PROGRAMACION DE VIDEOJUEGOS III

Colisiones (parte I: detección)

En el siguiente tutorial comenzaremos a abordar un tópico de suma importancia a la hora de desarrollar juegos: la detección y manejo de colisiones.

HaxeFlixel permite detectar y manejar colisiones de una manera realmente sencilla e intuitiva a través de unas pocas llamadas a funciones. El motor cuenta con un sistema de manejo de colisiones incorporado que utiliza AABBs *(Axis Aligned Bounding Box)* para cada uno de sus objetos. Además, utiliza quadtrees para realizar la detección de solapamiento entre un gran número de objetos de manera eficiente.

Para el ejemplo que desarrollaremos en éste tutorial, hemos utilizado los sprites del juego Evoland (la versión clásica) obtenidos desde GitHub. Evoland es un juego de acción/aventura que representa la evolución de los videojuegos, fue creado para la competencia LudumDare #26, de la cual resultó ganador. El juego fue creado por Nicolas Cannasse, casualmente el principal creador del lenguaje haXe.

El ejemplo consiste en el personaje principal que puede moverse por la pantalla en la que también existen un enemigo que se mueve al azar y un item, entre los cuales se detecta solapamiento. El resultado puede observarse en la *Fig. 1*.

Fig. 1: Captura del ejemplo analizado a lo largo del tutorial, utilizar las teclas WASD para mover al personaje

Descargar código del ejemplo

En la *Fig. 2* se observa el código de la clase **PlayState** correspondiente a la escena principal del juego. La escena define varios atributos: personaje principal es de tipo **Hero**, mientras que el enemigo pertenece a la clase **Monster**. La clase **Sword** representa la espada. Todas las clases heredan de FlxSprite y definen su comportamiento propio, aunque no entraremos en detalle sobre ellas. La clase MessageBoard sirve simplemente para facilitar el procedimiento de mostrar mensajes temporales en pantalla.

La detección de colisiones se realiza dentro del método *update()* de la escena. El método estático *overlap()* de la clase **FlxG** permite conocer si dos objetos determinados se están solapando. El método recibe como parámetros dos objetos entre los que se desea probar si existe o no solapamiento y una función callback que será llamada en caso de que ésto ocurra. Además, el método devuelve verdadero o falso según se haya detectado o no solapamiento entre los objetos.

En el ejemplo, puede verse que se realizan dos llamadas al método: una para comprobar el solapamiento entre el héroe y el enemigo y otra entre el héroe y la espada. Ambas llamadas especifican dos callbacks distintos que se implementan más abajo.

```
import flixel.FlxState;
import flixel.FlxBasic;
import flixel.FlxSprite;
import flixel.FlxG;
```

```
class PlayState extends FlxState
{
    private var hero: Hero;
    private var monster: Monster;
    private var board: MessageBoard;
    private var sword: Sword;

    override public function create():Void
    {
        super.create();
        add(new FlxSprite(0, 0,
"assets/images/background.png"));

        monster = new Monster(64, 64);
        add(monster);

        hero = new Hero(128, 128);
        add(hero);

        sword = new Sword(160, 160);
        add(sword);

        board = new MessageBoard();
        add(board);
    }

    override public function update():Void
    {
        super.update();
        FlxG.overlap(hero, monster,
onOverlapHeroMonster);
        FlxG.overlap(hero, sword,
onOverlapHeroSword);
    }

    private function
onOverlapHeroMonster(TheHero: FlxBasic,
TheMonster: FlxBasic){
        TheHero.kill();
        board.showMessage("You've been
killed... :(");
    }

    private function onOverlapHeroSword(TheHero:
FlxBasic, TheSword: FlxBasic){
        TheSword.kill();
        board.showMessage("You've found a
sword!");
    }
}
```

Fig. 2: Contenido del archivo PlayState.hx del ejemplo analizado

Las funciones callbacks reciben como parámetros dos referencias a objetos de tipo **FlxBasic** que, en éste caso, serán los mismos que se han especificado en la llamada a *overlap()*. Los objetos de tipos **FlxBasic** representan a un actor básico de que puede agregarse a una escena. La clase **FlxSprite** es descendiente de **FlxBasic**. En éste caso, sabemos que los objetos que recibirá el callback no serán de tipo **FlxBasic** sino de tipo **Hero**, **Monster** o **Sword**. En tutoriales siguientes aprenderemos como realizar un casting desde el tipo genérico **FlxBasic** a la verdadera clase de los objetos que han colisionado.

El método *kill()* llamado en cada callback de colisión, quita al objeto de la escena, por lo cual ya no será actualizado ni dibujado. Dicho método pertenece, precisamente, a la clase **FlxBasic**. A pesar de eliminar al actor de la escena, el método *kill()* no elimina realmente al actor de la memoria, permitiendo que el mismo sea reutilizado más adelante. En tutoriales subsiguientes abordaremos también el tema de la reutilización de objetos.

Como hemos podido ver, la detección de colisiones en HaxeFlixel puede lograrse de manera simple gracias al sistema incorporado en el motor. Se recomienda consultar la documentación de la función *overlaps()* para conocer mayores detalles sobre sus parámetros y funcionamiento. Del mismo modo, la documentación de la clase **FlxBasic** puede mostrarnos algunos atributos y métodos que nos pueden resultar útiles.

Resumen:

- HaxeFlixel posee un motor para detección y manejo de colisiones incorporado que utiliza AABBs y quadtrees
- El método estático *overlap()* de la clase **FlxG** permite detectar si dos objetos están o no solapándose
- Prácticamente todos los objetos en HaxeFlixel descienden de la clase **FlxBasic**, que representa un objeto básico que puede ser agregado a una escena

Volver al índice de tutoriales...