UNIVERSIDAD NACIONAL DEL LITORAL
FACULTAD DE INGENIERÍA Y CIENCIAS HÍDRICAS

Tecnicatura en diseño y programación de videojuegos

UNL VIRTUAL

Prototipado rápido

Unidad 4
Sonidos y animaciones

Docente
Ariel Cid

CONTENIDOS

La Unidad 3 tenía el objetivo de brindar el conjunto de conocimientos necesarios para la elaboración de un prototipo básico. En esta Unidad y en las siguientes se verán temas que nutren el repertorio de técnicas disponibles para el desarrollo de diferentes mecánicas y *features*.

1. Sonido

Un tópico que es relativamente sencillo, pero no por eso menor, es el manejo de sonidos dentro de Flashpunk. Si bien no difiere mucho de la inclusión de *assets* gráficos, es conveniente dedicar un breve apartado a esta herramienta.

Para poder reproducir audio en un juego de Flashpunk se usa la clase *Sfx*, que tiene métodos que facilitan el manejo de los assets de sonido en el juego. Por ejemplo, *play* reproduce el sonido una sola vez mientras *loop* hace que se reproduzca desde el inicio cada vez que llega al final del archivo (lo cual resulta muy útil para la reproducción de música fondo). También se dispone de un atributo llamado *volume*, que permitirá ajustar cuán alto se desea reproducir dicho sonido.

Se procede entonces a cargar un sonido en la clase *Enemy*, para que se reproduzca cada vez que el jugador colisione con una entidad enemiga. En los atributos de dicha clase se agrega el siguiente código:

```
[Embed(source = "boom.mp3")]
private const BOOM_MP3:Class;
private var boomSfx:Sfx;
```

Luego, en el constructor de *Enemy*, se crea una nueva instancia de *Sfx* con la ruta antes indicada.

```
boomSfx = new Sfx(BOOM_MP3);
```

Por último, dentro de la detección de colisión que se creó en la Unidad 3, se incluye el código para reproducir dicho sonido.

```
if (this.collide("Player", this.x, this.y))
{
        boomSfx.play();
        FP.world = new GameWorld();
}
```

Cabe aclarar que sólo una instancia de *Sfx* puede estar en reproducción, es decir, no es posible ejecutar otro sonido en simultáneo; al darle *play* o *loop* al que ya está siendo ejecutado éste se detiene y se vuelve a iniciar.

2. Animaciones

Una de las maneras más sencillas de implementar *sprites* animados en *Flashpunk* es mediante de una clase que trae su librería: *Spritemap*. Esta clase toma una imagen y la divide en una matriz de elementos de igual tamaño, a los cuales entenderá como *frames* de animación. Por supuesto, hay que explicitarle la cantidad de filas y columnas que debe contener esa grilla, como también qué elementos corresponden a qué animación. Esto facilitará luego la llamada a reproducción de dicha animación, a la cual se accederá tan sólo con un nombre.

Para comenzar a implementar esta clase, se agregan los siguientes atributos a Player con el objetivo de reemplazar al viejo *playerImage*:

```
[Embed(source = "playerAnim.png")]
private const PLAYER_ANIM:Class;
private var playerAnim:Spritemap;
```

El paso siguiente es indicarle cómo se desea que esté dividida dicha imagen en columnas y filas. Para la creación de una nueva instancia de *Spritemap*, se explicita entonces la ruta de la imagen a usar, junto al ancho y el alto de una celda o fotograma (en pixels).

```
playerAnim = new Spritemap(PLAYER_ANIM, 16, 16);
```

Luego resta otorgar nombre a las animaciones mediante el método *add*, y delimitar los fotogramas que las componen. El conteo de fotogramas dentro de la matriz se realiza siempre de izquierda a derecha y de arriba hacia abajo (como el orden de lectura occidental). Otro dato requerido por el método *add* es la cantidad de FPS a las que corre la animación (es decir, su velocidad) y si la animación *loopea* al terminar la misma.

Como paso final se pasa dicho *Spritemap* al constructor de la clase padre (*Entity*) para que sea asignado como gráfico de esa entidad.

```
playerAnim.add("fly", [0, 1, 2, 3], 30, true);
playerAnim.add("death", [4, 5, 6, 7], 30, false);
super(px, py, playerAnim);
```

Desde ese momento ya se pueden utilizar las animaciones de *fly* y *death*. Para esto, el código que se encuentra al final del constructor de *Player* llamará a la primera de ellas.

```
playerAnim.play("fly");
```

En el código subido a la Unidad 4 de la plataforma virtual se puede ver cómo está implementada la animación *death* para que se ejecute cuando hay una colisión con un enemigo.

Hay varios cambios adicionales que deben hacerse al código para que siga funcionando como lo hacía con una imagen estática, y que también están en el proyecto de esta Unidad. Por ejemplo, se deben ajustar las líneas en las que se hace mención del ancho o alto de la imagen del jugador ya que ahora van a contener un mapa de *sprites* mucho más grande.