

Tutorial 6 - Animación con ActionScript 3.0

Paso 1 de 19

En este tutorial vamos a hacer una primera aproximación a la programación con ActionScript 3.0, la última versión del lenguaje de programación de Flash.

Con este lenguaje de programación podemos añadir interactividad a nuestras películas, controlar el aspecto o movimiento de objetos en el escenario, crear aplicaciones complejas, etc.

En primer lugar vamos a aprender a controlar los clips que se encuentran en el escenario mediante programación. En este caso controlaremos un clip realizado a partir de la nube que creamos en el tutorial 1.



Paso 2 de 19

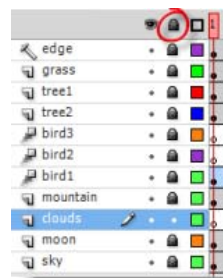
Abrimos el documento *tutorial1.fla*. Hacemos clic sobre la nube con el **botón derecho** del ratón y seleccionamos **Copiar**. Si no podemos seleccionarla será posiblemente porque tenemos bloqueada la capa *cloud* en la línea de tiempo, así que tendremos que desbloquearla previamente.



Una vez copiada, ya podemos cerrar el documento sin necesidad de guardar los cambios.

Abrimos el archivo *tutorial5.fla*, que contiene la animación de los pájaros, y lo guardamos como *tutorial6.fla*.

Creamos una nueva capa por detrás de la montaña y le damos el nombre *clouds*.



Bloqueamos el resto de las capas para evitar posibles modificaciones accidentales. La forma más rápida es hacer clic sobre el candado que está sobre todas las capas, y después desbloquear la capa que nos interesa.

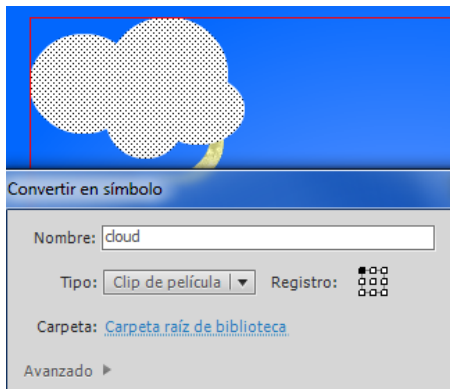
Con la capa *clouds* activa, seleccionamos cualquier punto del escenario con el **botón derecho** del ratón y seleccionamos **Pegar**.

Situamos la nube en la esquina superior izquierda del escenario.



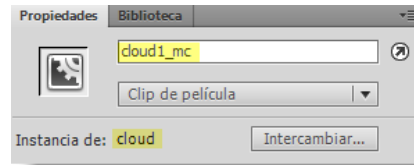
Paso 3 de 19

Con la nube seleccionada, pulsamos **F8** o **Modificar > Convertir en símbolo**. Le damos el nombre *cloud* y como tipo seleccionamos **Clip de película**. Situamos el punto de registro en el extremo superior izquierdo.



Ahora tendremos un clip de película llamado *cloud* en la biblioteca, mientras que en el escenario tendremos una instancia de ese clip de película, tal y como nos indicará la parte superior del inspector de **Propiedades**.

Para poder controlar una instancia con ActionScript, lo primero que tenemos que hacer es darle un nombre de instancia en el inspector de **Propiedades**. Le vamos a dar el nombre *cloud1_mc*.



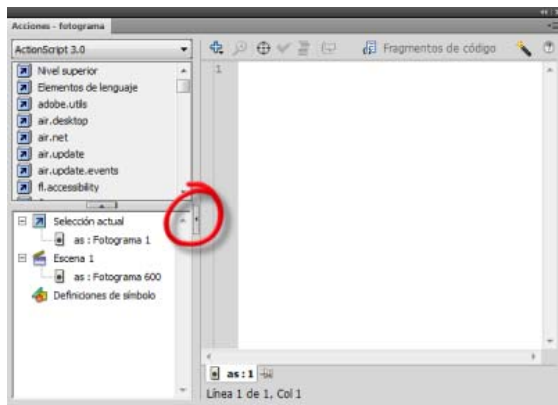
Por convención los nombres de instancia generalmente comienzan por una letra minúscula y terminan de la siguiente manera:

- *_mc* para clips de película (movie clip)
- *_btn* para botones
- *_txt* para campos de texto

Paso 4 de 19

Vamos a escribir las acciones en una **nueva capa** que situaremos por encima de todas las demás, y a la que llamaremos *as* (por ActionScript). Es conveniente tener todas las acciones separadas en una capa para facilitar la organización del documento.

Abrimos el panel **Acciones** pulsando **F9** o **Ventana > Acciones**.



En la columna izquierda de este panel tenemos en primer lugar un acceso exhaustivo a las diferentes clases y funciones disponibles. En nuestro caso no vamos a utilizar esta forma de introducir código.

En la parte inferior de la misma columna tenemos información sobre la selección actual (en nuestro caso el fotograma 1 de la capa *as*). Cuando tengamos acciones distribuidas en más fotogramas, desde aquí tendremos un acceso cómodo y directo que nos permitirá acceder a la programación de los diferentes fotogramas.

En la parte central tenemos el editor, que es el área sobre la que escribiremos el código. Sobre esta área tenemos algunas herramientas para comprobar el código, insertar comentarios, acceder a la ayuda, etc. Recomendamos tener desactivado el asistente de script.

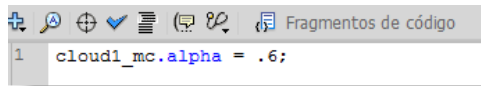
Para disponer de más espacio para escribir el código, recomendamos colapsar la columna izquierda haciendo clic sobre la pequeña flecha que separa ambas columnas. Podremos expandirla de nuevo cuando lo necesitemos.

Paso 5 de 19

Vamos a comenzar asignando un valor de `.6` a la propiedad `alpha`. Este valor significa una opacidad del 60%. Este valor puede oscilar entre 0 (transparente) y 1 (opaco).

Comenzamos escribiendo el nombre de la instancia seguido por un punto. Si después de escribir el punto pulsamos **Ctrl + Barra espaciadora** aparecerá automáticamente un desplegable con diferentes métodos y propiedades que podemos asignar.

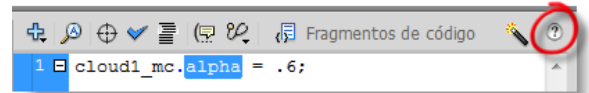
Tras el punto se escribe el método que queremos ejecutar o la propiedad que queremos asignar. En este caso hemos seleccionado la propiedad `alpha`. Las palabras clave del lenguaje aparecen en color azul.



Después, tras un signo `=` escribimos el valor para esa propiedad. En este caso hemos escrito `.6` (también podríamos haber escrito `0.6`). Los espacios antes y después del signo `=` no son necesarios, pero mejoran la legibilidad del código. Finalizamos cada sentencia con un punto y coma `;`.

Si tenemos problemas con el tamaño del texto del código, podemos abrir las **preferencias (Ctrl+U)** y, en la categoría de **ActionScript**, cambiar el tipo de texto, el tamaño de la letra o los colores. Es preferible mantener el coloreado del código, ya que nos ayuda a distinguir palabras clave, comentarios, etc.

Una forma rápida de acceder a la ayuda sobre alguna palabra clave es seleccionarla directamente en el editor y hacer clic sobre el icono de la ayuda. En la página de ayuda podremos ver definiciones y ejemplos de uso de cada palabra clave.



Paso 6 de 19

Si contraemos el panel **Acciones** pulsando sobre la barra gris superior del panel para ver el escenario, no observamos ningún cambio en la nube. Esto se debe a que la programación se ejecuta cuando probamos o publicamos la película (tiempo de ejecución), y no directamente en el escenario de edición (tiempo de edición).

Para comprobar cómo ha cambiado el alfa de la nube tendremos que probar la película con **Control > Probar película (Ctrl+Intro)**.

Si hemos asignado un valor para una propiedad en el código, este valor prevalecerá sobre lo que hayamos definido en tiempo de edición. Por ejemplo, en el caso del alfa, podemos asignar diferentes valores en el inspector de Propiedades, pero cuando probemos la película siempre veremos la nube con el 60% del alfa, ya que es el valor que hemos introducido en el código.

De la misma forma, si asignamos en el código valores para otras propiedades, como por ejemplo las posiciones X e Y del clip, éste aparecerá en las coordenadas que especifiquemos en la programación, y no en las coordenadas en que lo hayamos situado en el escenario.

Para añadir **interactividad** a una película, lo más probable es que necesitemos detectar cuándo ocurre un determinado **evento** (que se haya clicado sobre un botón, que se haya pulsado una tecla, etc.), y ejecutar después una **acción** en base a ello.

Para ello tenemos que agregar un *listener*, es decir, añadir a un objeto la capacidad de detectar si algo ha ocurrido, y que llame a una función cuando ello ocurra.

La forma general de crear un *listener* es la siguiente:

```
nameInstance.addEventListener(nameEvent, nameFunction);
```

En nuestro caso vamos a hacer que nuestra nube se arrastre cuando estemos pulsando sobre ella.

Para ello añadiremos la siguiente línea al código:

```
cloud1_mc.addEventListener(MouseEvent.CLICK, drag);
```

Con este código hemos añadido a nuestra instancia `cloud1_mc`, un detector para un evento del ratón (`MouseEvent`). En concreto, el evento del ratón que queremos que detecte es si el usuario ha pulsado sobre ella (`CLICK`). Si es así, ejecutará una función a la que hemos llamado `drag`, y que definiremos más adelante.

Paso 7 de 19

Un evento de ratón similar es `MouseEvent.CLICK`. La diferencia es que con `CLICK` se detecta si se ha pulsado y soltado el ratón, es decir, si se ha hecho clic. Más adelante veremos que en nuestro caso vamos a ejecutar acciones diferentes para las acciones de pulsar y soltar el ratón, y es por ello por lo que hemos elegido el evento `MOUSE_DOWN` (y después utilizaremos `MOUSE_UP`).

Ahora pasaremos a definir nuestra función `drag`. Una **función** es un bloque de instrucciones agrupadas, y que se ejecuta al ser llamada desde otra función o método.

Al escribir el código, las funciones y métodos se diferencian de las propiedades porque tras el nombre de los primeros hay un paréntesis. El paréntesis puede estar vacío, o bien puede recibir uno o varios parámetros separados por comas. Por ejemplo, `alpha` es una propiedad, mientras que `addEventListener` es un método con dos parámetros (evento y función).

Para crear una función tenemos que comenzar con la palabra clave `function`. Tras ella, escribimos el nombre que queremos dar a nuestra función. En este caso le damos el nombre `drag`. Tras el nombre, escribimos los paréntesis:

```
function drag()
```

Las funciones que vayan a ser llamadas desde un `addEventListener`, como es nuestro caso, reciben un **parámetro**. Este parámetro es un objeto del tipo de evento que ha desencadenado la función, en este caso `MouseEvent`.

Entre los paréntesis escribiremos el nombre del objeto (el nombre que queramos), dos puntos `:`, y el tipo de objeto.

Por lo tanto, de momento nuestra función comenzará así:

```
function drag(e:MouseEvent)
```

Le hemos dado al parámetro el nombre `e` por evento, pero podríamos haberle dado otro nombre. Generalmente se suele dar a este parámetro el nombre `e` o `event`.

Tras el nombre de la función es conveniente especificar el tipo de datos que devolverá. En nuestro caso, la función no va a devolver ningún resultado, sino que simplemente va a ejecutar una acción. En estos casos, como tipo de datos se asigna `void` (vacío).

Nuestra función queda como sigue:

```
function drag(e:MouseEvent):void
```

Paso 8 de 19

Ahora nos queda especificar las acciones que queremos que ejecute esta función. Todo el contenido de una función se escribe entre llaves, así que para asegurarnos de que no se nos olvidan, es conveniente escribir primero las llaves, y pasar después a escribir entre ellas las distintas instrucciones.

```
function drag(e:MouseEvent):void
{
    //your code here
}
```

Las dos barras que hemos añadido en el código anterior antes de `your code here`, indican que se trata de un comentario de una sola línea. Podemos añadir los comentarios que queramos en nuestro código, ya que no se ejecutarán. Para comentarios de varias líneas utilizaremos `/* y */` para delimitar el texto que queremos que sea un comentario.

Los comentarios pueden ser muy útiles no sólo para añadir información, sino también para hacer que algunas líneas del código no se ejecuten temporalmente, lo cual nos ayudará a comprobar el funcionamiento de nuestro código.

Probemos a escribir la función con el código siguiente:

```
function drag(e:MouseEvent):void
{
    trace("I'm clicking on the cloud");
}
```

La función `trace` mostrará en un panel llamado **Salida** lo que se encuentre dentro del paréntesis. Si escribimos entre comillas el contenido del `trace`, en el panel **Salida** se mostrará exactamente la frase que hayamos escrito. Las comillas permiten escribir cadenas de caracteres, y las veremos en color verde en el panel **Acciones**.

Los datos que aparezcan en este panel no los verá el usuario final, pero sí cuando probemos la película. Si aparece esa frase quiere decir que esa línea de código se está ejecutando.

Resumiendo: hemos añadido a la nube un detector del evento de pulsar sobre ella con el ratón, y cuando ese evento sea detectado, se ejecutará la función `drag`, que tiene la instrucción de mostrar en el panel **Salida** la frase `I'm clicking on the cloud`.

Probamos película con **Control > Probar película** y hacemos clic sobre la nube para comprobar el funcionamiento de nuestro código.

Paso 9 de 19

Ahora que hemos comprobado que nuestro código funciona correctamente, sustituimos la función `trace` por la acción que realmente queremos que se realice, que es que la nube se arrastre.

```
function drag(e:MouseEvent):void
{
    cloud1_mc.startDrag();
}
```

Si probamos de nuevo nuestra película, podremos ver que al pulsar sobre la nube, ésta comienza a arrastrarse junto con el ratón, y no deja de arrastrarse en ningún momento.

Para que cese el arrastre, tendremos que introducir una nueva función que permita que cuando se suelte el botón del ratón, la nube deje de arrastrarse.

El funcionamiento será el mismo que el anterior. Tendremos que crear otro detector para el evento de soltar el ratón, y asignarle una función que detenga el arrastre de la nube.

Por legibilidad del código podemos juntar por un lado las líneas con la creación de los listeners, y por otro lado las funciones.

La forma más rápida para escribir este nuevo código es copiar y pegar el que ya habíamos creado, ya que es en muchos aspectos similar. Sustituiremos el `MOUSE_DOWN` por `MOUSE_UP`, y el nombre de la función nueva será `drop` en vez de `drag`. Por último, la función que detiene un arrastre activo es `stopDrag()`.

El código que hemos creado hasta ahora en el primer fotograma de la capa `as` es el siguiente:

```
cloud1_mc.alpha = .6;

cloud1_mc.addEventListener(MouseEvent.CLICK, drag);
cloud1_mc.addEventListener(MouseEvent.CLICK, drop);

function drag(e:MouseEvent):void
{
    cloud1_mc.startDrag();
}

function drop(e:MouseEvent):void
{
    cloud1_mc.stopDrag();
}
```

Paso 10 de 19

Además del arrastre interactivo de la nube, vamos a añadir un desplazamiento continuo de la nube por el escenario mediante una función a la que llamaremos `wind`.

Existe un tipo de evento llamado `ENTER_FRAME`, que pertenece a una categoría genérica de eventos llamada `Event`, que se ejecuta cada vez que la cabeza lectora se desplaza un fotograma. Si la cabeza lectora está detenida, pero el objeto al que se asocia está en el escenario (esté o no visible), también se ejecutará con la misma frecuencia que los fotogramas por segundo que tengamos definidos.

En el caso de nuestra película, una función que se asocie a un evento `ENTER_FRAME` se ejecutará cada vez que avance la cabeza lectora, es decir, 24 veces por segundo. Cambiaremos ligeramente la posición de la nube cada 1/24 de segundo, por lo que mostrará una animación fluida.

Agregamos este nuevo listener bajo los otros dos que habíamos creado. Para ello escribimos este código:

```
cloud1_mc.addEventListener(Event.ENTER_FRAME, wind);
```

A falta de definir las instrucciones que ejecutará la función `wind`, su definición quedará como sigue:

```
function wind(e:Event):void
{
}
```

Como podemos ver, el evento `ENTER_FRAME` pertenece a una categoría llamada `Event`, y no a eventos de ratón (`MouseEvent`). Este `Event` aparecerá tanto en la creación del listener como en el parámetro de la función.

Para que la nube avance hacia la derecha tendremos que variar su posición `x` poco a poco dentro de la función `wind`. Por ejemplo, podemos especificar que la posición `x` de la nube aumente de uno en uno su valor, lo que traducido a código sería:

```
cloud1_mc.x = cloud1_mc.x + 1;
```

o bien, más sencillo:

```
cloud1_mc.x += 1;
```

que significa sumar su valor más el número que se encuentra tras el `=`, una unidad en este caso.

Paso 11 de 19

De momento tendremos la función `wind` como sigue:

```
function wind(e:Event):void
{
    cloud1_mc.x += 1;
}
```

Probamos la película para comprobar cómo se desplaza la nube hacia la derecha, mientras que sigue siendo arrastrable. Podemos probar con distintos valores hasta encontrar una velocidad que nos parezca adecuada (por ejemplo 0.4).

Si mantenemos pulsada la nube para arrastrarla pero no soltamos el ratón, la nube continuará avanzando hacia la derecha fuera del ratón. Aunque después soltemos el ratón, el evento `MOUSE_UP` ya no se producirá, ya que significa levantar el botón del ratón encima de la nube (no fuera de ella). De esta forma ya no podremos soltar el arrastre de la nube.

Para solucionarlo podemos asociar a la función `drop` otro evento de ratón llamado `ROLL_OUT`, que significa estar pulsando sobre la nube pero que el ratón se arrastre fuera de ella. Podemos crear el nuevo evento bajo el evento `MOUSE_UP`.

Por ahora los listeners que hemos creado serán los siguientes:

```
cloud1_mc.addEventListener(MouseEvent.CLICK, drag);
cloud1_mc.addEventListener(MouseEvent.CLICK, drop);
cloud1_mc.addEventListener(MouseEvent.CLICK, drop);
cloud1_mc.addEventListener(MouseEvent.CLICK, drop);
cloud1_mc.addEventListener(MouseEvent.CLICK, drop);
```

En este caso no necesitaremos crear una nueva función, ya que este listener llamará a la misma función `drop` que habíamos creado previamente.

Si probamos la película veremos que si en algún momento dejamos de arrastrar la nube, ésta terminará desapareciendo por la parte derecha del escenario. Aunque deje de visualizarse, la instancia seguirá ejecutando el código.

Lo siguiente que vamos a programar va a ser que cuando haya desaparecido por la parte derecha, vuelva a aparecer por la parte izquierda del escenario.

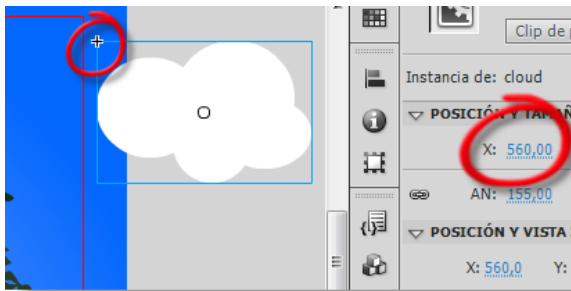
Para ello vamos a introducir una sentencia condicional dentro de la función `wind`, de tal forma que si se cumplen unos determinados requisitos, la nube se posicione en la parte izquierda del escenario.

Paso 12 de 19

En nuestro caso, vamos a definir que si la posición `x` de la nube alcanza determinado valor (cuando haya desaparecido por la derecha), entonces que vuelva a la parte izquierda del escenario.

Una forma rápida de hacer el cálculo de las posiciones de inicio y de fin es colocar la nube en un extremo y otro del escenario, y anotar el valor de `x` que aparece en el inspector de Propiedades.

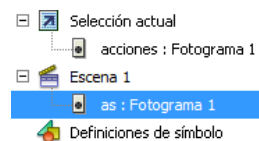
La posición del clip la marca una pequeña cruz. Esta posición depende del punto de registro que hayamos seleccionado al crear el clip.



En este caso, vamos a seleccionar como extremo derecho 560 (algo más que la anchura del escenario), y como extremo izquierdo -160 (una posición en la que todavía no se ve la nube). Estos valores dependerán del tamaño de vuestra nube y del punto de registro.

Podríamos hacer estos cálculos utilizando como datos la anchura del clip y la anchura del escenario. El motivo de no hacerlo de esta manera es que en próximos pasos añadiremos otra nube a la que programaremos profundidad en el eje `z`, y entonces la posición `x` del objeto ya no será equivalente a los píxeles del escenario, ya que se moverá en un escenario que simulará más profundidad, y por tanto más anchura que los 550 píxeles de nuestro escenario.

Colocamos la nube en el lugar en el que queramos que comience su animación y volvemos al panel **Acciones (F9)**. Recordemos que podemos navegar por las distintas acciones de nuestra película pulsando sobre la línea correspondiente en la parte inferior izquierda del panel.



En este caso, si no tuviéramos seleccionado el fotograma 1 de la capa `as` en la línea de tiempo, podríamos ir a él pulsando **as: Fotograma 1**.

Paso 13 de 19

La sentencia `if`, que es la que vamos a utilizar, evalúa si se cumple una condición, y en caso de cumplirse, ejecuta las instrucciones que indiquemos entre las llaves. La estructura es la siguiente:

```
if (condition)
{
    //statements
}
```

En la sentencia, por tanto, tendremos que especificar que si la posición `x` actual de la nube es mayor que 560 (el signo `>` significa mayor que), entonces que la posición pase a ser -160.

La función `wind` quedará de esta manera:

```
function wind(e:Event):void
{
    cloud1_mc.x += .4;

    if (cloud1_mc.x > 560)
    {
        cloud1_mc.x = -160;
    }
}
```

Vemos que la función tiene sus propias llaves, y que dentro de ella también hay un condicional con sus propias llaves de inicio y de fin.

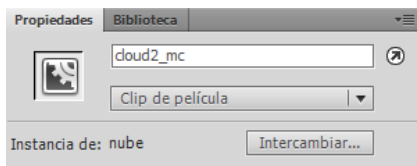
Probamos ahora la película con **Control > Probar película**. Vemos que al poco de desaparecer por el extremo derecho, aparece de nuevo por el extremo izquierdo.

A diferencia de la línea de tiempo principal, que tiene una extensión de 600 fotogramas, vemos que la animación del movimiento de la nube es independiente de esta extensión. En cada repetición del vuelo de los pájaros, la nube puede encontrarse en un punto diferente.

Para animar mediante programación es suficiente con un solo fotograma en la línea de tiempo. Si hubiéramos hecho esta animación en la línea de tiempo, hubiéramos necesitado extender mucho más el número de fotogramas.

Paso 14 de 19

Arrastramos otra instancia del clip `cloud` al escenario en la misma capa `clouds` que la nube anterior, y en el inspector de **Propiedades** le damos el nombre de instancia `cloud2_mc`.



Al principio de la programación, tras la línea en la que habíamos definido el `alpha` de la primera instancia, añadimos algunas propiedades para la nueva instancia.

```
cloud1_mc.alpha = .6;
cloud2_mc.alpha = .4; //more transparent
cloud2_mc.scaleY = .7; //lower height (70%)
cloud2_mc.z = 300; //add depth
```

Después añadimos los mismos listeners para esta segunda instancia. Para ello basta con copiar y pegar los listeners ya creados y sustituir `cloud1_mc` por `cloud2_mc`:

```
cloud1_mc.addEventListener(MouseEvent.CLICK, drag);
cloud1_mc.addEventListener(MouseEvent.CLICK, drop);
cloud1_mc.addEventListener(MouseEvent.CLICK, drop);
cloud1_mc.addEventListener(MouseEvent.CLICK, drop);
cloud1_mc.addEventListener(MouseEvent.CLICK, drop);

cloud2_mc.addEventListener(MouseEvent.CLICK, drag);
cloud2_mc.addEventListener(MouseEvent.CLICK, drop);
cloud2_mc.addEventListener(MouseEvent.CLICK, drop);
cloud2_mc.addEventListener(MouseEvent.CLICK, drop);
cloud2_mc.addEventListener(MouseEvent.CLICK, drop);
```

Dentro de cada función hacíamos también referencia a la instancia `cloud1_mc`. En vez de repetir las líneas con la programación para `cloud2_mc` dentro de las funciones, lo que vamos a hacer es que la programación haga referencia al objeto que inició el evento.

Es decir, si la función `drag` ha sido llamada por un listener de la `cloud1_mc`, arrastraremos esa nube, pero si la función ha sido iniciada por la `cloud2_mc`, será esta nube la que arrastremos.

Paso 15 de 19

Para ello tenemos que escribir en primer lugar el nombre del parámetro del evento que hemos puesto en la función (`e` en nuestro caso). Después añadimos la propiedad `target`. Con esto se hará referencia al objeto que disparó la acción al recibir el evento.

Por lo tanto, si sustituimos dentro de las funciones el nombre de la instancia por `e.target`, conseguiremos tener una referencia directa a los objetos que han enviado la función.

Para comprobarlo en la función `drag`, además de sustituir `cloud1_mc` por `e.target`, añadiremos un `trace` que nos muestre el nombre (`name`) de la instancia que desencadena el evento.

```
function drag(e:MouseEvent):void
{
    e.target.startDrag();
    trace(e.target.name);
}
```

Al no estar el contenido del `trace` entre comillas, la función no escribe literalmente lo que hemos escrito entre paréntesis, sino su valor (en este caso es un nombre). De esta manera, veremos que al pulsar sobre cada nube el panel de salida nos muestra su nombre.

Añadimos `e.target` sustituyendo a `cloud1_mc` en todas las funciones.

Ahora tenemos un problema con las posiciones que habíamos determinado en el condicional `if`, ya que al tener la segunda nube más profundidad deberíamos distanciar ambos extremos.

Una forma de saber aproximadamente estos valores es cambiar el `trace` de la función `drag` para que nos muestre la posición `x` del clip sobre el que clicemos:

```
trace(e.target.x);
```

Después desactivamos temporalmente el `if` de la función `wind` convirtiéndolo en un comentario añadiendo `/*` antes del `if` y `*/` después de la llave de cierre del `if`.

```
/*if (e.target.x > 560)
{
    e.target.x = -160;
}*/
```

Paso 16 de 19

Ahora probamos la película y tratamos de pulsar sobre la segunda nube en posiciones muy cercanas al límite del escenario por ambos lados para hacernos una idea de los valores de `x` adecuados.

El panel **Salida** mostrará el valor `x` para esta nube, que como podemos ver, ha variado considerablemente al estar programada para mostrarse en un plano más profundo.

De hecho, esa diferencia entre la propiedad `x` dependiendo de la profundidad la podemos observar también en la animación, ya que aunque ambas nubes tienen asignado el mismo incremento en el valor `x`, la más lejana parece avanzar más despacio que la más cercana.

En nuestro caso, como valor mínimo seleccionaremos `-315` y como valor máximo `715`, pero esos valores dependerán del tamaño de nuestra nube.

Ahora la primera nube tardará más en aparecer. Podríamos añadir un condicional, y dependiendo de si el `target` es la primera nube o la segunda, tomar unos límites u otros. En este caso no tiene importancia que una nube tarde más en aparecer de nuevo, así que dejamos como límites los de la nube más lejana.

Como hemos podido comprobar, el uso de `trace` y de los comentarios pueden ser muy útiles a la hora de programar.

Después de borrar el `trace`, pues ya no lo necesitamos, las funciones quedarán así:

```
function drag(e:MouseEvent):void
{
    e.target.startDrag();
}

function drop(e:MouseEvent):void
{
    e.target.stopDrag();
}

function wind(e:Event):void
{
    e.target.x += .4;
    if (e.target.x > 715)
    {
        e.target.x = -315;
    }
}
```


Paso 17 de 19

Como último detalle, vamos a hacer que se muestre un cursor con forma de mano cuando estemos sobre la nube, para que parezca un botón y sea más fácil intuir que la nube se puede arrastrar.

Para ello, antes de los listeners añadimos las siguientes propiedades:

```
cloud1_mc.buttonMode = true;
cloud2_mc.buttonMode = true;
```

Con esto ya veremos que el cursor adopta la forma de una mano cuando estamos encima de una nube, ya que trata visualmente a las nubes como si fueran un botón.

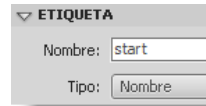
Analicemos lo que ocurre con nuestra película. Comienza en el fotograma 1 y el player de flash lee toda la programación que hemos escrito y la va ejecutando. Después la línea de tiempo continúa durante 600 fotogramas, y de vuelta al fotograma 1, donde vuelve a leer toda la programación.

Para no sobrecargar al player con información repetitiva, podemos hacer que el bucle de la línea de tiempo principal vaya del fotograma 600 al fotograma 2, sin pasar de nuevo por el fotograma 1, que es donde hemos escrito toda nuestra programación.

Para ello creamos un **fotograma clave** en el fotograma 600 de la capa as, y escribimos la siguiente programación:

```
gotoAndPlay(2);
```

Esto significa que cuando la cabeza lectora llegue al fotograma 600, donde está escrita la programación, irá al fotograma 2 y continuará con la reproducción.



Si hubiéramos creado un fotograma clave y le hubiéramos asignado en el inspector de Propiedades el nombre *start*, también podríamos haber ido a ese fotograma escribiendo:

```
gotoAndPlay("start");
```

Este paso no es necesario en este caso, pero puede resultar útil conocer esta opción para crear fácilmente menús de navegación.

Paso 18 de 19

Este es el aspecto completo que tendrá la programación completa de este tutorial:

```
cloud1_mc.alpha = .6;
cloud2_mc.alpha = .4;
cloud2_mc.scaleY = .7;
cloud2_mc.z = 300;
cloud1_mc.buttonMode = true;
cloud2_mc.buttonMode = true;

cloud1_mc.addEventListener(MouseEvent.CLICK, drag);
cloud1_mc.addEventListener(MouseEvent.CLICK, drop);
cloud1_mc.addEventListener(MouseEvent.CLICK, drop);
cloud1_mc.addEventListener(Event.ENTER_FRAME, wind);

cloud2_mc.addEventListener(MouseEvent.CLICK, drag);
cloud2_mc.addEventListener(MouseEvent.CLICK, drop);
cloud2_mc.addEventListener(MouseEvent.CLICK, drop);
cloud2_mc.addEventListener(Event.ENTER_FRAME, wind);
```

```
function drag(e:MouseEvent):void
{
    e.target.startDrag();
}

function drop(e:MouseEvent):void
{
    e.target.stopDrag();
}

function wind(e:Event):void
{
    e.target.x += .4;
    if (e.target.x > 715)
    {
        e.target.x = -315;
    }
}
```

Paso 19 de 19

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

1. Cambia la programación para que las nubes se desplacen de derecha a izquierda.
2. Añade una nueva nube en el escenario que también se desplace, pero que no se pueda arrastrar.
3. Haz que las nubes se vuelvan más transparentes al desplazarse, pero que al aparecer de nuevo por el otro lado del escenario tengan de nuevo su valor alfa original.

