PROGRAMACION DE VIDEOJUEGOS III

# Nuestro primer proyecto en HaxeFlixel

En éste tutorial daremos finalmente nuestros primeros pasos programando en HaxeFlixel. Entre otros temas, abordaremos la forma de crear un proyecto y la estructura del mismo, e introduciremos algunos de los elementos del lenguaje de programación haXe mediante el análisis de un ejemplo concreto.

## Creación de un proyecto de HaxeFlixel

Para crear un proyecto de HaxeFlixel en blanco, será necesario utilizar nuevamente la consola y escribir el siguiente comando:

```
haxelib run flixel-tools template -n "Nombre de
nuestro proyecto"
```

o bien:

```
flixel template -n "Nombre de nuestro proyecto"
```

Será necesario reemplazar la cadena *"Nombre de nuestro proyecto"* por el nombre que deseemos para nuestro proyecto. Recordar que puede abrirse una consola de comandos en un directorio determinado haciendo click con el botón derecho al mismo tiempo que se mantiene apretada la tecla *Shift* y seleccionando la opción *"Abrir ventana de comandos aquí"*. Al ejecutar éste comando, se creará una nueva carpeta con el nombre del proyecto en el directorio de trabajo actual. Dentro de esa carpeta se encontrará un archivo con el nombre del proyecto y la extensión *.hxproj* que podrá abrirse con FlashDevelop. Asimismo, a modo de ayuda, puede utilizarse éste script para facilitar el proceso de creación de un nuevo proyecto.

# Estructura de un proyecto

En la *Fig. 1* puede observarse la estructura de directorios de un proyecto en HaxeFlixel.

El directorio *assets* incluye otros subdirectorios en los que podremos depositar los recursos artísticos (imágenes, música, archivos de niveles, etc) necesarios para nuestros juegos.

El directorio *export* (en algunos casos llamado *bin*) no existirá inicialmente sino que será creado luego y contendrá los ejecutables generados para cada plataforma.

En el directorio source se encontrarán los fuentes del proyecto. Los archivos de código fuente en lenguaje haXe llevan la extensión *.hx*.

Finalmente, existirá un archivo con extensión .hxproj (que no aparece en la imagen) que corresponde al archivo de proyecto utilizado por FlashDevelop, mientras que el archivo *Project.xml* es un descriptor de la aplicación que será generada, el mismo contiene diversa información como el icono del ejecutable o la resolución utilizada en plataformas móviles y otros parámetros de configuración para producir los ejecutables.

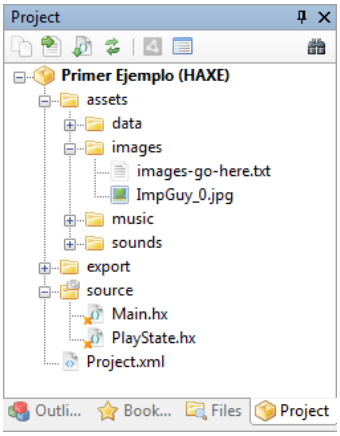


*Fig. 1: Estructura de archivos de un proyecto de HaxeFlixel*

# Primer ejemplo en HaxeFlixel

A continuación, se analizará el ejemplo cuya estructura de archivos se observa en la *Fig. 1* y que hemos explicamos anteriormente. El ejemplo corresponde a un proyecto minimalista que posee sólo dos archivos fuente: *Main.hx* y *PlayState.hx*. El primero contiene declaraciones e inicializaciones de valores necesarios para el funcionamiento del motor sobre los que no profundizaremos por el momento. Por otro lado, centraremos

nuestra mirada en el contenido del archivo *PlayState.hx*, el cual puede observarse en la *Fig. 2*.

```
import flixel.FlxState;
import flixel.FlxSprite;

class PlayState extends FlxState
{
    private var sprite1: FlxSprite;

    override public function create():Void
    {
        super.create();
        sprite1 = new FlxSprite(100, 100,
"assets/images/ImpGuy_0.jpg");
        add(sprite1);
    }

    override public function update():Void
    {
        super.update();
    }
}
```

*Fig. 2: Contenido del archivo PlayState.hx del proyecto analizado*

En HaxeFlixel, las escenas o estados de juego (por ejemplo la pantalla de menu, la pantalla de juego, la pantalla de game Over, etc) se representan mediante la clase **FlxState**, el primer paso en la creación de un videojuego es definir al menos una escena. Para ésto, debemos crear una clase que herede de **FlxState** y redefinir algunos de sus métodos.

Los métodos de la clase **FlxState** que nos interesará redefinir son los siguientes:

- *create()*
- *update()*
- *draw()*
- *destroy()*

El motor se encargará de llamar a cada función automáticamente en el momento indicado. Los métodos *create()* y *destroy()* serán invocados al inicializar y destruir la escena (no debe confundirse a éstos métodos con el constructor y destructor). Por otro lado los métodos *update()* y *draw()* se invocarán una vez por cada frame para actualizar y dibujar la escena.

La cantidad de veces por segundo que la escena se

actualizará y dibujará, así como también la escena inicial del juego, pueden definirse modificando las respectivas variables en el archivo *Main.hx*.

En el código de la *Fig. 2* se observa que hemos definido una escena llamada **PlayState** la cual hereda de la clase **FlxState**. En el lenguaje haXe, se utiliza la palabra reservada ***extends*** para indicar que una clase hereda de otra.

En el ejemplo sólo se han redefinido dos de las funciones listadas anteriormente (lo cual implica que las restantes mantienene el comportamiento heredado de la clase padre). En el código se puede observar la forma de declarar una función en haXe:

```
[private|public] function nombreDeLaFuncion(
                                    NombreParam1: TipoParam1,
                                    NombreParam2: TipoParam2):
                                    TipoRetorno
{
}
```

Además, al redefinir un método que ha sido heredado de una clase padre debe siempre anteponerse la palabra reservada **override** a la declaración de la función, tal como se observa en la *Fig. 2*.

En el caso de redefinir funciones en una clase hija, la palabra reservada **super** permite hacer referencia a los métodos sobreescritos de la clase padre. Por ejemplo, en el caso de nuestra clase **PlayState**, la función *update()* consiste simplemente en invocar a la función con el mismo nombre que fue definida en la clase padre, por lo que dicha construcción es equivalente a no redefinir la función y permitir que conserve el comportamiento de la clase padre.

La parte más interesante del código corresponde al método *create()* que, como explicamos anteriormente, será llamado para inicializar la escena. Un poco más arriba de su definición se observa la declaración de una variable/atributo que es utilizada luego dentro del método *create()*: *sprite1*. En haXe, la declaración de una variable se realiza de la siguiente manera:

```
[private|public] var nombreDeLaVariable: TipoDeLaVariable;
```

Al igual que en otros lenguajes como Java, Python o AS3, en Haxe, todas las variables son referencias, un concepto que guarda similitud con el de punteros y objetos dinámicos de C++. Ésto significa que todas las variables que se declaren serán simplemente contenedores vacíos hasta que se les asigne un

objeto específico creado utilizando la primitiva **new**.

Dentro del método *create()* puede observarse, primero, la invocación del método con el mismo nombre de la clase padre (mediante la palabra reservada **super**) y luego la creación de un objeto de tipo **FlxSprite** y su asignación a la variable/referencia *sprite1*. Éste objeto representa, por supuesto, un sprite del juego. Para crearlo, se especifican en la llamada a su constructor las coordenadas de su posición inicial y la imagen del mismo.

La llamada al método *add()* (perteneciente a la clase **FlxState**) agrega el sprite creado a la escena, de manera que la misma se encargará, de ahora en adelante, de actualizarlo y dibujarlo de manera automática.

Finalmente, y como puede apreciarse en el código de la *Fig. 2*, la inclusión de bibliotecas en haXe, se hace mediante la palabra reservada ***import***. En nuestro caso, hemos importado las clases **FlxState** y **FlxSprite**. Cabe destacar que en haXe, los nombres de clases deben comenzar siempre con mayúsculas y el nombre debe ser el mismo del archivo que contiene la definición.

En la *Fig. 3* puede observarse la salida del programa analizado, el cual consiste en una escena que dibuja en pantalla el sprite creado (puede hacerse click sobre la imagen para ejecutarlo). La imagen utilizada en el ejemplo ha sido tomada de aquí.

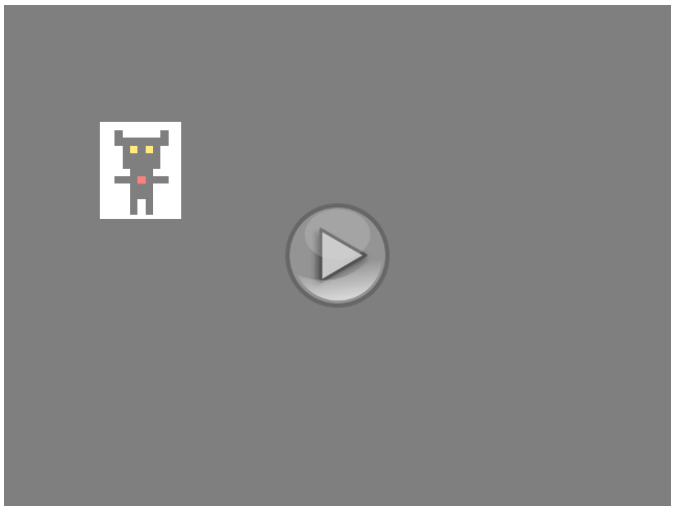

*Fig. 3: Captura del ejemplo analizado a lo largo del tutorial*

Descargar código del ejemplo

Para finalizar el tutorial, se ofrece debajo una breve síntesis de los conceptos más sobresalientes abordados durante el tutorial. Además, se recomienda dar una rápida mirada a la

documentación de referencia de las clases FlxState y FlxSprite.

**Resumen:**

En HaxeFlixel...

- La clase **FlxState** representa una escena o estado de juego
- La clase **FlxSprite** representa un sprite
- Los sprites/actores agregados a la escena mediante el método *add()* serán actualizados y dibujados automáticamente por el motor

En el lenguaje de programación haXe...

- La herencia se realiza mediante la palabra reservada **extends**
- Se debe utilizar la palabra reservada **override** para redefinir un método heredado de una clase padre
- Los nombres de clase comienzan con mayúscula y son iguales al del archivo que contiene la definición
- La palabra reservada **super** permite acceder a los métodos de la clase padre en caso de haberlos redefinido
- Las bibliotecas se incluyen mediante la palabra reservada **import**

Volver al índice de tutoriales...