# Sonido en videojuegos

Grado en Desarrollo de Videojuegos

**Examen final, enero 2021**

---

**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 **NombreApellido1Apellido2.zip** que 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 como *Archivo* → *Exportar* → *Exportar múltiple*.
- **Comprobar que al cargar las pistas en Audacity, se obtiene el resultado esperado.**

---

1. **[2 pt]** El archivo *agua.ogg* contiene una pista estéreo con sonido subacuático. Se pide:
   - Realizar un loop coherente de 2 segundos con la técnica vista de *fades* en clase: tomar una muestra de 2.5 segundos, hacer *fade-in* de 0.5 segundos al principio de la muestra, *fade-out* de 0.5 segundos al final y hacer la superposición temporal de ambos fragmentos. Para poder ver la construcción del loop exportar el resultado en dos archivos (ambos estéreo):
     - *agua1.wav* con el fragmento de fade-in seguido del resto de la muestra
     - *agua2.wav* únicamente con el fragmento de fade-out.
2. **[2 pt]** El archivo *galope.wav* contiene una muestra de sonido de un caballo al galope. Se pide:
   - Construir un loop coherente (galope continuo) de unos 5 segundos de duración y exportarlo en *galope1.wav*.
   - Duplicar la pista resultante y dividirla en dos pistas mono. Utilizar el balance para enviar la primera al canal izquierdo y la segunda al derecho. A continuación utilizar la herramienta de envolvente en ambas pistas para conseguir un efecto de movimiento desde el canal izquierdo al derecho. Por último obtener una pista estéreo con las dos pistas resultantes y exportar el resultado en *galope2.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.

---

3. **[3 pt]** En este ejercicio implementaremos algunas opciones básicas de posicionamiento 3D con FMOD. En la carpeta *FMOD API User Manual* se proporciona la documentación de la API. 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 la muestra de *musica.ogg* como banda sonora y *latido.wav* como emisor 3D que podremos mover en el plano X-Z. Para facilitar la depuración y corrección del programa implementar un sencillo **renderizado** que muestre la posición del emisor (latido) y listener, así como el pitch.

   Implementaremos de manera incremental las siguientes funcionalidades:

   - Cargar *musica.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 de *latido.wav* en loop 3D y asociarla a un canal emisor *latido*. Situar dicho emisor en la posición $(0, 0, 4)$ 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 (también terminará la reproducción del canal *música*).
   - A continuación, extender la funcionalidad del bucle permitiendo mover el listener en el plano (X-Z) en saltos de 0.5 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 *latido* con la tecla `'P'` o bajarlo con `'p'`, en saltos de 0.1 unidades.
   - Añadir un cono al emisor con ángulos $15^{\text{o}}$ (interior) y $30^{\text{o}}$ (exterior) y cuya dirección será la del eje Z. Al mover el listener en el plano debe percibirse el efecto de atenuación de este cono.

**FMOD Studio**

4. **[3 pt]** En este ejercicio utilizaremos el editor de FMOD Studio para crear dos eventos sencillos. Comenzaremos creando un nuevo proyecto *ex.fspro* en el cargaremos los assets *music.ogg* y todas las muestras *pajaroX.ogg*.

   Evento de pájaros:

   - Crear un evento *pajaros* de tipo *3D Timeline*.
   - Añadir un *scatterer instrument* y añadirle todas las muestras de *pajaroX.ogg*.
   - Añadir a las muestras una modulación aleatoria de pitch de 4 semitonos.

   Evento de música acuática:

   - Crear un evento *musica* de tipo *2D Timeline*, cargar la la muestra *musica.ogg* en el *timeline* de ese evento y crear una *región de loop* sobre la muestra.
   - 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)=22 kHz* y con *profundidad=1* será *Frec.(A)=20 Hz*.

   Guardar el proyecto con ambos eventos en el archivo *ex.fspro* y **asegurarse de que se abre correctamente en el editor antes de entregarlo**.