Sonido en videojuegos

Grado en Desarrollo de Videojuegos

Examen final, enero 2019

Indicaciones generales:

- Los archivos de audio mencionados en los enunciados se encuentran en la carpeta *muestras*.
- El programa debe compilar y funcionar correctamente (si no se implementa alguna de las funcionalidades pedidas no incluir código incorrecto o incompleto).
- Los ejercicios se guardarán en dos carpetas diferentes. Después se comprimirán en un único archivo de la forma *NombreApellido1Apellido2.zip* que se subirá al servidor FTP del laboratorio.

---

1. FMOD. API

En este ejercicio implementaremos algunas opciones básicas de posicionamiento 3D con FMOD. En la carpeta FMOD se proporciona la API, así como la ayuda de FMOD en un archivo .chm. Para facilitar la corrección implementaremos un único proyecto con un único archivo .cpp. Dicho proyecto debe ser autocontenido, es decir, debe incluir las cabeceras y librerías (lib y dll) necesarias en su propia estructura.

Para procesar el input de teclado, utilizaremos dos funciones de la cabecera `conio.h`:

- `_kbhit()`: devuelve *true* si hay pulsación de teclado; *false* en caso contrario.
- `_getch()`: devuelve un caracter con la tecla pulsada.

El archivo *helicoptero.wav* contiene un loop de motor de helicóptero que utilizaremos como emisor en este ejercicio. Implementaremos de manera incremental las siguientes funcionalidades:

- [**1 pt**] Situar el listener en la posición $(0, 0, -1)$, orientado en posición vertical y mirando hacia adelante.

  Cargar la muestra (helicóptero) en loop y asociarla a un canal emisor. Situar dicho emisor en la posición $(0, 0, 4)$, con velocidad nula y reproducir la muestra (en loop). Para ello implementaremos un bucle principal que mantiene la reproducción del canal hasta detectar la pulsación `'q'`, que terminará dicho bucle.

- [**2 pt**] A continuación, extender la funcionalidad del bucle permitiendo mover el listener en el plano (x-z) en saltos de 0.1 unidades con las teclas habituales "asdw", que detectaremos con las funciones `_kbhit()` y `_getch()` mencionadas arriba.

- [**1 pt**] Extender la funcionalidad permitieno subir el pitch del canal con la tecla `'P'` o bajarlo con `'p'`, en saltos de 0.01 unidades.

- [**2 pt**] Incluir una *Reberb3D* con el preset `FMOD_PRESET_CONCERTHALL` y situarla en las coordenadas (2,0,6), con distancias mínima y máxima de 1 y 10, respectivamente. Incluir la opción de activar y desactivar la reverb con las teclas `'R'` y `'r'`, respectivamente.

- **[2 pt]** Utilizando la clase *Geometry* situar un cuadrado de tamaño 4×4 en el plano x-y y centrado en el origen de coordenadas para experimentar la oclusión entre listener y emisor. Incluir la opción de activar la oclusión con las teclas `'O'` y `'o'` respectivamente.

Nota: para facilitar la depuración y corrección del programa escribir en pantalla en cada frame la posición del listener, el pitch, el estado de la reverb (activa o inactiva) y de la oclusión de la geometría.

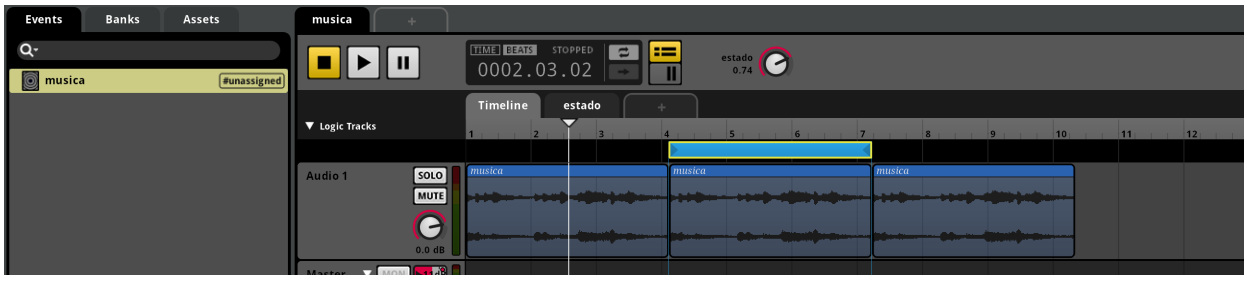
2. FMOD Studio

En este ejercicio utilizaremos el editor de FMOD Studio para crear dos eventos. Comenzaremos creaando un nuevo proyecto en el editor y cargando las muestras *helicoptero.wav* y *musica.wav* como assets del proyecto.

- **[1 pt]** Con *helicoptero.wav* crearemos un evento *rotor*. Añadiremos a dicho evento un parámetro *rpm* con valores en el intervalo [0,1] que permita modificar el pitch de la muestra entre -8 y +8 semitonos (valor *st* mostrado en la pista de automatización correspondiente).

- **[1 pt]** Con *musica.wav* crearemos un nuevo evento *musica* con un parámetro *estado* para gestionar las envolventes (fadein, fadeout) de la muestra. Para ello cargaremos en el *timeline* tres instancias completas de la misma muestra de música, la primera con una envolvente ascendente de volumen (fadein de 0 a 1), la segunda con el volumen al máximo y la tercera con una envolvente descendente (fadeout de 1 a 0).

  A continuación crearemos un parámetro *estado* con valores en el intervalo [0,1] para controlar la reproducción en loop sobre la segunda muestra. Después, sobre el *timeline* añadiremos una *región de loop* condicional sobre la segunda instancia de modo que reproducirá esa instancia en loop mientras que *estado* esté en el intervalo [0,0.5]. Saldrá del loop cuando estado>0.5, alcanzando la tercera instancia con el fadeout de la música. El evento tendrá el siguiente aspecto en el editor (no se muestra la condición de la región de loop):



Guardar el proyecto con los dos eventos *.fspro* y asegurarse de que se abre correctamente en el editor antes de entregarlo.