Grupos (parte I)

En el tutorial anterior aprendimos a detectar colisiones entre objetos mediante la función *overlap()*. En el ejemplo visto el personaje podía colisionar con un item y con un enemigo, pero ¿qué ocurriría en el caso de que hubiese cientos de enemigos e items?¿Sería necesaria una llamada a *overlap()* para cada uno de ellos?

Para abordar éste problema y otros problemas, HaxeFlixel nos facilita la clase **FlxGroup**, que representa un grupo de actores. Es decir, un grupo es una clase especial de actor especial que puede contener dentro de sí mismo a otros actores, incluso otros grupos.

Los objetos del juego pueden agregarse a un grupo de la misma forma que a una escena. A su vez, un grupo puede ser agregado a la escena como cualquier otro actor. Al igual que cualquier actor, los grupos redefinen los métodos *update()* y draw(). En cada llamada a *update()*, el grupo invoca al método con el mismo nombre para cada uno de los actores que contiene. Lo mismo ocurre con el método *draw()*.

Como puede verse, el comportamiento de un objeto de tipo **FlxGroup** es muy similar al de una escena y, de hecho, la clase **FlxState** hereda de **FlxGroup**.

Tal como veremos en el ejemplo que se desarrollará en éste tutorial, el cal es una versión ligeramente modificada del ejemplo del tutorial anterior, la funcion *overlap()* vista anteriormente también admite grupos como parámetros para saber si existe solapamiento de algún objeto con alguno de sus miembros.

El ejemplo que analizaremos a continuación consiste en algunas modificaciones realizadas sobre ejemplo del tutorial anterior de manera que puedan incluirse varios enemigos en lugar de sólo uno. En la *Fig. 1* se observa el código correspondiente a la escena, el cual explicaremos brevemente.

La escena utiliza un objeto de tipo **FlxGroup**, el cual es un

atributo de la misma, para agrupar a los enemigos. Mediante el método *add()* se agregan al grupo cinco enemigos creados en posiciones aleatorias y éste es luego agregado a la escena de la misma manera. De ésta manera los enemigos se actualizarán y dibujarán automáticamente, ya que la escena actualiza y dibuja al grupo y éste, a su vez, hará lo mismo con los objetos que contiene.

```
import flixel.FlxBasic;
import flixel.FlxG;
import flixel.FlxSprite;
import flixel.FlxState;
import flixel.group.FlxGroup;
import flixel.text.FlxText;
import flixel.util.FlxRandom;

class PlayState extends FlxState
{
    private var hero: Hero;
    private var enemies: FlxGroup;
    private var board: MessageBoard;
    private var sword: Sword;
    private var energyText: FlxText;

    override public function create():Void
    {
        super.create();
        add(new FlxSprite(0, 0,
"assets/images/background.png"));
        // creamos un grupo para contener a los
enemigos
        enemies = new FlxGroup();
        // agregamos enemigos al grupo
        for(i in 0...5)
        {
            var x = FlxRandom.intRanged(0, 14)
* 16;
            var y = FlxRandom.intRanged(0, 19)
* 16;
            enemies.add(new Monster(x, y));
        }
        // agregamos al grupo a la escena
        add(enemies);

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

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

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

```
        energyText = new FlxText(0, 0, 200);
        updateEnergyText();
        add(energyText);
    }

    override public function update():Void
    {
        super.update();
        // comprobamos si colisiona el heroe
con los enemigos
        FlxG.overlap(hero, enemies,
onOverlapHeroEntities);
        FlxG.overlap(hero, sword,
onOverlapHeroEntities);
    }

    private function
onOverlapHeroEntities(TheHero: FlxBasic,
TheEntity: FlxBasic){
        // guardamos en una variable la clase
del objeto TheEntity
        var entityClass =
Type.getClass(TheEntity);
        // si es un monstruo...
        if(entityClass == Monster)
        {
            // convertimos la referencia tipo
FlxBasic a otra de tipo Hero
            var theHero: Hero = cast(TheHero,
Hero);
            theHero.receiveDamage();
            updateEnergyText();
            if(theHero.energy == 0)
            {
                theHero.kill();
                board.showMessage("You've been
killed... :(");
            }
        }
        // si colisionamos con la espada...
        else if(entityClass == Sword)
        {
            TheEntity.kill();
            board.showMessage("You've found a
sword!");
        }
    }

    private function updateEnergyText()
    {
        energyText.text = "Energy  ";
        for(i in 0...hero.energy)
        {
```

```
                energyText.text += "| ";
            }
        }
    }
```

Fig. 1: Fragmento del archivo PlayState.hx del ejemplo analizado

Como dijimos anteriormente, el método *overlap()* también admite grupos como parámetros para saber si existe solapamiento de algún objeto con alguno de sus miembros. Como puede verse en el código de la función update() del código de la *Fig. 1*, se realiza una llamada a *overlap()* pasando el grupo de enemigos, lo cual basta para detectar solapamiento con cualquiera de ellos. Es importante destacar que, cuando el callback de colisión sea llamado, no recibirá como segundo parámetro al grupo de enemigos, sino al enemigo específico contra el que se detectó la colision. En caso de que existiese una colisión simultánea con más de un enemigo, la función será llamada una vez por cada enemigo.

A diferencia del ejemplo del tutorial anterior, en éste caso ambas llamadas a *overlap()* especifican el mismo callback de colisión. Será necesario, entonces, encontrar una manera diferenciar si se ha colisionado contra un enemigo o contra la espada. Recordemos que los objetos son pasados al callback como referencias de tipo **FlxBasic**. Sin embargo, haXe es capaz de obtener el tipo verdadero de la referencia utilizando el método estático *getClass()* de la clase **Type**. Con ésta nueva información, es posible comparar el tipo real de la referencia con el de los objetos que nos interesa interceptar (en éste caso **Sword** y **Enemy**).

Por otro lado, aunque sea posible conocer el tipo real de los parámetros pasados a *overlap()*, en muchos casos será necesario realizar una conversión efectiva a dicho tipo. Por ejemplo, en éste caso, hemos agregado a la clase **Hero** el método *receiveDamage()* para que pierda energía al colisionar con un enemigo. Es obvio que no es posible invocar a dicho método con el parámetro *TheHero* que fue pasado al callback ya que el mismo es de tipo **FlxBasic**. Para realizar dicha conversión, se emplea el método *cast()*, que recibe el objeto que deseamos convertir y la clase de destino. En éste caso, también se podría haber utilizado el atributo *hero* para invocar a *receiveDamage* y evitar así hacer el cast.

Una de las características muy útiles del lenguaje de programación es la inferencia de tipos. Su uso se ejemplifica en la primer línea del método *onOverlapHeroEntities()*. Al declarar

una variable, si la misma es inicializada en la misma línea en que se la declara, no es necesario especificar su tipo ya que haXe será capaz de inferirlo o adivinarlo a partir del valor con el que se trate de inicializarla.

El ejemplo que hemos desarrollado puede finalmente observarse en la *Fig. 2*.

Fig. 2: Captura del ejemplo analizado a lo largo del tutorial, utilizar las teclas WASD para mover al personaje

Descargar código del ejemplo

En éste tutorial hemos visto como utilizar grupos para simplificar la detección de colisiones. El método *overlap()* admite como parámetros tanto objetos simples como grupos. Es decir que es posible invocarla con dos grupos de objetos y la función callback será invocada para cada par de objetos que se solapen. En la documentación de HaxeFlixel, se recomienda agrupar muchos objetos (incluso poner grupos dentro de otros grupos) para lograr una detección de colisiones más eficiente (cada grupo tiene su propio quadtree).

Nuevamente se recomienda consultar la documentación de HaxeFlixel para conocer mejor las características de la clase **FlxGroup**. También se recomienda hechar una mirada a la documentación de la clase **Type** de haXe, la misma posee algunas funciones que pueden llegar a ser de utilidad. Finalmente, como puede verse en el ejemplo, se ha utilizado la clase **FlxRandom** de HaxeFlixel para generar valores aleatorios. Dicha clase posee métodos muy sencillos de utilizar pero que proveen funcionalidades muy avanzadas en cuanto a generación de valores aleatorios. Su empleo nos será de suma utilidad en nuestros desarrollos futuros, por lo que se recomienda consultar su documentación para estar al tanto de las funciones

disponibles.

Resumen:

- La clase **FlxGroup** representa a un actor capaz de contener a otros actores
- El método *overlap()* puede también trabajar con grupos
- La clase **Type** posee métodos para conocer el tipo de un determinado objeto
- Para optimizar la detección de colisiones, se recomienda agrupar la mayor cantidad de objetos posible para minimizar las llamadas a *overlap()*.

Volver al índice de tutoriales...