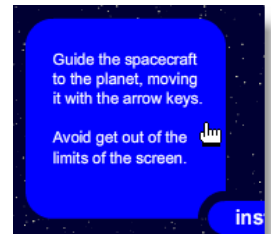
Creemos otra **capa** por encima de este rectángulo para escribir el texto de las instrucciones. Insertamos un **fotograma clave** en el estado *Sobre* de esta capa.

El **texto**, en color **blanco** para destacar sobre el recuadro azul, podría ser algo similar a lo siguiente: *Guide the spacecraft to the planet, moving it with the arrow keys. Avoid get out of the limits of the screen.*

Volviendo a la **escena principal**, seleccionamos **Control > Habilitar botones simples** para poder comprobar el efecto de situar el puntero sobre un botón sin necesidad de probar la película.

Podemos comprobar que si situamos el puntero sobre el área donde se muestran las instrucciones, el botón también se activa.

Sin embargo, queremos que sólo se muestren las instrucciones si nos situamos sobre el rectángulo rojo del botón *instructions* en su estado de reposo.

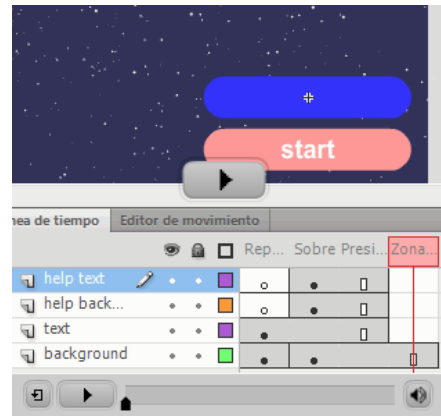
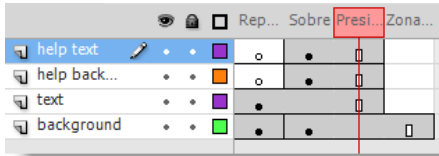


Paso 21 de 26

Seleccionamos de nuevo **Control > Habilitar botones simples** para desactivar esta vez los botones, y así poder hacer **doble clic** sobre el botón *instructions* para editarlo.

Insertamos **fotogramas (F5)** de tal forma que en la *zona activa* se muestre el fondo del botón, es decir, el rectángulo redondeado que servía de base al botón. Esa será el área de clic del botón.

Insertamos fotogramas en el resto de las capas para que se muestren también en el estado *Presionado*, y así evitar que en ese estado se muestre sólo el fotograma del fondo del botón.



Si ahora habilitamos de nuevo los botones simples para hacer la prueba, veremos que el botón se comporta de la manera esperada.

Esta forma de utilizar el estado *Sobre* de un botón, también llamado *rollover*, puede resultar muy útil para crear sencillas aplicaciones educativas, ya que podemos mostrar diferentes tipos de información al situarnos sobre un área concreta.

Paso 22 de 26

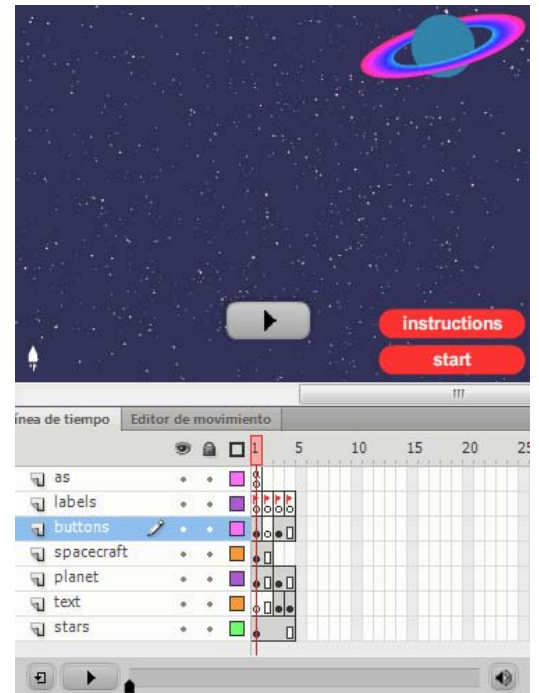
Volvemos a la escena principal. Los botones *instructions* y *start* sólo deben verse en el primer fotograma, ya que durante el juego no deben estar visibles.

Creamos el botón que nos falta, al que llamaremos *replay*. Será un botón igual que *start*, salvo que en el texto escribiremos *play again*. Este botón deberá estar visible en los fotogramas *win* y *lose* (3 y 4).

Colocamos por tanto este tercer botón en el tercer fotograma de la capa *buttons*, en la misma posición que tenía el botón *start* en el primer fotograma y le damos el nombre de instancia *replay_btn*. Al botón *instructions* no le dimos nombre de instancia porque no va a llevar asociada ninguna programación.

Insertamos un fotograma en blanco en el segundo fotograma (en el que se desarrolla el juego), para que en ese fotograma no se muestre ningún botón.

Repasando el contenido que hemos creado, en el primer fotograma (*start*) se verán, además de la nave y el planeta, los botones *instructions* y *start*. En el segundo fotograma (*game*) sólo veremos la nave y el planeta. En el tercer fotograma (*win*) y en el cuarto fotograma (*lose*), veremos el botón *replay*, así como una imagen ampliada del planeta en el centro del escenario, con diferentes textos según sea el caso de ganar o perder. Las estrellas serán el fondo común de los cuatro fotogramas.



Paso 23 de 26

Ahora que tenemos los botones creados, necesitamos programar su funcionamiento. Volvemos al fotograma 1 de la capa *as* y pulsamos **F9** para abrir el panel **Acciones**.

En primer lugar, al iniciar nuestra película queremos que se detenga en el primer fotograma, por lo que incluiremos la instrucción `stop()`. Podemos colocar esta instrucción por ejemplo en la primera línea de nuestra programación.

El movimiento de la nave ahora no debe permitirse directamente, sino que debe ser activado cuando pulsemos el botón *start_btn*. Por otro lado, el botón *start_btn* también debe llevarnos al fotograma 2, etiquetado como *game*, en el que no se muestra ningún botón.

Añadimos por tanto un listener al botón *start_btn*, de tal forma que ejecute una función a la que llamaremos `playGame` cuando hagamos clic sobre él.

```
start_btn.addEventListener(MouseEvent.CLICK,
playGame);
```

Las sentencias que habíamos creado para seleccionar (poner el foco) y añadir los listeners a la nave deben estar ahora dentro de la función `playGame`.

La función `playGame` quedará por tanto de la siguiente manera:

```
function playGame(e:MouseEvent):void
{
    gotoAndStop("game");
    spacecraft_mc.focusRect = false;
    stage.focus = spacecraft_mc;

    spacecraft_mc.addEventListener(KeyboardEvent.KEY_DOWN
, arrows);

    spacecraft_mc.addEventListener(Event.ENTER_FRAME,
moveSpacecraft);
}
```

Para el botón *instructions* no hay programación, ya que su función es mostrar la ayuda cuando situamos el puntero sobre el botón, y eso lo hemos conseguido mostrando la ayuda en el estado *Sobre* del botón.

Cambiamos ahora las sentencias `trace` de la función `moveSpacecraft` por las sentencias siguientes, según corresponda:

```
gotoAndStop("win");
gotoAndStop("lose");
```

Si no hubiéramos puesto etiquetas en los fotogramas, las instrucciones serían `gotoAndStop(3)` en un caso y `gotoAndStop(4)` en el otro caso.

Paso 24 de 26

Aunque todavía nos falta por programar el botón para volver a jugar, si probamos ahora la película y llegamos al planeta, comprobaremos que después de mostrar el fotograma de que hemos ganado, al poco tiempo se mostrará el fotograma de que hemos perdido.

Esto se debe a que, pese a que no vemos la nave en los fotogramas 3 y 4, no hemos eliminado los listeners, por lo que se siguen calculando posiciones para la nave. Es decir, que después de llegar al planeta, la posición de la nave saldrá por un extremo del escenario, dando lugar a que veamos el fotograma de que hemos perdido.

Para solucionar este problema, antes del `gotoAndStop` que envía la cabeza lectora a los fotogramas *win* o *lose*, tenemos que eliminar los listeners que habíamos creado para la nave.

```
spacecraft_mc.removeEventListener(KeyboardEvent.KEY_D
OWN, arrows);
spacecraft_mc.removeEventListener(Event.ENTER_FRAME,
moveSpacecraft);
```

Después de la sentencia del `gotoAndStop`, que enviará la cabeza lectora a los fotogramas del final, podemos añadir el listener para habilitar el botón *replay*, ya que es entonces cuando el botón aparecerá en escena:

```
replay_btn.addEventListener(MouseEvent.CLICK,
replay);
```

Por tanto, los pasos que ocurrirán tanto si ganamos como si perdemos, será remover en primer lugar los listeners de la nave, después ir al fotograma *win* o *lose*, según sea el caso, y por último añadir un listener al botón *replay*.

Salvo el nombre del fotograma al que dirigimos, las instrucciones que damos son las mismas en ambos casos (ganar o perder). Para no repetir la programación en dos lugares diferentes (en los dos condicionales `if` dentro de `moveSpacecraft`), lo más adecuado en estos casos es crear una función independiente que contenga estas sentencias.

Para solucionar el hecho de que haya que ir a un fotograma diferente en cada caso, podemos añadir un parámetro a la función, de tal forma que reciba el nombre del fotograma al que tiene que ir.

Así podríamos incluir en la llamada a la función, a la que llamaremos `gameOver`, el nombre del fotograma de la siguiente manera (lo mismo pero con *lose* en el otro caso):

```
if (e.target.hitTestObject(planet_mc))
{
    gameOver("win");
}
```

Paso 25 de 26

La función, que recibe un parámetro con el nombre del fotograma, tendrá esta definición:

```
function gameOver(frameLabel:String):void
{

spacecraft_mc.removeEventListener(KeyboardEvent.KEY_DOWN, arrows);

spacecraft_mc.removeEventListener(Event.ENTER_FRAME,
moveSpacecraft);
    gotoAndStop(frameLabel);
    replay_btn.addEventListener(MouseEvent.CLICK,
replay);
}
```

Como vemos, al parámetro que recibe la función le hemos llamado `frameLabel`, y a la sentencia `gotoAndStop` le indicamos que vaya a un fotograma con el valor del parámetro recibido (en nuestro caso los valores recibidos son "win" o "lose").

La función `replay`, que será llamada cuando pulsemos el botón `replay_btn`, tan sólo indicará que volvamos al fotograma inicial:

```
function replay(e:MouseEvent):void
{
    gotoAndStop("start");
}
```

Resumiendo la programación creada en la primera parte del juego, tenemos en primer lugar un `stop` y la creación de las variables `course` y `speed`.

Después añadimos un listener al botón `start_btn`.

Por último tenemos creadas cinco funciones, que serán llamadas en diferentes momentos. Las funciones que hemos creado son:

- `playGame`, que activa el movimiento de la nave.
- `arrows`, que rota la nave e indica la dirección del movimiento.
- `moveSpacecraft`, que desplaza la nave y evalúa si ha llegado al planeta o ha salido fuera del escenario.
- `gameOver`, que borra los listeners, va a la pantalla final y prepara el botón para volver a jugar.
- `replay`, que va al fotograma inicial del juego.

En el siguiente tutorial añadiremos complejidad al juego añadiendo obstáculos que se moverán de forma aleatoria por el escenario.

Paso 26 de 26

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

1. Crea en un nuevo documento un dibujo de un cuerpo humano utilizando botones, de tal forma que cuando situemos el puntero sobre cada área del cuerpo, el área quede resaltada y se muestre un recuadro con información sobre esa área.
2. Crea un nuevo juego con diferentes niveles, en el que podamos mover un objeto con el teclado y al llegar a un objetivo pasemos a otra pantalla. En cada pantalla nueva aumentamos la velocidad.

