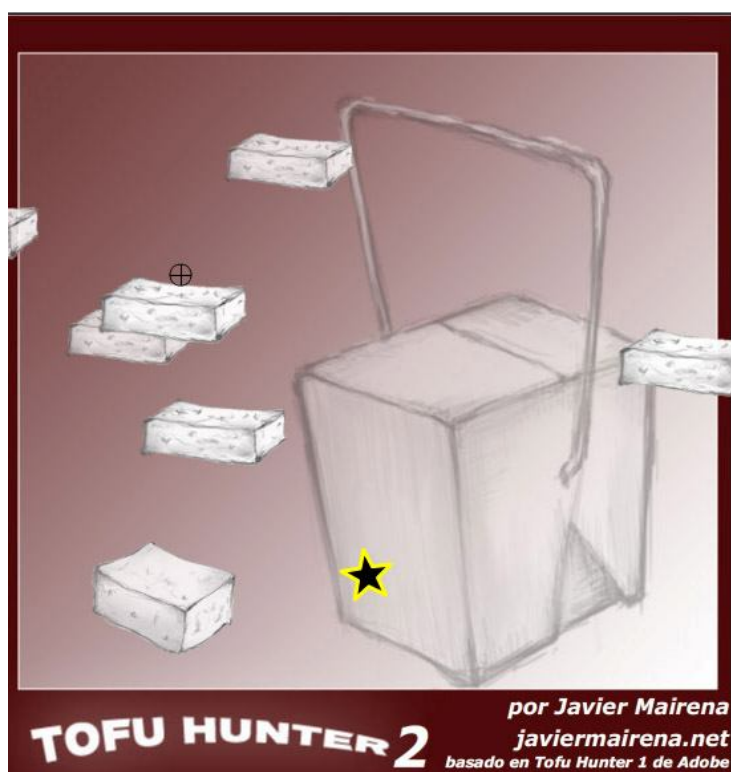


Programación de videojuegos en ActionScript 3

Tofu Hunter 2

2ª edición



Escrito por **Javier Mairena** - www.javiermairena.net
javier.mairena@gmail.com

<i>Advertencia: Cómo hacer y cómo no hacer juegos en flash.....</i>	<i>3</i>
<i>Introducción.....</i>	<i>5</i>
<i>Antes de empezar.....</i>	<i>6</i>
<i>Elegir entre Flash o Flex</i>	<i>7</i>
<i>1. Estructura del código AS3 para un videojuego</i>	<i>8</i>
1.1. Una clase principal.....	8
1.2. Programación organizada.....	8
<i>2. Tofus orientados a objetos.....</i>	<i>9</i>
<i>3. Timers para controlar el juego</i>	<i>10</i>
<i>4. Uso de Tweens para animaciones sencillas.....</i>	<i>11</i>
4.1 Tofus en movimiento con Tweens	11
4.2 Clases Tweens que no son de Adobe.....	12
<i>5. El sonido de un Tofu.....</i>	<i>13</i>
<i>6. Colisiones de una estrella.....</i>	<i>13</i>

Advertencia: Cómo hacer y cómo no hacer juegos en flash

Este apartado ha sido añadido en la 2ª edición de este documento para advertir sobre cómo hacer y cómo no hacer un juego flash, ya que este documento comete los mismos fallos de programación que muchos tutoriales que hay en Internet sobre desarrollo de juegos en flash.

He decidido no eliminar este documento a pesar de que en él se explique una forma no correcta de hacer un juego en flash, porque sí que puede resultar una forma más fácil para los que estén aprendiendo programación en general, como primer paso fácil y motivador a seguir aprendiendo. Pero aquellos programadores experimentados deberían seguir con atención las siguientes razones de porque no es correcto este manual y cómo se debería hacer realmente un juego.

Existe un peligro grande de aprender a programar videojuegos empezando por Flash/Actionscript por varias razones:

- Es una API claramente para software “orientado a eventos” del usuario, como una página web o software de gestión.
- El 99% de los libros y manuales que hay sobre ActionScript usa la librería de igual forma, pensando en software orientado a eventos del usuario.

La forma de programar orientado a eventos se centra mucho en esperar las acciones del usuario y a partir de ahí lanzar cierta lógica de ejecución, sin importar demasiado que exista una única “línea lógica de ejecución” ya que muchas de las acciones son totalmente independientes y no influyen entre ellas.

Por esta razón en muchas webs, programas o juegos hechos en flash se observará un uso excesivo de “event listeners” que ejecutan código generando varias líneas lógicas de ejecución haciendo imposible cierto tipo de control sobre el juego.

Para hacer bien un videojuego en flash hay que tener siempre en mente el modo clásico de un motor de un videojuego con un “main game loop”. Muchos libros y artículos en Internet tratan este tema, por ejemplo:

http://en.wikipedia.org/wiki/Game_programming#The_game_loop

Y evitar, en la medida de lo posible, el uso de los event listeners.

Además, recomiendo la lectura de estos dos artículos para entender mejor la forma en la que funciona flash:

El sistema de renderizado de Flash Player

<http://llops.com/blog/2008/05/24/el-sistema-de-renderizado-de-flash-player/>

EnterFrame vs Timer (I)

<http://llops.com/blog/2008/07/27/enterframe-vs-timer-i/>

Trabajando de esta manera tendremos:

- Control completo de la lógica del juego en todo momento.
- Facilidad para cambiar la velocidad del juego en cualquier momento o hacer pausa.
- Control para asegurarnos de que el juego se ejecutará siempre a la misma velocidad en todos los pcs.

Teniendo en cuenta el modelo clásico de cómo hacer un videojuego, Flash sigue siendo una herramienta muy potente porque posee una API muy completa, fácil y útil para videojuegos y podemos exportar para gran cantidad de plataformas.

Introducción

Este documento repasa algunos métodos de programación recomendados para la creación de un videojuego con ActionScript 3 (AS3).

Es recomendable tener conocimientos de AS3, del entorno de trabajo de Flash 9 y tener claros los conceptos de programación orientada a objetos.

Tomaré de ejemplo el juego **Tofu Hunter** que Adobe da como ejemplo de programación en su web y explicaré como desarrollar un nuevo juego, **Tofu Hunter 2**, mejorando el anterior en cuanto a estructura y forma de programación haciéndola más adecuada para la creación un videojuego en AS3 e incluiré la reproducción de sonidos y la detección de colisiones.

Principalmente veremos:

- Estructura del código AS3 para un videojuego.
- Timers para controlar la mecánica del juego.
- Programación orientada a objetos en AS3.
- Uso de Tweens para animaciones sencillas.
- Reproducción de sonidos.
- Detección de colisiones.

En este documento sólo trato los conceptos de forma general, para ver su implementación real es necesario consultar el código fuente de **Tofu Hunter 2** que está disponible en mi web: www.javiermairena.net.

Nota: no quiero desvalorar el trabajo que ha hecho Adobe al dar **Tofu Hunter** como ejemplo de programación en AS3, soy consciente de que Adobe dio este ejemplo para que se entendieran algunos conceptos de programación, dejando a un lado otros para un mayor entendimiento.

En **Tofu Hunter 2**, para simplificar el ejemplo yo he preferido dejar de lado el sistema de control de puntuaciones de **Tofu Hunter** y dar un paso más explicando otros conceptos importantes a la hora de programar un videojuego en AS3.

Antes de empezar

Necesitaremos tener instalado Flash 9 o superior.

Puedes descargar una versión de evaluación de Flash 9 (CS3) en:

https://www.adobe.com/cfusion/tdrc/index.cfm?product=flashpro&loc=es_es

Después, en mi web (www.javiermairena.net) se encuentra el código fuente de **Tofu Hunter** y **Tofu Hunter 2**.

Nota:

Originalmente el código fuente de **Tofu Hunter 1** se encuentra en el conjunto de ejemplos que nos da Adobe sobre programación en AS3 en Flash:

http://livedocs.adobe.com/flash/9.0/main/samples/Flash_ActionScript3.0_samples.zip

Hay también una referencia a este archivo en:

<http://www.adobe.com/support/documentation/en/flash/samples/>

Tofu Hunter 1 tiene el nombre **animation** dentro del archivo **Flash_ActionSript3.0_samples.zip**

Elegir entre Flash o Flex

Para programar un videojuego podemos elegir entre usar Flash o Flex, los dos generan la misma película flash y el resultado será el mismo.

Aunque a la hora de programar, un programador se sentirá más cómodo en Flex porque tanto FlashDevelop como FlexBuilder (los dos entornos de trabajo con los que podemos trabajar con Flex) están totalmente orientados a la programación, muy parecidos a otros entornos de programación.

Otra ventaja de usar Flex es que las librerías de Flex son gratuitas al igual que el entorno de trabajo FlashDevelop, aunque seguiremos necesitando usar Flash si queremos importar gráficos creados por nosotros.

En este tutorial usaremos Flash por ser un ejemplo sencillo y para no tener que explicar el uso de otro entorno.

1. Estructura del código AS3 para un videojuego

1.1. Una clase principal

En **Tofu Hunter 1** todo el código del juego se encuentra en la capa “actions” del proyecto Flash.

Esto sería correcto sólo si el código de nuestra película Flash fuera algo muy sencillo, y en el caso de un videojuego no es así.

Es algo habitual encontrar esta forma de programar en flash en muchos de los ejemplos que hay en Internet, pero de esta forma el código se vuelve menos legible y menos reutilizable.

En **Tofu Hunter 2** crearemos una clase principal del juego en un archivo “.as” externo a nuestro proyecto Flash, y un archivo .as por cada clase que creemos.

De esta forma el código queda separado de la parte de diseño de flash, y cualquier clase se podrá reutilizar en otro juego tan sólo copiando su archivo .as

Para crear la clase principal del juego y especificar al proyecto flash cual es:

1º- Creamos el archivo “.as”.

2º- Una vez abierto el nuevo documento AS lo guardaremos en **la misma carpeta de nuestro proyecto Flash**.

3º- En las propiedades de la película Flash en “Clase de documento” escribimos: “TofuHunter2”.

4º- En nuestra capa “actions” sólo introduciremos una llamada a nuestra función encargada de hacer iniciar el juego.

1.2. Programación organizada

En el código **Tofu Hunter** se mezclan las declaraciones de variables con las ejecuciones de instrucciones y las declaraciones de las funciones.

Parece que se ha programado con una visión lineal de un lenguaje estructurado, pero AS3 es un lenguaje orientado a objetos y esto nos posibilita tener una programación mucho más organizada y reutilizable.

En **Tofu Hunter 2** tendremos nuestra clase principal del juego en la que declararemos nuestras variables globales (que son los atributos de nuestra clase principal) y que tendrá una función que se encargará de iniciar el juego y llamar al resto de funciones del juego, que podrán ser de la clase principal o de otras que creamos.

Por ejemplo, en vez de crear y programar el comportamiento de nuestro puntero en la línea principal del tiempo (como se hace en **Tofu Hunter**), haremos una clase “Puntero” de la que crearemos un instancia en nuestra clase principal del juego. Dentro de la clase “Puntero” es donde programaremos el comportamiento de nuestro puntero, que será seguir el movimiento de la posición del ratón.

Finalmente en la clase principal tan sólo escribimos algo que queda mucho más claro:

```
var oPuntero:Puntero = new Puntero();
oPuntero.Inicio(this);
```

En Tofu Hunter 2 tendremos la siguientes clases:

- TofuHunter2: la clase principal del juego.
- Puntero: controla el comportamiento del puntero en el juego.
- Tofu: todos los tofus serán objetos de esta clase.
- EstrellaMalvada: estrella que nos servirá de ejemplo para las colisiones.
- Utils: agrupa funciones genéricas que nos podrán servir en otros juegos.

Cada clase irá en un archivo .as distinto y todos deberán estar en la misma carpeta del proyecto Flash.

2. Tofus orientados a objetos

Como vimos en el apartado *1.2 Programación organizada*, **Tofu Hunter** no utiliza el potencial de la programación orientada a objetos. Antes nos referíamos a la estructura del código en general, ahora veremos como se debería utilizar para la creación de instancias de una nueva clase Tofu.

En **Tofu Hunter**, en la clase principal del juego se crean nuevos Tofus de esta manera:

```
switch (randomTofu) {
    case 1:
        thisMC = new tofu1_mc();
        break;
    case 2:
        thisMC = new tofu2_mc();
        break;
    case 3:
        thisMC = new tofu3_mc();
        break;
    default:
        return;
        break;
}
```

Y a continuación se cambian propiedades de thisMC y se le asignan ciertos Listeners. En total unas 70 líneas de código aproximadamente.

En **Tofu Hunter 2** crearemos la clase Tofu, donde que será donde programaremos las propiedades con la que se tiene que iniciar un Tofu y los Listeners que le queramos asignar, ya sea en el propio constructor de la clase o en una función propia para iniciarlo.

Al iniciar el objeto de la clase Tofu también le añadiremos como hijo uno de los tres tipos posibles de MovieClips que son los dibujos de los Tofus.

En nuestra clase principal tan sólo escribiremos esto:

```
var oTofu:Tofu = new Tofu();  
oTofu.IniciaTofu(this);
```

3. Timers para controlar el juego

En **Tofu Hunter**, una vez iniciado el juego, la mecánica básica se ejecuta cada vez que la película Flash dibuja una imagen de la animación (frame).

Incluyeron un Listener para el evento ENTER_FRAME:

```
stage.addEventListener(Event.ENTER_FRAME, enterFrameHandler);
```

La function enterFrameHandler es la que se encarga de generar los tofus que aparecen en pantalla.

Esto es algo peligroso porque estamos basando la velocidad de la mecánica del juego en la capacidad de mostrar imágenes del ordenador.

Si tenemos configurada la película Flash a 30 imágenes por segundo puede que en un ordenador antiguo, o en un juego Flash muy complejo, la cantidad de imágenes por segundo que el ordenador consiga mostrar sean 20. Con lo que la velocidad real del juego se está reduciendo también y realmente a 20 imágenes por segundo se podría haber seguido jugando con una suavidad visual de movimientos aceptable.

En **Tofu Hunter 2** crearemos un Timer para controlar la creación de Tofus.

Un Timer que se ejecute cada 33 milisegundos es equivalente a llamar a una función en cada frame de una película configurada a 30 imágenes por segundo:

```
private var oTimerCreaTofu:Timer = new Timer(33);  
oTimerCreaTofu.addEventListener("timer", TimerCreaTofuHandler);  
oTimerCreaTofu.start();
```

De esta manera aunque en algún momento nuestro PC no fuera capaz de llegar a pintar 30 imágenes por segundo y viéramos las animaciones con menos de fluidez de imágenes, el desarrollo real del juego seguiría ejecutándose a la misma velocidad.

Cabe pensar aquí el caso en el que la lógica del juego fuera muy compleja y al no estar sincronizada con el pintado de las imágenes se mostrara una imagen con algún elemento en una situación no coherente con el resto de la escena, pero en la mayoría de los proyectos que realicemos que se de esta situación será algo más raro que el

inconveniente explicado anteriormente por haber escrito la lógica en el evento Enter_Frame; y aún así siempre podremos crear un sistema para controlar esto.

4. Uso de Tweens para animaciones sencillas

4.1 Tofus en movimiento con Tweens

El movimiento automático de los tofus en el juego es sencillo: se desplazan de izquierda a derecha, subiendo un poco, como si flotaran, y cuando son destruidos descienden.

En **Tofu Hunter** para controlar este movimiento incluyeron un Listener al MovieClip del tofu en su evento ENTER_FRAME.

La función que ejecuta el Listener cada vez que es llamada se encarga de incrementar la coordenada X y disminuir la coordenada Y del tofu.

Esto hubiera sido una buena práctica si se hubiera llamado a la función con un Timer y no con el evento ENTER_FRAME como ya expliqué en apartado **2. Timers para controlar el juego**.

Además tampoco tiene sentido estar incrementando los valores X e Y del MovieClip en un movimiento tan sencillo como es un desplazamiento lateral y vertical que no varía su trayectoria porque el juego no tiene elementos que así lo hagan cuando existe la clase Tweens.

Para movimientos sencillos en AS3 tenemos la clase Tween, que se encarga automáticamente de variar una propiedad de un objeto de forma progresiva según le digamos. Así, por ejemplo, podemos variar progresivamente las coordenadas de un objeto.

En **Tofu Hunter 2**, para el desplazamiento lateral añadiremos el siguiente Tween una vez es creado el tofu:

```
this.oTweenLateral = new Tween(this, "x", Regular.easeIn, this.x,
this.mcJuego.originalWidth + 100, nRandom, true);
```

Para eliminar el tofu cuando haya salido del juego por la derecha añadiremos un Listener al Tween que se ejecute cuando termine la animación y que se encargue de eliminar el tofu del juego:

```
this.oTweenLateral.addEventListener(TweenEvent.MOTION_FINISH,
TweenLateralHandler);
```

Para hacer que también vaya subiendo poco para que parezca que flotan, como en **Tofu Hunter**, incluiremos otro Tween:

```
this.oTweenVertical = new Tween(this, "y", Regular.easeIn, this.y, this.y -
50, nRandom, true);
```

Con el movimiento de caer y girar un poco cuando es destruido haremos algo muy parecido:

```

this.oTweenVertical = new Tween(this, "y", Strong.easeIn, this.y,
this.mcJuego.originalHeight + 100, 1, true);

//Para eliminarlo cuando termine
this.oTweenVertical.addEventListener(TweenEvent.MOTION_FINISH,
TweenVerticalHandler);

//Lo hacemos girar un poco cuando es destruido, como Tofu Hunter 1
this.oTweenLateral = new Tween(this, "rotation", Regular.easeIn,
this.rotation, this.rotation + 100, 1, true);

```

4.2 Clases Tweens que no son de Adobe

En TofuHunter2 hemos usado la clase Tween de Adobe por ser fácil de usar y por no tener que incluir el uso de otra clase externa, pero la clase Tween de Adobe no funciona correctamente y es muy aconsejable usar otras clases externas Tween que no son de Adobe.

La clase Tween de Adobe tiene dos problemas muy importantes:

- **Las animaciones se detienen sin ninguna razón.**
Si la usamos para modificar varias propiedades de un mismo objeto a la vez (por ejemplo para moverlo, girarlo y ampliarlo) acabaremos teniendo problemas porque habrá veces que las animaciones se detengan sin ninguna razón.
- **El rendimiento no es bueno.**
Si ejecutamos muchos tweens sobre muchos objetos el rendimiento es muchísimo menor que con otras clases Tween que no son de Adobe.

Las clases Tween más usadas que no son de Adobe son:

- **Tweener.**
Gratuita. <http://code.google.com/p/tweener/>
Personalmente es la que menos me gusta a la hora de programar porque no puedes crear objetos de la clase y tratarlos por separado, si no que añades y quitas animaciones a la clase principal.
Puede que tenga esta estructura por ser una adaptación de la clase Tween de ActionScript 2.
- **TweenLite y TweenMax.**
Gratuitas sólo para fines no comerciales. <http://blog.greensock.com/>
Según un test hecho por su creador son las de mejor rendimiento. TweenLite puede resultar muy útil para la creación de banners por su tamaño más reducido.
- **gTween.**
Gratuita. <http://www.gskinner.com/libraries/gtween/>
Actualmente en desarrollo, pero funciona sin errores.

Ventajas de usar cualquiera de estas clases:

- **Funcionan.**
No se detienen sin razón como lo hace la clase Tween de Adobe.

- **Rendimiento mayor.**
El rendimiento es mayor cuando ejecutamos muchos Tweens sobre muchos objetos.
Puedes ver una comparativa de rendimiento hecha por el creador de TweenMax: <http://blog.greensock.com/tweening-speed-test/>
- **Código más legible.**
Con la clase Tween de adobe tenemos que crear un objeto de esta clase por cada propiedad que queramos cambiar en otro objeto.
Con cualquiera de las otras clases Tweens una instancia y en una sola línea de código podemos lanzar cambios en tantas propiedades como queramos.
- **Más tipos de suavizados.**
Estas otras clases de Tweens incluyen más variedad de tipos de suavizados en la animación o variación de una propiedad.

5. El sonido de un Tofu

En **Tofu Hunter 2** incluiremos en la función `tofuClickHandler` un par de líneas para que cada vez que destruimos un tofu se reproduzca un sonido que previamente hemos agregado a la biblioteca del proyecto:

```
var sndExplosion:Explosion = new Explosion();
var channel:SoundChannel = sndExplosion.play();
```

6. Colisiones de una estrella

Tofu Hunter 2 tiene un nuevo elemento en el juego: una estrella que si toca alguno de los Tofus los elimina del juego.

Cada instancia de la clase Tofu tendrá un Timer que comprobará constantemente si está colisionando con la estrella.

En la función `IniciaTofu`:

```
this.oTimerColision = new Timer(10);
this.oTimerColision.addEventListener("timer", TimerColisionHandler);
this.oTimerColision.start();
```

Y para comprobar si colisiona crearemos la siguiente función que es llamada por el Timer:

```
private function TimerColisionHandler(event:Event){
    if (this.hitTestObject(this.mcJuego.oEstrella))
    {
        this.Eliminar();
    }
}
```

En ActionScript existen dos tipos de detección de colisiones:

hitTestObject – Evalúa la colisión de dos objetos tratándolos como cuadrados que rodean al objeto, sin tener en cuenta la figura real del objeto.

hitTestPoint – Evalúa la colisión entre un punto en concreto con la figura real de un objeto.