PROYECTO DE SONIDO EN VIDEOJUEGOS

Resonorización de Proyectos 3 con FMOD

Gonzalo Sanz Lastra y Jorge Rodríguez García

Proyecto del año pasado:

El resultado de Proyectos 3 del año pasado es **Holy Spoons**, hecho con Ogre, Physx e irrklang como motor de audio. **Su arquitectura está basada en componentes y entidades orientadas a datos**. Está ideado de forma que las entidades (y sus componentes) del juego sean modificables desde el exterior del código mediante json. De esta forma, se pueden configurar nuevos niveles fácilmente añadiendo o eliminando nuevas entidades o modificando sus componentes o parámetros.

Es un **FPS** (first person shooter) por rondas en el que tienes que acabar con varias **oleadas** de enemigos para pasar de ronda. Cada oleada tiene un número de enemigos de varios tipos, que aumenta dependiendo de la ronda en la que estés.
El juego consta de **tres mapas: volcán, océano y cielo**. Cada mapa tiene una disposición diferente de los enemigos y de las plataformas sobre las que se mueve el jugador.

El proyecto de sonido consiste en **resonorizar Holy Spoons, sustituyendo irrklang por FMOD**, y además aprovechar la capacidad de este último para **mejorar el apartado sonoro** ya existente del juego, añadiendo funcionalidades de todos los campos vistos de FMOD en clase: **direccionalidad, reverbs y geometrías**.

Objetivos:

- Reemplazar irrklang por FMOD lowlevel:

Nuestra estructura con **irrklang** era la siguiente: teníamos una interfaz **SoundManager** que controlaba todos los sonidos que había en el juego. Los reproducía, los pausaba, los creaba etc. Al no haber canales, todos los sonidos se guardaban en dos listas: sonidos 2D y sonidos 3D, ya que irrklang los distinguía por clases distintas (irrSound2D y irrSound3D). Además del gestor, teníamos un componente **SoundEmitterComponent**. Las entidades que tuvieran este componente podían reproducir sonidos en 2D o en 3D, éstos últimos con posicionalidad.

Ahora bien, con la nueva implementación en **FMOD**, todas estas cosas las hace él por debajo. FMOD tiene referencia a todos los sonidos, a sus parámetros y a sus métodos.
La clase **SoundManager** sigue existiendo, pero esta vez sólo gestiona el system de FMOD (lo crea, actualiza y proporciona acceso a él), y también pausa y reproduce grupos de canales.
SoundEmitterComponent también ha cambiado, ahora tiene referencia a un canal de FMOD y a un sonido. Desde json se pueden seleccionar varios parámetros como el modo, si es tridimensional o no, volumen, pitch etc. Tiene métodos para reproducir su sonido y pararlo. Sirve como punto de partida y componente con la funcionalidad sonora “estándar”.

FMOD también da soporte para otros efectos más complicados como las geometrías, la direccionalidad de los sonidos, y reverberaciones, cosas que con irrklang sería imposible.
Además, muchos bugs que ocurrían con irrklang se han solucionado al introducir FMOD, como reproducciones simultáneas de todos los sonidos cargados al inicio de cada escena, no existía la posibilidad de pausar un sonido y reanudarlo posteriormente, o que para reproducir un mismo sonido varias veces había que cargarlo varias veces, etc.

- Mejorar el apartado sonoro del proyecto

Hemos aplicado los conocimientos adquiridos en la **edición de audio con Audacity** a los efectos sonoros que ya estaban presentes en el juego:
Los sonidos destinados a reproducirse en bucle han sido retocados mediante la técnica de **loop con fade in y fade out**, y muchos de ellos han sido **amplificados, normalizados** o se les han añadido **nuevos efectos** para adaptarse a su función dentro del juego (ej. ecualización de agudos en los sonidos emitidos por los enemigos para ser más identificables alrededor del jugador, efecto wahwah para la oscilación del ovni, etc.).

Además, se han añadido **nuevos sonidos** que antes no estaban presentes en el juego, como por ejemplo: efecto de oscilación durante el vuelo del ovni, disparo del enemigo

francotirador, sonido de los portales, sonido ambiente de agua, lava y viento (uno por cada tipo de mapa).

- Rangos de audición y direccionalidad de los objetos móviles

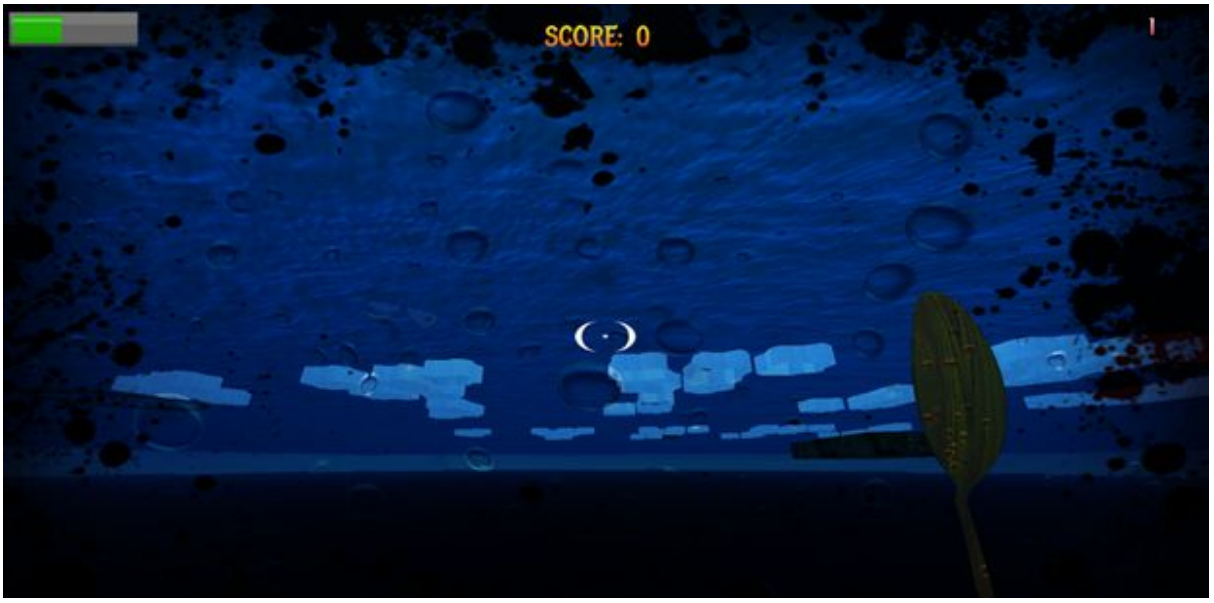
Han sido añadidos a entidades y eventos del juego que produzcan sonidos posicionales:

Monos, enemigos estándar del juego, que se mueven continuamente alrededor del jugador. La atenuación con la distancia y la direccionalidad ayudan a tener un rápido mapa mental de la posición de los múltiples enemigos que rodean al jugador, sin necesidad de estar viendo a éstos en todo momento.
Portales, que funcionan como “nidos” de nacimiento de los enemigos. Sólo sonarán los portales activos en ese momento con un rango estrecho para no abrumar al jugador.
Ovnis, enemigos más poderosos que los estándar. Tienen su direccionalidad apuntando hacia abajo, de forma que puedan ser oídos a gran distancia para tener una idea de su posición (ya que en muchas ocasiones son más difíciles de ver, al estar por encima del jugador), pero con un efecto mucho mayor al estar justo bajo ellos, momento en el que se busca un efecto de “abducción”.

Esta funcionalidad queda recogida en el componente **DirectionalSoundComponent**, que hereda del componente SoundEmitterComponent. Las entidades que tengan este componente emitirán sonidos 3D, pero además con conos de direccionalidad y rangos de audición, cuyos parámetros pueden ser configurados desde json.

- Añadir reverbs dependiendo del mapa

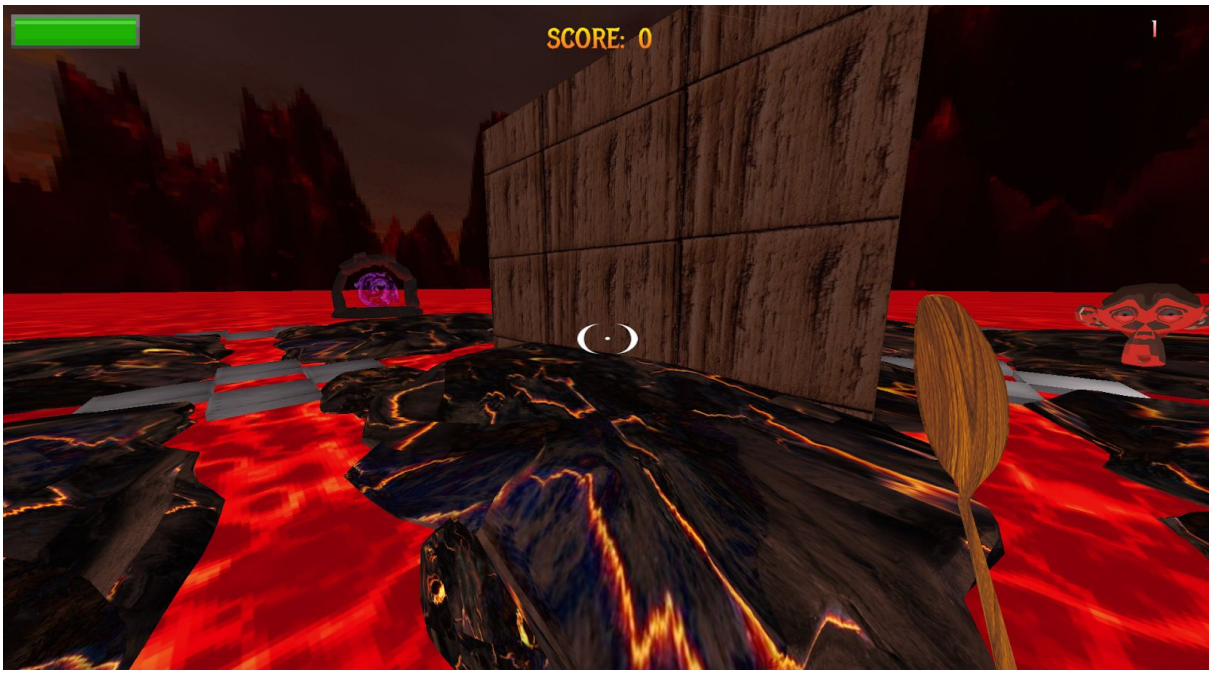
Cada uno de los mapas (lava, agua, cielo) tiene ahora un **sonido ambiental característico** que suena cuando el jugador cae de alguna plataforma. A esto se le suma una **reverb personalizada** a partir de la FMOD_PRESET_UNDERWATER dada por FMOD, que influye sobre los demás sonidos de la escena.
De esta forma, al caer a la lava, agua o al vacío, sonarán los sonidos correspondientes y se activará la reverb que distorsiona todos los demás, dando **sensación de estar bajo un líquido ahogándote o cayendo al vacío**. Este efecto conjunto se anula si el jugador vuelve a la superficie, ya que en los mapas de lava y agua se puede salir a flote de nuevo, momento en el que todos los sonidos de la escena volverán a escucharse como antes.



Esta funcionalidad queda recogida en el componente **ReverbComponent**. Las entidades que tengan este componente establecerán una Reverb3D en su posición, cuyos parámetros pueden ser configurados desde json.

- Modificar geometría de los mapas para añadir paredes de oclusión

Hemos añadido posibilidad de añadir entidades en cualquier mapa que funcionan como muros que ocluyen el sonido. Puedes ponerlas del tamaño que quieras en la posición que quieras y con la orientación que quieras, todo desde json. Actualmente hay una pared central en el mapa del volcán y otra en el mapa del océano. Se puede utilizar cualquier enemigo que emita un sonido tridimensional para probar esto, poniendo la pared entre el jugador y el enemigo en cuestión.



Esta funcionalidad queda recogida en el componente **GeometryComponent.** Las entidades que tengan este componente establecerán un muro de oclusión en su posición, cuyos parámetros pueden ser configurados desde json.