Sonido en videojuegos

Grado en Desarrollo de Videojuegos

Examen final, enero 2022

Indicaciones generales:

- Los archivos de audio mencionados en los enunciados se encuentran en la carpeta *muestras*.
- **Cada ejercicio se guardará en una carpeta con el número del mismo**.
- Después todas estas carpetas se comprimirán en formato .zip en un único archivo que se llamará **NombreApellido1Apellido2.zip** y se subirá al servidor FTP del laboratorio.

Audacity En los siguientes ejercicios, para evitar problemas de incompatibilidad de versiones, en vez de guardar proyectos de Audacity (*.aup*) se guardarán las pistas como archivos independientes. Para ello:

- Todas las pistas deben comenzar en el instante 0 (rellenar la pista con silencio inicial si la muestra comenzase retrasada en el tiempo).
- Se exportarán las pistas con la opción de menú *Archivo* → *Exportar* → *Exportar múltiple*.
- **Comprobar que al cargar las pistas en Audacity, se obtiene el resultado esperado.**

---

1. **[2 pt]** El archivo *olas.mp3* contiene una pista estéreo con sonido del mar (se aprecia el sonido continuo de oleaje y olas aisladas). Se pide:

   - Realizar un *loop coherente* de unos 30 segundos de duración. Para ello seleccionar una muestra de la longitud adecuada y que **no corte** el sonido de ninguna ola aislada. Después, utilizando la técnica vista en clase (fades al principio y el final), construir el loop. Para ello se hará un *fade-in* de corta duración al principio de la muestra, otro *fade-out* del mismo tamaño al final y después se hará la superposición temporal de ambos fragmentos.

     Para poder ver la construcción del loop exportar el resultado en dos archivos (ambos estéreo):

     - *olas1.wav* con el fragmento de fade-in seguido del resto de la muestra
     - *olas2.wav* únicamente con el fragmento de fade-out, en el lugar correspondiente.

2. **[2 pt]** El archivo *corazon.mp3* contiene una muestra de latidos de corazón. Se pide:

   - Construir un loop coherente (latidos regulares) de unos 4-5 segundos de duración y exportarlo en *corazon1.wav*.
   - En el latido se aprecian dos tipos de pulso intercalados (fuerte-débil). Vamos a generar un efecto estéreo enviando los pulsos fuertes al canal izquierdo y los débiles al derecho. Para ello, dividimos la pista resultante del apartado anterior en dos pistas mono y utilizamos el balance para enviar la primera al canal izquierdo y la segunda al derecho. A continuación silenciaremos los pulsos débiles de de la primera pista y los fuertes de la segunda (puede hacerse de varios modos). Por último, se obtendrá una pista estéreo con las dos pistas resultantes y se exportará el resultado en *corazon2.wav*.

FMOD API En el siguiente ejercicio, el programa pedido debe compilar y funcionar correctamente. Si no se implementa alguna de las funcionalidades pedidas, no incluir código incorrecto o incompleto, puesto que no podrían evaluarse el resto de funcionalidades. La documentación de FMOD (*API User Manual*) está disponible en los ordenadores del laboratorio.

---

3. **[3 pt]** En este ejercicio implementaremos algunas opciones básicas de posicionamiento 3D con FMOD. Para facilitar la corrección se implementará un único proyecto con un único archivo .cpp. Debe ser un **proyecto 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 disponibles en `conio.h`:

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

   Vamos a utilizar *music.ogg* como banda sonora y *corazon.mp3* como emisor 3D que podremos mover en el plano X-Z. Para facilitar la depuración y corrección del programa se implementará un sencillo **renderizado en consola** que muestre posición del emisor y el listener, así como el pitch y el estado de la reverb que se mencionan a continuación.

   Implementaremos de manera incremental las siguientes funcionalidades:

   - Cargar *music.ogg* en modo loop 2D, lanzar su reproducción en el canal *música*, con el volumen a 0.1 y dejarla en reproducción.
   - Situar el listener en la posición $(0, 0, 0)$, orientado en posición vertical y mirando hacia adelante.

     Cargar la muestra *corazon.mp3* y asociarla a un canal *corazon*. 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.
   - 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.
   - Extender la funcionalidad permitiendo subir el pitch del canal con la tecla `'P'` o bajarlo con `'p'`, en saltos de 0.1 unidades.
   - Incluir una *Reberb3D* con el preset `FMOD_PRESET_CONCERTHALL` y situarla en las coordenadas (3,0,7), 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.

FMOD Studio

4. En este ejercicio utilizaremos el editor de FMOD Studio para crear dos eventos sencillos. Comenzaremos creando un nuevo proyecto *ex.fspro* en el que cargaremos los assets *music.ogg* y todas las muestras *remoX.ogg*. Con ellos implementaremos los eventos:

   - **[2 pt]** Evento *remo*: simulará el sonido continuo de remos en el agua, mediante un *scatterer instrument* en el que cargaremos todas las muestras *remoX.ogg*. Añadir a cada muestra una variación aleatoria de pitch de 6 semitonos.

     A continuación implementaremos dos parámetros continuos con valores en [0,1]:

     - *densidad*: controla el número máximo de muestras que pueden sonar simultáneamente, mediante una automatización de *Spawn rate* en el scatterer.
     - *reverb*: añadimos al evento un efecto de reverb y este parámetro controlará la cantidad de reverb (*wet level*) de dicha reverb.
   - **[1 pt]** Evento *música* acuática:
     - Crear un evento *musica* de tipo *2D Timeline*, cargar la la muestra *musica.ogg* en el *timeline* y crear una **región de loop** sobre ella.
     - A continuación crear un parámetro *profundidad* de tipo continuo y con rango [0,1] para simular la inmersión en agua. Para ello añadiremos un (pre)-efecto *MultiBand EQ* y una automatización que decremente la frecuencia de corte al incrementar el valor del parámetro: con *profundidad=0* tendremos *Frec.(A)=22000 Hz* y con *profundidad=1* será *Frec.(A)=20 Hz*.

   Guardar el proyecto con ambos eventos y **asegurarse de que se abre correctamente en el editor antes de entregarlo** (debe contener el archivo de proyecto *ex.fspro* y todas las carpetas necesarias para cargarlo).