Alumno: Rubén Garrido Valdepeñas
Asignatura: Efectos visuales y sonoros
Curso: 2022-2023
Profesor: Rafel Pérez Vidal

PEC 3: La cueva de los condenados

El movimiento del jugador lo he dividido en dos partes. Por un lado está la rotación sobre sí mismo y por otro el movimiento. Lo he dividido así porque rotar sobre sí mismo tiene una afectación al sonido que se está reproduciendo y el movimiento del jugador puede desencadenar diferentes eventos (movimiento del monstruo, reproducción de sonidos extras...).
Para representar la dirección en la que está mirando el jugador tengo un enum con todas las posibilidades. Aparte, tengo un valor entero para saber la dirección actual del jugador.
Cada vez que se pulsan las teclas “A”, ”D”, “<-” y “->” se actualiza la dirección del jugador mediante un switch usando el enum. Esto provoca que se tengan que actualizar los sonidos del monstruo y la cascada. Concretamente el panning. Esto lo hago mediante un método que mira la posición del jugador con la posición del orígen del sonido. Aquí dependiendo de la posición del jugador y la dirección se ajusta el panning. En mi caso lo he reducido a cuatro casos:

- El jugador está mirando arriba: Cuando el jugador está mirando arriba miro si la posición Y (columna) del origen del sonido es menor que la del jugador. En este caso el orígen del sonido se encuentra a la izquierda del jugador por lo que el panning del lado derecho se tiene que reducir. Si la Y es superior al jugador el orígen está en la derecha. Finalmente se puede dar el caso de que el valor de Y sea igual al del jugador. En este caso el sonido se escucha por igual en los dos lados.

```
case Direction::Up:
if (playerPositionY > posY) {
        // Left side
        directions[1] = 128;
}
else if (playerPositionY < posY) {
        // Right side
        directions[0] = 128;
}
break;
```

- El jugador está mirando abajo: El comportamiento es el mismo que arriba pero invirtiendo las condiciones.
- El jugador está mirando a la izquierda: El funcionamiento es igual al primero pero se miran las posiciones X (fila).
- El jugador está mirando abajo: El comportamiento es el mismo que la izquierda pero invirtiendo las condiciones.

El valor del panning siempre empieza en 255 y dependiendo de los casos anteriores el valor se reduce en la mitad.

Cuando el jugador pulsa la “W” o la flecha de dirección hacía arriba se produce el movimiento del jugador. Lo primero es saber la nueva posición del jugador. Para esto utilizo la posición actual del jugador y calculo el offset utilizando la dirección.
Una vez que tengo esto miro que hay en la nueva posición. Si no hay nada, actualizo la posición del jugador sumándole los offsets. Seguidamente tengo que actualizar la distancia de los sonidos, porque puede que nos estemos acercando al origen de la fuente o nos estemos alejando.
Como el mapa que tengo tiene una estructura matricial lo que hago es calcular la distancia entre los dos puntos y multiplicar ese valor por un índice de distancia (teniendo en cuenta que el valor no debe superar el 255).

```
(abs(playerPositionX - posX) * distanceFactor + abs(playerPositionY - posY) * distanceFactor);
```

Finalmente queda reproducir el sonido de los pasos del jugador. En este caso el sonido se reproduce con la misma intensidad por la izquierda y la derecha y la distancia es cero.

El otro caso que se puede producir es que el jugador se coche con una pared. Cuando se choca con la pared se disparan varios eventos. El primero es que se tiene que reproducir el sonido de choque con la pared. Este tiene las mismas características que los pasos. Seguidamente hay que activar al monstruo. Este evento empieza con la parada del sonido del ronquido del monstruo.

```
Mix_HaltChannel(snoring);
```

Seguidamente hay que mover al monstruo. Para esto busco las posibles direcciones a las que se puede mover, descartando las posiciones que tienen una pared.

```
if (environment[monsterPositionX - 1][monsterPositionY] == 0) {
        possiblePositions[counter][0] = monsterPositionX - 1;
        possiblePositions[counter][1] = monsterPositionY;
        counter++;
}
```

Seguidamente cojo una opción aleatoria. Una vez que se ha movido reproduzco el sonido de movimiento del monstruo añadiendo la distancia y el panning al jugador. Seguidamente miro si el monstruo está en la misma posición del jugador se acaba el juego.

```
(playerPositionY == monsterPositionY && playerPositionX == monsterPositionX)
```

Si se acaba el juego paro todos los sonidos que están en loop. Reproduzco el sonido de muerte del jugador y al cabo de un rato pongo la música de derrota.

```
Mix_PlayChannel(death, audios[death], 0);
SDL_Delay(2500);
Mix_HaltChannel(gameOver);
Mix_PlayChannel(gameOver, audios[gameOver], 0);
Mix_Pause(waterFall);
Mix_Pause(snoring);
```

Finalmente queda explicar tres cosas más. La inicialización del sld_mixer, el loop de los sonidos y la limpieza.
En mi caso he utilizado configuraciones por defecto, 44100 hercios y dos canales (modo estéreo). Aunque no me hace falta indico que aloje 128 canales.

```
//Sound audio active
Mix_OpenAudio(44100, MIX_DEFAULT_FORMAT, 2, 1024);
// Asign audio chanel numbers
Mix_AllocateChannels(128);
```

Con esto ya se pueden cargar los archivos de audio:

```
loadSound = Mix_LoadWAV("Assets/waterfall.wav");
audios.push_back(loadSound);
```

Para el loop de los sonidos del ronquido y de la cascada he utilizado un enfoque diferente. Se que se puede hacer un PlayChannel con el último valor a -1 para que se reproduzca en loop.

```
Mix_PlayChannel(waterFall, audios[waterFall], -1);
```

Finalmente me queda liberar todos los audios y cerrar el SDL_Mixer

```
// Free audio
for (int i = 0; i < audios.size(); ++i)
{
	Mix_FreeChunk(audios[i]);
}
//Quit SDL subsystems
Mix_CloseAudio();
```