UNIVERSIDAD NACIONAL DEL LITORAL
FACULTAD DE INGENIERÍA Y CIENCIAS HÍDRICAS

Tecnicatura en diseño y programación de videojuegos

UNL VIRTUAL

Prototipado rápido

Unidad 6
Partículas

Docente
Ariel Cid

CONTENIDOS

Esta Unidad trata sobre la implementación de las clases necesarias para tener emisores de partículas básicos funcionando en los prototipos. El motivo detrás de esto es la facilidad que brindan dichos emisores para generar *feedback* visual de manera rápida y eficiente; es crucial para muchos juegos que se entienda qué está ocurriendo en pantalla, y las partículas pueden ayudar a resaltar algunas interacciones, como también pueden agilizar la implementación de diversos efectos visuales (humo, fuego, luces, chispas).

1. *Emitter* - declaración y creación de una instancia

La clase base que se utiliza en la Unidad 6 es *Emitter*, y se encarga de generar, manipular y controlar un subconjunto de partículas en pantalla. A lo largo de esta Unidad se trabajará para implementar una explosión de partículas que se dispare cuando el jugador muera, partiendo del proyecto de prueba que se viene iterando clase a clase.

Ante todo, se declaran dos atributos en la clase Player; el primero de la clase *Emitter,* como se mencionó anteriormente, y el segundo de la clase *Graphiclist* (que contendrá todos los gráficos relacionados con la entidad actual).

```
private var playerGraphiclist:Graphiclist;
private var explosionEmitter:Emitter;
```

Deben ser definidos a continuación varios parámetros dentro del constructor de Player para determinar el comportamiento del emisor de partículas.

```
explosionEmitter = new Emitter(new BitmapData(1,1),1,1);
```

Esta línea llama al constructor de un nuevo *Emitter*. El primer parámetro es la imagen de origen (aquella que se desea utilizar para dibujar las partículas); para ello se usa una clase que hasta ahora no se había abordado: *BitmapData*, que permite generar gráficos desde su definición en el código, “dibujando” un conjunto de puntos definidos aquí. En este caso, se está creando un mapa de bits de 1x1 pixeles, de color blanco por default. Los últimos parámetros que se pasan al constructor en la línea antes transcripta son el alto y ancho de cada partícula.

2. Seteo de parámetros de *Emitter*

Luego de crear una nueva instancia de *Emitter* se deben setear otros parámetros pertinentes para que la misma opere de la manera deseada; recuérdese que el potencial y alcance de un emisor de partículas depende en gran parte de su correcta programación.

```
explosionEmitter.newType("explode",[0]);
```

Esta línea define como *"explode"* al tipo de partícula que se encuentra en el primer *frame* de *explosionEmitter* (el cuadro número cero), lo cual opera de manera similar a el atributo *type* de las entidades.

```
explosionEmitter.setAlpha("explode",1,0);
```

Aquí se setea el rango de *alpha* que se desea recorrer a lo largo del tiempo de vida de cada partícula. El primer parámetro indica el tipo al cual se le setea este parámetro, mientras el segundo y el tercero indican respectivamente el valor inicial y final que tendrá. Puede agregarse como cuarto parámetro algún tipo de *easing* para el recorrido entre los valores.

```
explosionEmitter.setMotion("explode",   0,   100,   3,   360,   -40,   -0.5,
Ease.quadOut);
```

En esta línea se define el movimiento que debería tener cada una de las partículas emitidas por *explosionEmitter*. Los parámetros que necesita este método son:

- El tipo de partícula afectada.
- El ángulo mínimo de emisión (en grados).
- La distancia mínima de recorrido.
- El tiempo mínimo de vida.
- El rango máximo del ángulo de emisión. El valor del ángulo de cada partícula será aleatorio, con rango desde el mínimo definido anteriormente hasta el mínimo más el valor definido en este parámetro.
- El rango máximo de la distancia de recorrido. Al igual que el anterior, el valor de la distancia de cada partícula será aleatorio, con rango desde el mínimo definido anteriormente, hasta el mínimo más el valor definido en este parámetro.
- El rango máximo del tiempo de vida. También el valor del tiempo de vida de cada partícula será aleatorio, con rango desde el mínimo definido anteriormente, hasta el mínimo más el valor definido en este parámetro.
- *Easing*, si es que se utiliza alguno.

Por último, conviene definir si las coordenadas para la emisión de la partícula son absolutas (o sea, coordenadas de la pantalla) o relativas (o sea, con el cero de coordenadas definido por esta entidad).

```
explosionEmitter.relative = false;
```

Para la definición de los gráficos de la entidad se han modificado líneas de código para adaptarlas a la nueva estructura visual que se está implementando. Se carga en la lista de gráficos a la animación del jugador junto al emisor de partículas, para luego pasar esta lista al constructor de *Player*.

```
playerGraphiclist = new Graphiclist(playerAnim, explosionEmitter);
super(px, py, playerGraphiclist);
```

3. Llamada a la emisión de partículas

Dentro del método definido en unidades anteriores, *Player.die*, se colocan las líneas que serán necesarias para la emisión de partículas cuando el jugador muera.

```
this.collidable = false;

explosionEmitter.setColor("explode", 0xFFFFFF, 0xFFFFFF);

for (var i:uint = 0; i < 200; i++)
explosionEmitter.emit("explode", x + width / 2, y + height / 2);
```

Este segmento es bastante autoexplicativo: se vuelve al jugador incolisionable (para evitar generar varias veces las partículas), se define el color de la explosión, y luego se emiten doscientas partículas del tipo *explode*, con coordenadas iguales al centro de la entidad. Resta ahora adaptar el código del *update* del jugador para que logre esperar a que todas las partículas sean borradas antes de permitir reiniciar el mundo.

```
if (playerAnim.currentAnim == "death" && playerAnim.complete)
{
        this.playerAnim.visible = false;
        if(explosionEmitter.particleCount == 0)
        {
                var gWorld:GameWorld = FP.world as GameWorld;
                gWorld.gameReboot();
        }
}
```