UNIVERSIDAD NACIONAL DEL LITORAL
FACULTAD DE INGENIERÍA Y CIENCIAS HÍDRICAS

Tecnicatura en diseño y programación de videojuegos

UNL VIRTUAL

Modelos y algoritmos para videojuegos I

Unidad 9

Docentes
Sebastián Rojas Fredini
Patricia Schapschuk

CONTENIDOS

9. MÁS FUERTE QUE NO TE ESCUCHO

9.1. De cómo lo interesante puede resultar inexpresivo

Hasta el momento hemos visto conceptos y herramientas básicas necesarias para comenzar a desarrollar nuestros videojuegos. Hemos abarcado el apartado visual y la manipulación de eventos, tanto de usuario como de sistema.

Si ponemos todo esto en la batidora y la prendemos por un rato, al volcar el contenido probablemente veamos un juego interesante, pero totalmente mudo.

El sonido es un componente crucial en el desarrollo de videojuegos. Cuando hablamos de sonido, nos referimos a efectos especiales como los pasos de un personaje, el sonido de dos espadas al chocar y la música que nos ayuda a sumergirnos en ese mundo que intentamos crear.

Generalmente, los efectos de sonido y la música se suelen tratar de forma separada debido a las características que los diferencian entre si, a saber:

- *Efectos de sonido*
    - Corta duración en el tiempo.
    - Un mismo efecto de sonido puede ser reproducido muchas veces en períodos cortos de tiempo.
    - La mayoría de los efectos de sonido se repiten a lo largo de todo el juego.
    - La reproducción se encuentra ligada a eventos del juego.
- *Música*
    - Duración considerable en el tiempo (mucho mayor que un efecto).
    - La música suele depender de los niveles o situaciones.
    - El tamaño de los archivos de música a veces son demasiado grandes para almacenarlos completamente en memoria, por lo que se utilizan técnicas de streaming, es decir, se cargan en memoria a medida que se van necesitando.
    - Es común que el mismo fragmento de música se repita constantemente a lo largo de un nivel.
    - Debido a la longitud de los temas, éstos suelen comprimirse en formatos mp3, ogg, etc.

Si nuestro videojuego fuese privado de este aspecto, el resultado sería mucho menos inmersivo. Imaginen un *Resident Evil* donde no puedan escuchar los pasos del personaje ni el balbuceo de los zombies;o un *Final Fantasy VIII* sin *Eyes on me* flotando en el espacio.

El sonido transmite muchas sensaciones muy difíciles de transmitir de otra manera. Por ello siempre debe ser un aspecto tan cuidado como el visual.

Veamos, ahora, cómo hacemos para reproducir sonidos en SFML.

9.2. ¿Cuál era el play?

Para trabajar con sonidos en SFML, primero debemos incluir el siguiente encabezado, que contiene las clases *SoundBuffer* y *Sound*.:

```
#include <SFML/Audio.hpp>
```

Estas clases funcionan exactamente de la misma manera que *Image* y *Sprite*. Es decir, *SoundBuffer* contiene los datos del sonido, mientras que *Sound* es una instancia de dicho *Buffer*.

Emplearemos *SoundBuffer* para cargar los datos de un sonido desde un medio de almacenamiento, generalmente un archivo en disco. Para esto, utilizaremos la función *LoadFromFile,* miembro de *SoundBuffer,* y le pasaremos como argumento el path del archivo que contiene el sonido:

```
sf::SoundBuffer Buffer;

if
(!Buffer.LoadFromFile("sound.wav"))

{

    // Error...

}
```

Esta función nos devuelve un *boolean,* indicando si pudo cargar el archivo con éxito. Si la acción resultó exitosa, tendremos la información del sonido en memoria (como si fuera una imagen) y podremos empezar a crear instancias del sonido (como si fueran los sprites).

Cabe aclarar que varios sonidos pueden utilizar el mismo buffer. Para ser más gráficos, podemos pensar como al buffer como si fuera una clase y al sonido, un objeto, instancia de dicha clase.

Una vez cargado el buffer, debemos declarar objetos sound que lo utilicen. Estos objetos usan los datos del buffer y nos permiten setear algunas propiedades que controlan cómo ese buffer es reproducido.

La propiedad más evidente es el volumen del buffer. Esto implica que podamos tener dos sonidos que reproduzcan, por ejemplo, el mismo buffer, pero con distinto volumen. Esta flexibilidad es posible por la división entre sonidos y buffers.

Para declarar un objeto sound y setear el buffer, hacemos lo siguiente:

```
sf::Sound Sound;

Sound.SetBuffer(Buffer);
```

Donde Buffer es el definido anteriormente.

De esta manera, ya tenemos un sonido asociado a un origen de datos de sonido (*SoundBuffer)* y ya podemos configurar cómo deseamos que el sonido reproduzca dicho buffer.

Veamos, ahora, dos propiedades que seguramente utilizaremos:

- *Sound.SetLoop(bool)* Permite establecer si deseamos que el sonido se repita ni bien termine.

- *Sound.SetVolume(float)* Permite setear el volumen con el cual el buffer debe ser reproducido.

La documentación de SFML contiene una referencia a todas las propiedades que podemos setear para los sonidos.

Una vez configurado, ya podemos invocar los métodos para reproducir y pausar sonidos: *Play()* y *Stop():*

```
Sound.Play();

Sound.Stop();
```

Los buffers son recursos que demoran en crearse, ocupan bastante memoria y son costosos para copiarlos, por lo que deben ser cuidadosos a la hora de trabajar con los mismos.

9.3. Música para mis oídos

Como dijimos anteriormente, la música es como un sonido, pero mucho más largo, por lo que no podemos tratarlos de la misma manera.

Para darnos una idea, cinco minutos de música en calidad CD nos consume aproximadamente 50 Mb de memoria, lo cual es demasiado para un recurso.

Es por ello que un tema musical nunca se carga completamente en memoria, sino que se va cargando a medida que es reproducido. De esta manera, optimizaremos el uso de memoria.

SFML provee una clase que nos hace sencilla esta tarea. La misma se denomina *Music* y provee funcionalidad para abrir un archivo, cargarlo dinámicamente a medida que avanza la reproducción y reproducirlo. El uso es muy sencillo y muy parecido a la clase *Sound*.

Para declarar y abrir un archivo:

```
sf::Music musica
if
(!musica.OpenFromFile("music.ogg"))
{
    // Error...
}
```

Y luego, para reproducirlo, detenerlo o pausarlo:

```
musica.Play();
musica.Pause();
musica.Stop();
```

Existen muchos conceptos y técnicas más avanzadas que serán desarrolladas en otras materias. No obstante, lo que hemos visto nos alcanza para programar los videojuegos que proponemos en esta asignatura.

BIBLIOGRAFÍA

Alonso, Marcelo; Finn, Edward. *Física: Vol. I: Mecánica*, Fondo Educativo Interamericano, 1970.

Altman, Silvia; Comparatone, Claudia; Kurzrok, Liliana. *Matemática/ Funciones*. Buenos Aires, Editorial Longseller, 2002.

Botto, Juan; González, Nélida; Muñoz, Juan C. *Fís Física*. Buenos Aires, Editorial tinta fresca, 2007.

Díaz Lozano, María Elina. *Elementos de Matemática*. Apuntes de matemática para las carreras de Tecnicaturas a distancia de UNL, Santa Fe, 2007.

Gettys, Edward; Séller, Frederick. J.; Skove, Malcolm. *Física Clásica y Moderna*. Madrid, McGraw-Hill Inc., 1991.

Lemarchand, Guillermo; Navas, Claudio; Negroti, Pablo; Rodriguez Usé, Ma. Gabriela; Vásquez, Stella M. *Física Activa*. Buenos Aires, Editorial Puerto de Palos, 2001.

Sears Francis; Zemansky, Mark; Young, Hugh; Freedman, Roger. *Física Universitaria*. Vol. 1, Addison Wesley Longman, 1998.

SFML Documentation, http://www.sfml-dev.org/documentation/

Stroustrup, Bjarne. The Design and Evolution of C++. Addison Wesley, Reading, MA. 1994.

Stroustrup, Bjarne. *TheC++ Programming Language.* Addison Wesley Longman, Reading, MA. 1997. 3° ed.

Wikipedia, http://www.wikipedia.org