**Integración e investigación del middleware WWISE**

Nuestro proyecto ha consistido primordialmente en integrar el middleware Wwise en un juego realizado el año pasado en Noruega durante nuestra estancia Erasmus.

El principal objetivo era comprobar si realmente Fmod y Wwise son tan diferentes como en algunas ocasiones parecen o si nos aportan funcionalidades similares.

Por otro lado, también queríamos familiarizarnos más con Wwise y aprender a manejarnos con él, ya que es realmente usado en grandes empresas para implementar el audio en sus videojuegos. Así, hemos enfocado la mayor parte del proyecto a usar su editor e integrar diferentes efectos con él.

Sin embargo, también hemos usado su API para realizar ciertas acciones, como reproducir eventos en ciertos momentos puntuales o generar el efecto de oclusión.

**Implementaciones**

**Integración de sonidos**

Como primer objetivo nos propusimos integrar todos los sonidos que estaban en el juego con Wwise.

Para esto, tuvimos que crear los diferentes eventos y soundbanks correspondientes. Acto seguido, generamos las soundbanks en el proyecto para poder trabajar con ellos.

Para llamar a estos eventos usamos la instrucción de la API AkSoundEngine.postEvent(), ya que eso nos permite tener un control completo acerca de dónde queremos lanzar los sonidos.

**Posicionamiento 3D y Paneo**

Para dar un toque más realista al juego, hemos implementado varios sonidos con posicionamiento 3D con el fin de conseguir oclusión y paneo.

Para ello, simplemente hemos convertido los audios en audios 3D desde el editor de Wwise y automáticamente se aplica paneo sobre esos clips, pero no oclusión.

**El problema de la atenuación**

Como hemos comentado anteriormente, el posicionamiento 3D no ofrece la atenuación de forma automática, así que tuvimos que implementarla nosotros.

En una primera aproximación de aplicar el efecto a los gameObjects, creamos un shareSet, que es un efecto de atenuación creado por nosotros en el que vamos modificando el volumen en función de la distancia. Este shareSet puede ser aplicado y compartido a todos los sonidos del proyecto, lo que realmente nos pareció bastante útil, ya que nos permite definir una atenuación común y aplicarla de forma sencilla a los audios que queramos.

Sin embargo, como sugerencia del profesor, se nos pidió que investigásemos un poco el efecto para ver si había alguna forma de aplicar esta atenuación de manera automática, sin tener que definirla como un shareSet. Y parece ser que no la hay, hemos estado leyendo e investigando la documentación y en ella todo lo relacionado con la atenuación te lleva a crear estas instancias de ShareSets para aplicarlas luego a los diferentes sonidos.

Adjuntamos un enlace al apartado de atenuación en la documentación.

<https://www.audiokinetic.com/fr/library/edge/?source=Help&id=applying_distance_based_attenuation>

En este aspecto, Fmod si que es más cómodo, ya que con hacer la fuente emisora un objeto 3D se aplica atenuación y posicionamiento 3D.

**Selección aleatoria de sonidos**

Otra cosa que decidimos añadir fue la selección aleatoria de sonidos. La idea era simular que hay varios caballos en el establo emitiendo sonidos, de tal forma que el jugador pueda escucharlos cuando está cerca. Esta implementación mezclaba un poco todo lo que habíamos hecho anteriormente, ya que necesitábamos sonidos 3D y atenuación, además de que añadía la selección aleatoria de sonidos, que era algo interesante puesto que la habíamos visto cómo hacerlo en Fmod Studio.

Para esto descargamos una muestra con diferentes sonidos de caballos y la recortamos en el propio editor de Wwise, obteniendo así 5 sonidos de caballos diferentes.

A continuación, creamos un Random Container que va seleccionando y reproduciendo los sonidos de forma aleatoria. Dichos sonidos están ya configurados como sonidos 3D y con un efecto de atenuación aplicado.

En este apartado sí que hemos observado que el editor de Fmod Studio resulta mucho más simple e intuitivo en general. Un ejemplo de ello se da en la claridad a la hora de seleccionar los porcentajes de reproducción de cada sonido.

**Geometrías y oclusión**

Para el efecto de oclusión, hemos optado por una de las propuestas más habituales en cuanto a programación se refiere a la hora de afrontar este efecto, que consiste en lanzar un rayo desde el objeto emisor hasta el jugador y comprobar si entre ambos existe otro objeto, y en caso afirmativo, llamar al método para aplicar la oclusión.

Para lograr el efecto de oclusión, hemos probado con dos acercamientos diferentes.

El primero de ellos fue crear dos variables de juego, una para el volumen y otra para el Low Pass Filter. La idea fue ajustar ambas variables para que produjesen un efecto de oclusión adecuado al contexto del juego. De esta forma, cuando el rayo detectase un objeto entre el listener y la fuente sonora, modificamos esos parámetros llamando a la API.

La segunda forma que hemos logrado implementar ha sido haciendo uso directamente de un método setObjectObstructionOcclusion() proporcionado por la API. Este método resulta mucho más sencillo y natural, ya que nos evita el trabajo de crear las variables de juego mencionadas anteriormente y el efecto logrado es más real.