Manipulación del sonido en videojuegos

PID_00249857

Rafael Pérez Vidal

Tiempo mínimo de dedicación recomendado: 2 horas

Universitat Oberta de Catalunya

Índice

Introducción

Uno de los puntos más importantes a la hora de crear videojuegos es el sonido. El sonido es de aquellas cosas que si no está se echa de menos, que si está mal se nota, y que si está pasa desapercibido. En todo videojuego hay que controlarlo y dominarlo.

Os proponemos trabajar cómo es el sonido y qué se requiere para hacer que esté vivo, que sea orgánico. Por eso, trabajaréis con una de las bibliotecas más utilizadas en el entorno de los videojuegos: SDL. Comprenderemos que el sonido no es una onda, sino un conjunto de datos, y como tales, pueden ser manipulados a nuestro gusto.

Antes que nada, presentaremos una introducción a ciertos conocimientos sonoros, características del sonido y cómo se define este.

Para este reto es necesario que se tengan conocimientos de programación en C++.

Como compilador, trabajaremos sobre una base de proyecto de Visual Studio C++, pero como el código es C++ universal, podéis utilizar cualquier otro compilador de C++.

La biblioteca básica que utilizaremos para realizar gráficos será SDL, Simple DirectMedia Layer (web oficial), y le aplicaremos otra complementaria para tener un control más simple, SDL_Mixer (web oficial).

Enlaces de información y conocimiento:

- SDL Simple Media Layer.
- SDL_Mixer, Sample Multi-Channel audio mixer library.
- FMOD Cross-platform audio engine.
- The Synthesis ToolKit in C++ (STK)

1. El sonido en los videojuegos

El videojuego es un producto multimedia, es decir, está formado por diferentes medios de transmisión de información para producir la obra completa: la imagen y el sonido. El sonido forma parte de la experiencia del juego y muy a menudo la experiencia queda enriquecida por el sonido. Nos hace sentirla más viva, más auténtica, y nos permite recordarla.

El sonido es una onda, una onda analógica que se transmite por un medio, el aire, hasta nuestros oídos. Para tratar el sonido en un videojuego, lo que se hace es digitalizar el sonido para hacer un tratamiento. La digitalización de una onda no tiene por qué hacerse siempre del mismo modo. Hemos de tener en cuenta toda una serie de características.

Una vez digitalizado y guardado dentro del ordenador o en un sistema de almacenamiento digital, al sonido hay que darle forma. Hay diferentes maneras de guardar un sonido o una música en un ordenador.

Una vez que ya lo tenemos guardado, veremos qué necesitamos manipular y cómo para utilizarlo en nuestros videojuegos.

1.1. Características del sonido

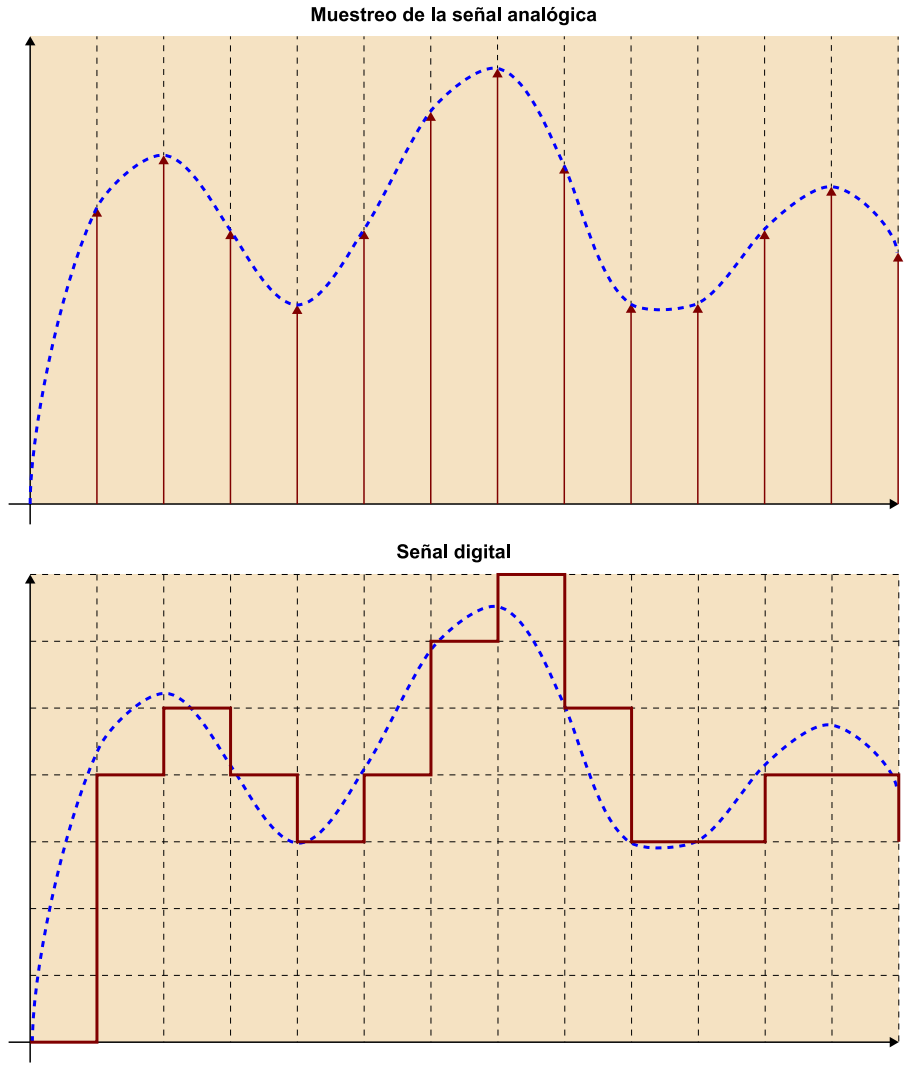
Veremos tres características básicas: la frecuencia de muestreo, los bits por muestra y los canales muestreados.

La frecuencia de muestreo es el número de muestras por unidad de tiempo que se toma de un sonido analógico para producir una grabación, durante el proceso necesario para convertir la señal de analógica a digital. Como todas las frecuencias, generalmente se expresa en hercios (ciclos por segundo; símbolo: Hz) o múltiplos suyos, como el kilohercio (símbolo: kHz), aunque pueden utilizarse otras magnitudes.

Los valores más típicos son 11.025, 22.050, 44.100 o 48.000 Hz por segundo.

Para decidir a qué frecuencia hay que hacer el muestreo, hemos de hacer caso al teorema de Nyquist.

Gráfica 1.



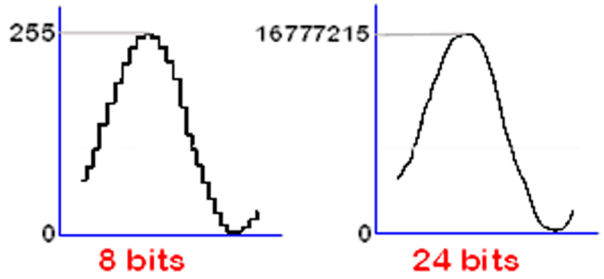
Según el teorema de muestreo de Nyquist-Shannon, para poder replicar con exactitud la forma de una onda, es necesario que la frecuencia de muestreo sea superior al doble de la máxima frecuencia que se ha de muestrear. El oído humano, sano y joven oye desde 20 hasta 22.050 hercios (algunas personas hasta 24.000 hercios). Por lo tanto, para hacer un muestreo que englobe todo lo que oímos, con 44.100 hercios es suficiente.

Contenido complementario

No todos los aparatos utilizan sonido a 44.100 hercios de muestreo. Ved una tabla comparativa.

Los bits por muestra marcan la resolución a la que se digitaliza el sonido, el peso que tiene cada altura del volumen de la digitalización. Cuantos más bits tengamos de resolución, más perfecta quedará la onda que se ha de reproducir. En caso de tener pocos bits, al reproducir el sonido se produce un efecto de susurro, como de viento, una distorsión.

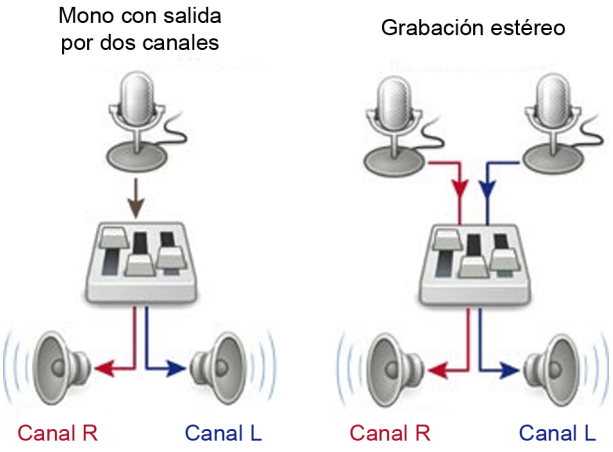
Gráfica 2.



Los valores más habituales son resoluciones de 8 o 16. Hay grabaciones a 32 o 64 bits, pero de cara a videojuegos no las usamos. Son más para estudios profesionales del sonido o para *masters* de grabación.

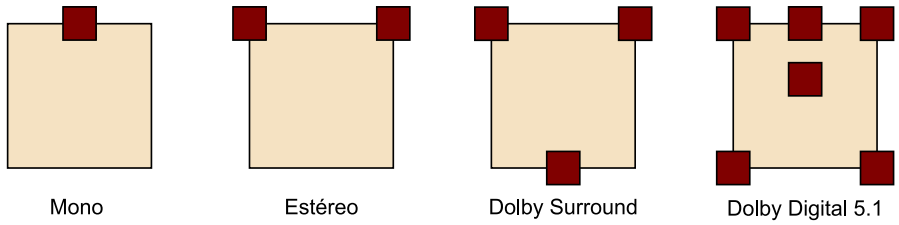
Los humanos tenemos dos oídos, y esto nos permite dar al sonido una situación espacial: derecha, izquierda, delante y detrás. Por nuestra disposición auditiva, nos cuesta más saber si un sonido se da arriba o abajo. Por lo tanto, a la hora de reproducir el sonido, para dar la sensación de que está a nuestro alrededor, para conseguir la inmersión, usamos más de un dispositivo de reproducción del sonido, más de un altavoz.

Imagen 1.



El sonido que tratamos puede estar grabado con un solo micrófono y reproducirse en un solo altavoz. Es lo que se denomina *sonido monoaural* (*mono* = 'uno'). O también se puede grabar con dos micrófonos y reproducir en un sistema estéreo, es decir, de dos altavoces. Si queremos más sensación de inmersión, usaremos otros sistemas extras, como el 5:1 o el 7:1, para posicionar el sonido allí de donde proviene de nuestro alrededor.

Imagen 2.



Los videojuegos han ido evolucionando a lo largo del tiempo junto con las máquinas que los reproducen. Por lo tanto, nos podemos encontrar con juegos monoaurales, estéreos, 5:1, etc.

Hay que decir, sin embargo, que un sonido normalmente parte de un punto en el espacio, es decir, es monoaural. Pensemos, por ejemplo, en el sonido de una bala, de una explosión o de una persona hablando. Este sonido parte de un emisor y llega a nuestros dos oídos, es decir, a un sistema estéreo. Dependiendo del volumen y la dirección hacia donde va, somos capaces de saber dónde está la bala, la explosión o la persona. Por lo tanto, es habitual tener sonidos monoaurales y reproducirlos en el juego posicionándolos en el espacio para que suenen por un sistema estéreo.

1.2. Tipos de ficheros de sonido

Cuando tenemos un sonido digitalizado, hay que darle un formato a la información, es decir, el fichero donde se guarda. Al igual que con la imagen, tenemos diferentes maneras de guardar el sonido y de trabajar.

Por un lado, están los ficheros **WAV** o **VOC**, que guardan sonido puramente. Es audio digital, normalmente sin compresión de datos. El formato WAV fue desarrollado y es propiedad de Microsoft y de IBM, se utiliza para almacenar audio en el PC y admite archivos mono y estéreo a varias resoluciones y velocidades de muestreo.

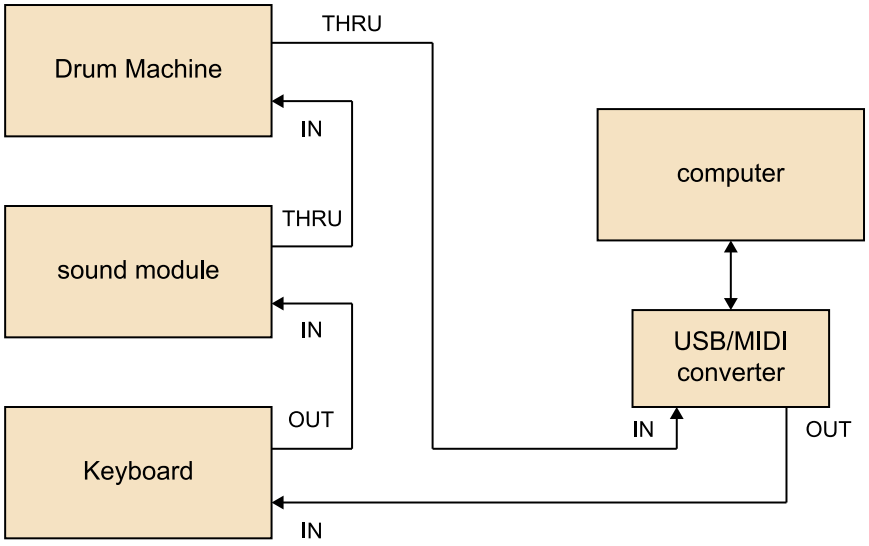
Se utiliza fundamentalmente con el formato PCM (no comprimido) y, al no tener pérdida de calidad, puede ser usado por profesionales.

Para tener calidad CD de audio, se necesita que el sonido se grabe a 44.100 Hz y 16 bits. Por cada minuto de grabación de sonido, se consumen unos 10 *megabytes* de espacio en disco.

Solo se pueden grabar archivos de 4 *gigabytes* como máximo, lo que equivale aproximadamente a 6,6 horas en calidad de CD de audio.

Otro formato de fichero de sonido es **MIDI** o **MID**. Hay que decir, no obstante, que MIDI no guarda puramente el sonido, sino la partitura de una canción.

Gráfica 3.

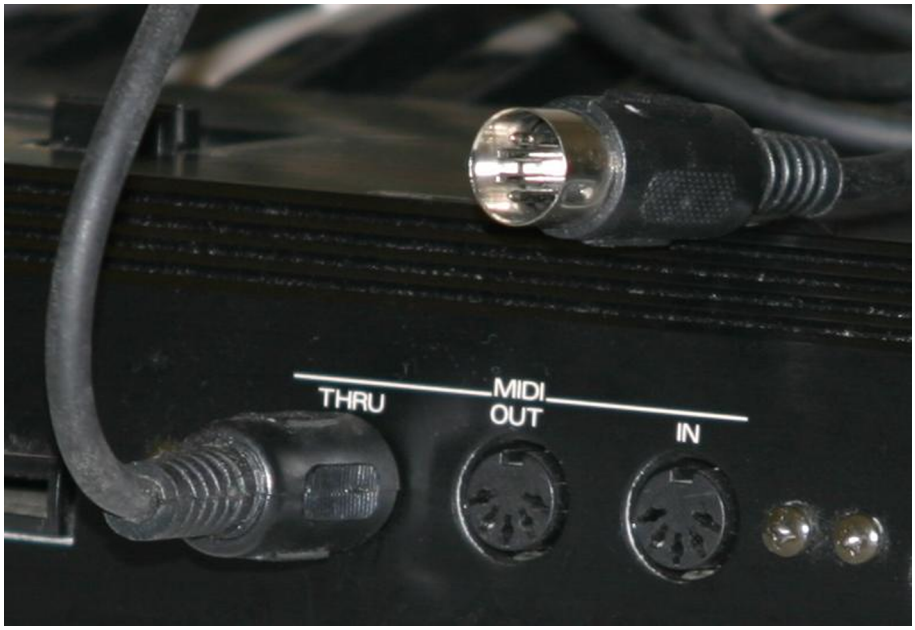


La palabra MIDI es el acrónimo de **musical instrument digital interface**, que quiere decir, más o menos, 'interfaz digital para instrumentos musicales'. MIDI describe una norma de comunicación física entre sistemas (hay conectores MIDI, cableados MIDI, protocolos de comunicación MIDI, instrumentos MIDI...) y las características del lenguaje que hacen posible el intercambio de información.

Contenido complementario

Cada aparato musical lleva su propia implementación MIDI. Es una hoja que describe cómo el instrumento entiende la partitura que le llega. Estas hojas de referencia se denominan *MIDI implementation charts*.

Imagen 3. Conectores MIDI de instrumentos musicales.



MIDI no transmite sonidos; transmite información digital. Así, los ficheros MIDI no son ficheros de sonido, al estilo de los WAV.

Las instrucciones de un fichero MIDI se componen de datos (por ejemplo, cuándo se pulsa una nota, cuándo se suelta, con qué fuerza, etc.). Cuando estamos oyendo un MIDI (*.mid) en nuestro ordenador, es como si alguien estuviera interpretando una canción en aquel mismo instante.

Hay formatos de fichero híbridos entre los dos mundos del WAV y el MIDI; son los formatos **MOD**, **XM**, **S3M** o **IT**.

MOD es un formato de archivo para música que surge a mediados de la década de 1980 y diseñado por secuenciadores de tipo *tracker*, como SoundTracker, ProTracker, ModPlugTracker, etc. Estos fueron ampliamente difundidos, especialmente por los computadores Amiga y Atari ST, pero en 2008 seguía estando muy extendido en sistemas operativos como GNU/Linux, especialmente por su facilidad para permitir crear música para videojuegos con bajo coste de espacio.

Imagen 4. Ejemplo de tracker: FastTracker II

Lo característico del formato MOD es la inclusión en su cabecera de *samples* o muestreos digitales. Estos samples son el sonido de los diferentes instrumentos que componen la música. Mediante una variación en la velocidad de reproducción de un *sample*, se consigue un tono más alto o más bajo, y se pueden reproducir así todas las notas musicales.

Hoy en día los *trackers*, los programas para hacer este tipo de ficheros, todavía se usan. Actualmente, los más populares y que funcionan en Windows son MilkyTracker y Schism Tracker.

Aquí podéis ver dos ejemplos de música con *tracker* grabados en vídeo:

BeaT / Osmosys - Celestial Fantasia

Depeche Mode - Enjoy the Silence

Si os queréis descargar la canción de Depeche Mode en formato MOD, la podéis encontrar en el ModArchive, que es un gran centro de música en este popular formato. Fijaos en lo poco que ocupa (168 Ks).

Estos *samples* están ordenados en patrones (*patterns*) y se hace una lista con ellos, cosa que hace que se puedan repetir o se pueda saber en todo momento qué patrón y qué nota se está reproduciendo. El formato MOD también contiene la información de cuándo y de qué instrumentos hacer sonar para la reproducción de la obra, así como un número limitado de efectos sonoros, como el vibrato, el deslizamiento de volumen, etc.

Tal como comentamos en el PLAN 1, se usaron muchísimo en los inicios de la demoscene.

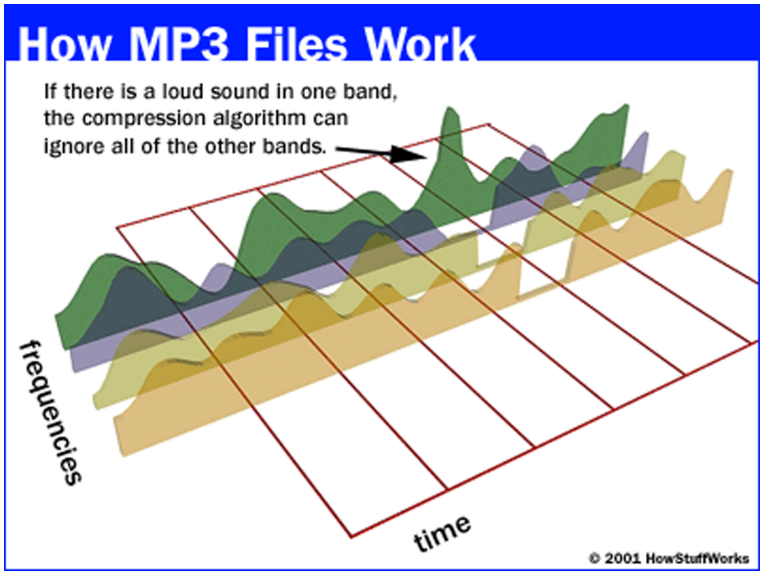
La gran ventaja del formato MOD es su reducido tamaño, por lo que era extensamente utilizado para música de videojuegos. En sus tiempos se distribuían en disquetes de 700 kilobytes, de manera que un solo disquete contenía una obra completa o varias canciones.

Parecidas limitaciones de espacio fueron relajándose con la introducción del CD en el mundo del videojuego, por lo que fue perdiendo terreno ante otros formatos.

Y para acabar, comentaremos los formatos de codificación de música de calidad más utilizados en Internet: **OGG** y **MP3**.

MP3 trabaja en formato como, por ejemplo, WAV, pero comprime aquellas partes que pueden resultar inaudibles. Rebaja mucho los tamaños de los archivos, pero en contra de la calidad de reproducción.

Imagen 5.



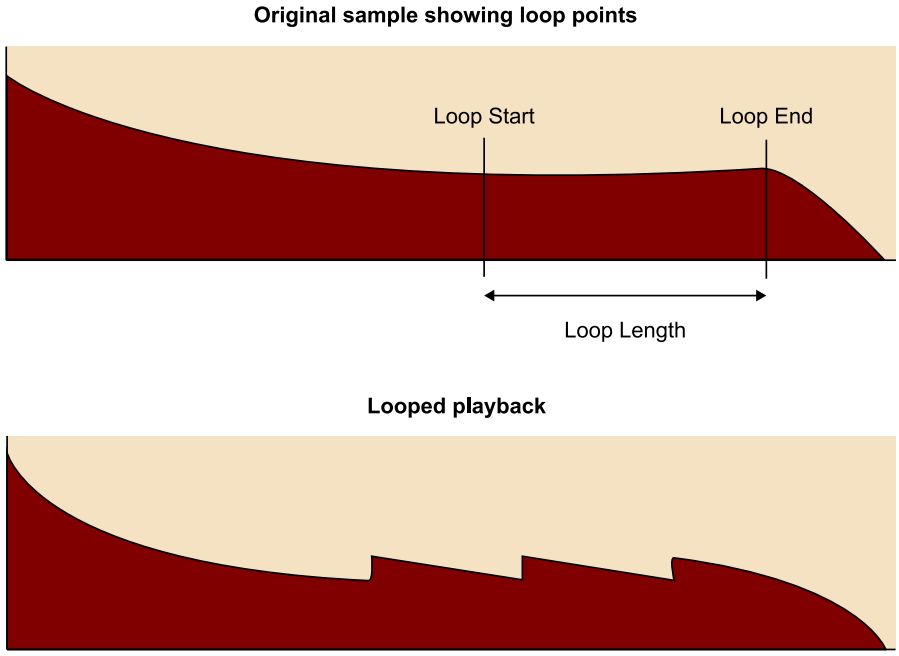
OGG es similar, pero libre y multiplataforma, y consigue calidades muy altas con pesos muy pequeños.

1.3. Características del sonido en videojuegos

Cuando hagamos la reproducción del sonido en nuestros videojuegos, hay unas funciones básicas que necesitamos que cumpla la biblioteca que utilizaremos. Estas funciones nos tienen que permitir posicionar correctamente nuestro sonido y poderlo poner en marcha cuando lo requiramos. Las funciones son las siguientes:

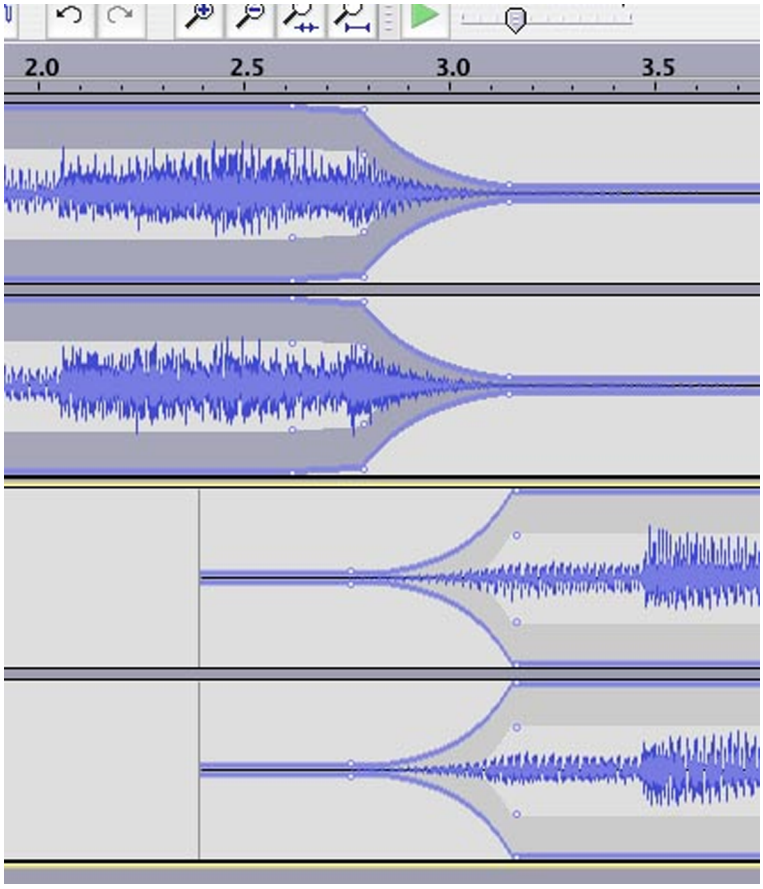
- Reproducir y parar un sonido.
- Reproducir un sonido en repetición (*loop*).

Gráfica 4.



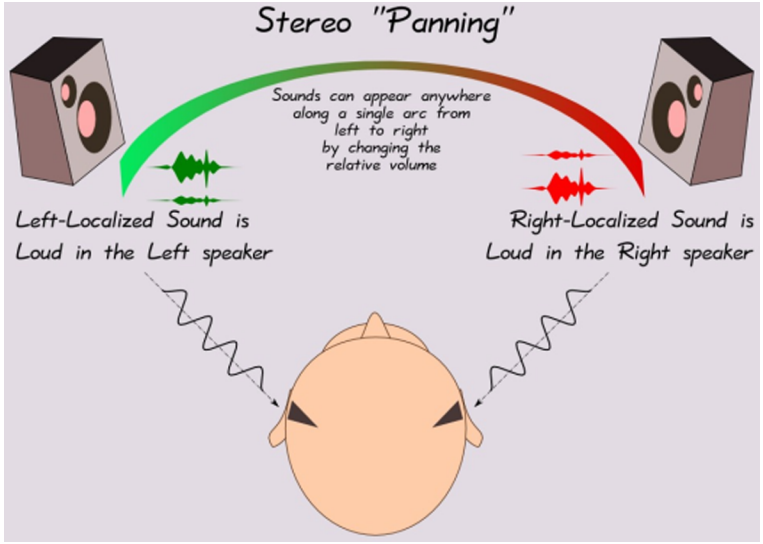
- Hacer aparecer o desaparecer progresivamente el sonido (*fade in* / *fade out*).

Imagen 6.



- Mover un sonido progresivamente de un altavoz a otro o posicionarlo en un punto del espacio (*panning*).

Imagen 7.



- Controlar el volumen de reproducción del sonido para dar la sensación de si está lejos, si está cerca o si se va aproximando (*distance*).

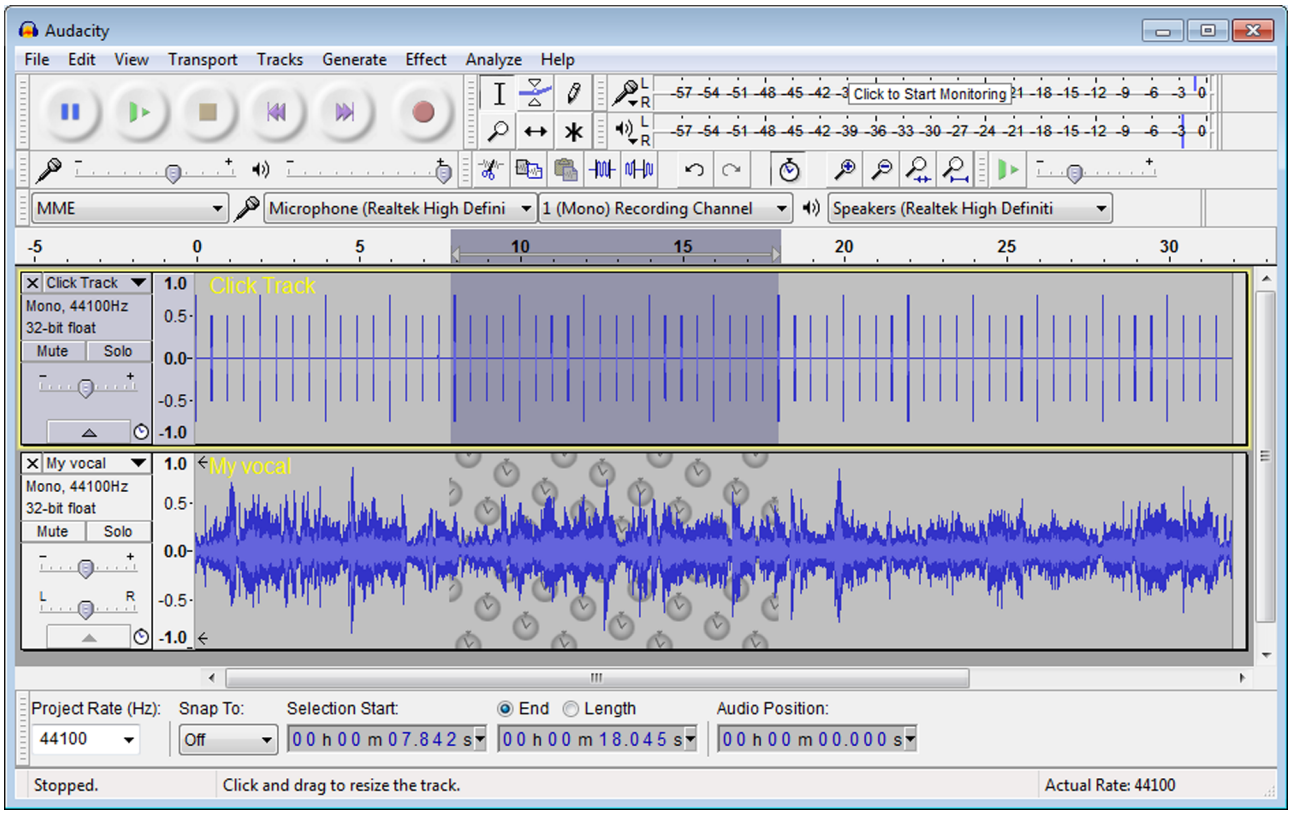
2. Programas para editar sonidos

Hay muchos programas que sirven para trabajar ficheros de audio. Nos centraremos en programas que modifican sonidos, Dejaremos de lado los programas compositores musicales, secuenciadores o *trackers*.

Los más populares para la edición de sonido y que son gratuitos son Audacity y Wavosaour .

Audacity nos permite la edición del sonido en formato multipista. Es decir, podemos cargar diferentes sonidos y juntarlos para hacer uno solo. También tiene muchos efectos de sonido que podemos aplicar.

Imagen 8.

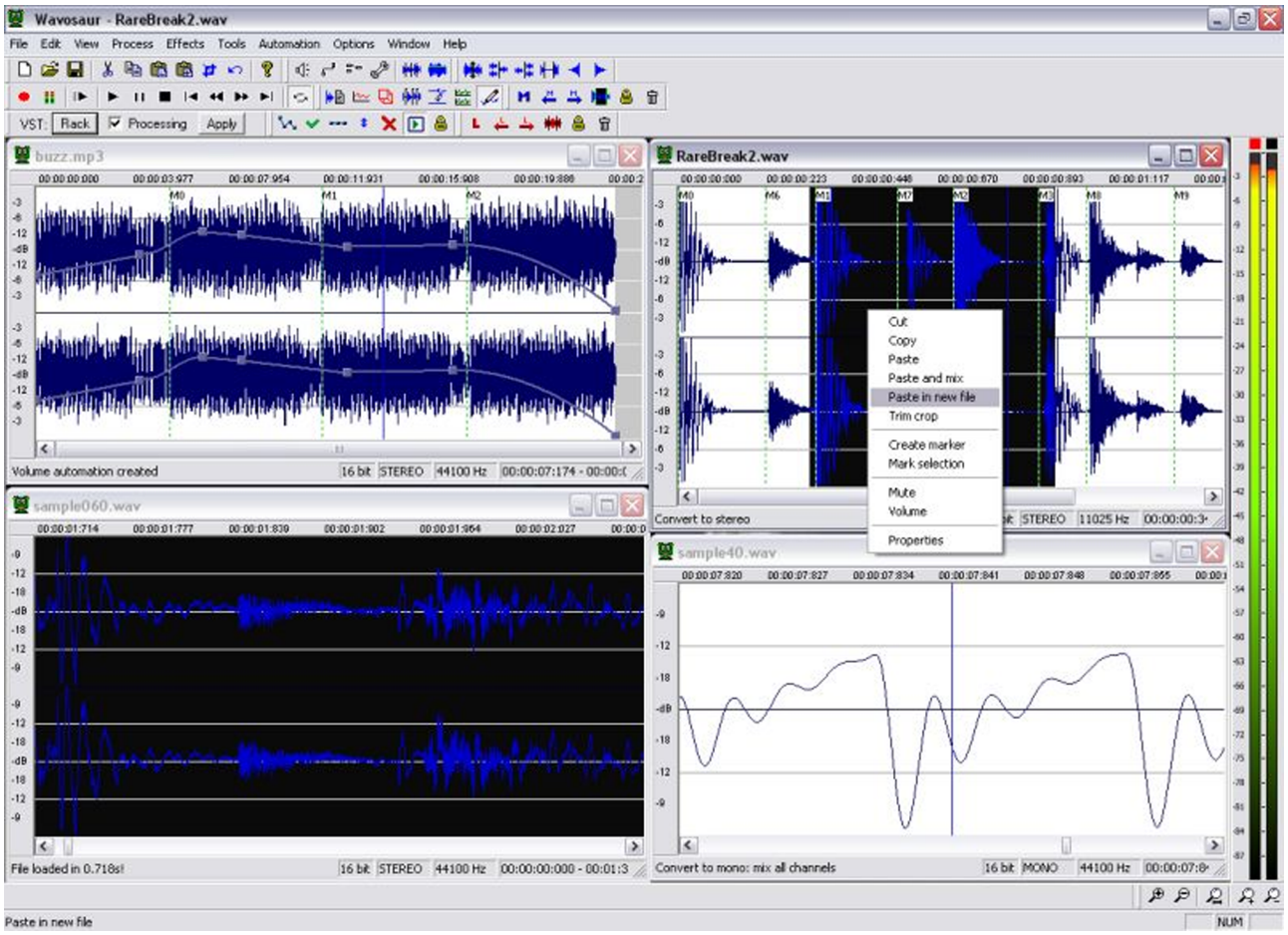


En cambio, Wavosaour es un editor de sonido de una única pista. Podríamos hacer el mismo trabajo con Audacity, pero este es más pequeño de peso, no requiere instalación y tiene una semejanza de control con el popular SoundForge, que es muy común en el entorno profesional del sonido. Permite también la posibilidad de trabajar con *plug-ins* VST.

Los VST son una tecnología desarrollada por la empresa Steinberg para conectar sintetizadores de audio y *plug-ins* a editores de sonido. Permiten usar instrumentos virtuales en vez de equipamiento físico.

Hay muchos gratuitos, y se recogen en webs como esta. Se pueden hacer nuevos o podéis crear vuestro propio instrumento.

Imagen 9.



3. Uso de una biblioteca para el sonido: SDL_Mixer

Para poner en práctica el uso del sonido, trabajaremos con la biblioteca SDL. Usaremos su biblioteca extensión SDL_Mixer.

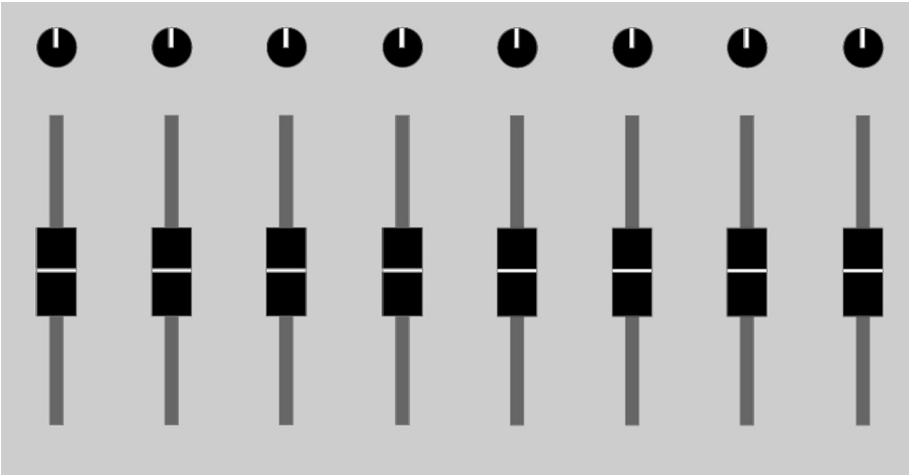
SDL_Mixer trabaja el sonido por *chunks*. Un *chunk* es una *struct*:

```
typedef struct Mix_Chunk {
    int allocated;
    Uint8 *abuf;
    Uint32 alen;
    Uint8 volume;     /* Per-sample volume, 0-128 */
} Mix_Chunk;
```

En nuestro juego cargaremos en memoria los sonidos (*chunks*) que necesitemos, y a medida que los vayamos reproduciendo, serán asignados a canales de sonido (*channels*).

SDL_Mixer tiene un mezclador de audio de 128 canales.

Imagen 10.



Channel 0	Channel 1	...	Channel 127
Play / Stop	Play / Stop	...	Play / Stop
Volume	Volume	...	Volume
Pause	Pause	...	Pause
Fade status	Fade status	...	Fade status
Sonido (*chunk*)	Sonido (*chunk*)	...	Sonido (*chunk*)

Si no decimos lo contrario, cuando hagamos la reproducción del sonido, se irá asignando ordenadamente cada sonido (*chunk*) que reproduzcamos al siguiente canal libre.

3.1. Inicialización del sistema y gestión. Apertura y cierre

Inicialización

Hay que tener inicializado el sistema de audio de SDL. Estas son las funciones de la biblioteca que nos hacen falta.
SDL_InitSubSystem(SDL_INIT_AUDIO);

o

SDL_Init(SDL_INIT_EVERYTHING);

Una vez hecho esto, iniciamos el SDL_Mixer con
Mix_OpenAudio(44100,MIX_DEFAULT_FORMAT,2,1024) ;
y asignamos cuántos canales usaremos del Mixer.
Mix_AllocateChannels(128);

Cierre del sistema

Cuando acabemos de utilizar el SDL_Mixer, hay que cerrar la biblioteca con
Mix_CloseAudio();

3.2. Carga de sonidos

La carga la realizamos desde una misma función para todos los formatos.
Mix_Chunk* sound = Mix_LoadWAV(file);

Tendremos que guardar estos punteros de *Mix_Chunk* en un *array*, vector, lista, mapa..., algún contenedor para poder hacer referencia después al reproducirlo en los canales.

Si tenemos que descargar un sonido de memoria, utilizaremos
Mix_FreeChunk

Podemos dar un volumen concreto al *chunk*, pero no es lo habitual, puesto que esperamos que el sonido ya venga dado con el volumen adecuado. Si acaso, ya modificaríamos el volumen del sonido al reproducirlo en el canal correspondiente.

3.3. Reproducción de sonidos

Haremos *play* del sonido (*chunk*) que queramos y le asignaremos un canal, o podemos hacer que el sistema lo asigne solo. Al hacer el *play*, se nos devolverá el canal (*channel*) por donde se reproduce. Es importante guardar este valor para controlar el sonido que se está reproduciendo. Mirad la documentación de la biblioteca SDL_Mixer para más información.

- Mix_PlayChannel(–1, sonido, 0);
  - channel (–1, ...), chunk, loops (–1, 0, 1, ...)

- Funciones útiles:
  - Mix_Volume
  - Mix_PlayChannel
  - Mix_PlayChannelTimed
  - Mix_FadeInChannel
  - Mix_FadeInChannelTimed
  - Mix_Pause
  - Mix_Resume
  - Mix_HaltChannel (stop)
  - Mix_ExpireChannel
  - Mix_FadeOutChannel
  - Mix_Playing (gets)
  - Mix_Paused (gets)
  - Mix_FadingChannel (gets)
  - Mix_GetChunk (gets)
  - Mix_ChannelFinished ***

Reto 1

Practicaremos la reproducción de sonidos. Este reto trata de hacer un reproductor de sonido con los botones necesarios de reproducción (PLAY), parada (STOP), pausa (PAUSE) y canción siguiente o anterior (NEXT, BACK).
Tenéis las tres canciones que se reproducirán y los gráficos que hay que usar.
El reproductor funciona con las siguientes teclas:
(Z) BACK (X) PLAY (C) PAUSE (V) STOP (B) NEXT

3.4. Efectos en tiempo real

SDL_Mixer nos permite hacer ciertos efectos especiales en el sonido una vez que está reproduciéndose en un canal. Básicamente, nos da dos prehechos y nos ofrece la posibilidad de ser capaces de programar nosotros mismos nuestros propios efectos.

- Mix_SetPanning
- Mix_SetDistance
- Mix_SetPosition (ángulo, distancia); es la suma de los dos anteriores.
- Mix_SetReverseStereo

Reto 2

Reproducid sonidos de una batalla espacial. Id lanzando diferentes sonidos de disparos, de naves y de explosiones en diferentes puntos del espacio del sonido. Jugad con la distancia y el posicionamiento a derecha e izquierda. Haced que los sonidos dados de las naves pasen de derecha a izquierda y de izquierda a derecha.

SDL nos permite realizar nuestras propias funciones para hacer efectos de sonido o tratar el sonido como queramos. Podemos interceptar el sonido de un canal o el sonido completo de la mezcla final, y con el *buffer* de sonido que nos da, trabajarlo para hacer un efecto. Estas funciones se ejecutarán en cada ciclo de ejecución de la biblioteca de sonido; permiten ver el *buffer* de sonido que se enviará a continuación a reproducir. Nos da el trozo de audio que se lanzará y con él podemos hacer lo que queramos.

Lo único que nos pide es que la función ha de tener un formato concreto. Debe tener unos parámetros definidos.

En el caso de querer interceptar el audio que sale de un canal, ha de tener el siguiente formato:

```
void Effect(int chan, void *stream, int len, void *udata)
```

La función tiene que ser de retorno *void*. El primer parámetro es el canal que ha ejecutado la función, El segundo es un puntero directo al *buffer* de sonido (aquí encontraremos los datos, en el formato establecido en Mix_OpenAudio). El tercero es la longitud de este *buffer*. El cuarto es un puntero a los parámetros que nosotros hemos definido que queremos tener cuando se llama a la función. Este último parámetro se define al hacer Mix_RegisterEffect.

En el caso de quererlo hacer al *buffer* final, a la mezcla de canales, los parámetros de la función del efecto responden a este otro patrón. Básicamente, no hay el canal, que ya es obvio que no esté, si estamos tratando la mezcla final.

```
void Effect(void *udata, Uint8 *stream, int len)
```

- Mix_RegisterEffect
- Mix_UnregisterEffect
- Mix_UnregisterAllEffects
- Mix_SetPostMix

Reto 3

Añadid una función en el punto final de la mezcla de audio de forma que podamos mostrar por pantalla una forma de onda, como si fuera un osciloscopio del audio real que se está reproduciendo.

3.5. Otras peculiaridades de SDL_Mixer

SDL_Mixer tiene otras funciones que nos pueden ser útiles también. Tiene la capacidad de hacer grupos con los canales de sonido para hacer cosas como bajar de volumen todos los canales de un grupo de golpe. Esta función no tiene demasiado uso.

La otra cosa que hace es distinguir un canal especial para hacer la reproducción de música de fondo. De este modo, podemos separar la manera de lanzar o hacer efectos de sonido de una canción de fondo. Este canal tiene algunas particularidades especiales de control. Y también es la única manera de utilizar con SDL_Mixer ficheros MIDI y MOD.

Hay que apuntar también que, dependiendo del formato que uséis, os requerirá algunos DLL específicos:

- FLAC -> flac 1.2.1
- MOD, XM, S3M, IT... (module formats -> libmikmod 3.1.12)
- OGG -> libogg 1.3.0 y libvorbis 1.3.2
- MP3 -> smpeg 0.4.5

Para más información sobre SDL_Mixer, podéis consultar la documentación de las bibliotecas en línea.

4. Otras bibliotecas de sonido

Para acabar el PLAN, querríamos comentar que hay otras bibliotecas referentes al sonido muy buenas. Si queréis entrar más en control del sonido, os parecerá interesante tenerlas en cuenta.

4.1. OpenAL

OpenAL quiere decir Open Audio Library. Es una API de audio multiplataforma gratuita desarrollada por Creative Labs para el renderizado eficiente de audio posicional y multicanal en tres dimensiones. Está ideada para su uso en videojuegos, mediante formato PCM, bien en 8 o 16 bits, en formato mono o estéreo. El motor de renderizado se encarga de todos los cálculos necesarios, como la atenuación, el Doppler, etc.

Es una API antigua, pero muy vigente. Hay muchos juegos conocidos que han hecho uso de esta biblioteca, como Doom 3, Quake 4, Unreal Tournament, GRID, Battlefield...

En su web oficial encontraréis todo lo que hace falta para hacer uso de esta biblioteca.

4.2. FMOD

FMOD es un motor de efectos de sonido para videojuegos y aplicaciones desarrollado por Firelight Technologies, que reproduce y mezcla archivos de sonido de varios formatos en muchos sistemas operativos. Tiene un potente editor de sonidos muy integrado con el punto de la programación en C++, lo que hace que el trabajo del diseñador de sonidos y el de programador queden muy separados y sean complementarios.

Es una biblioteca multiplataforma en C++ y actualmente está muy extendida. Los más importantes motores de juego la usan de manera integrada.

Aunque FMOD es gratuita si la aplicación o juego lo es también, si hay un beneficio económico, Firelight quiere su parte para una licencia.

En la web oficial de FMOD, encontraréis toda la información sobre la API y FMOD Studio.

4.3. Synthesis ToolKit

Synthesis ToolKit o STK es una biblioteca en C++. Es un conjunto de clases escritas en el lenguaje de programación C++ para el procesamiento de señales de audio. Es de código abierto. STK fue diseñada para facilitar el rápido desarrollo de la síntesis de música y software de procesamiento de audio, con énfasis en la funcionalidad multiplataforma, el control en tiempo real, la facilidad de uso y el código de ejemplo educativo.

SDK es muy portable. Actualmente se ejecuta con el soporte en tiempo real (audio y MIDI) en Linux, Macintosh OS X y plataformas de ordenador Windows.

Es una biblioteca utilizada para la generación de sonido y la manipulación de efectos de sonido muy potente.

Encontraréis toda la información en su web.

5. Reto final: la Cueva de los Condenados

Esta será la actividad final evaluativa del PLAN: la realización de un juego basado completamente en el sonido.

La Cueva de los Condenados

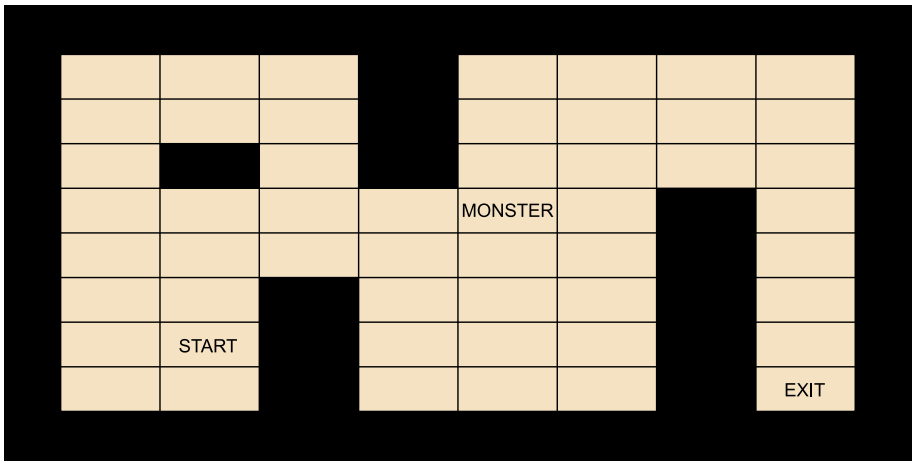
Detenido por tus robos, has sido echado al fondo de un agujero oscuro. Has caído en una cueva. No se ve nada de nada; solo puedes oír ruidos. En este agujero hay un monstruo; un monstruo que duerme y ronca. Puedes oír cómo duerme. Ten mucho cuidado de no despertarlo. Si se despierta, se te zampará rápidamente.
Lejos, muy lejos, puedes oír un salto de agua. Allí está la salida.

¿Crees que solo usando el oído podrías llegar a la salida de esta cueva?

La cueva la simularemos en una tabla de 10 × 10 casillas. Hay paredes que no puedes cruzar, una casilla de salida y otra de llegada, donde está el salto de agua.

Este sería el plano de la cueva:

Imagen 11.



El juego no tiene que mostrar nada en pantalla; solo una pantalla en negro. Podéis utilizar la pantalla de consola (de texto) para sacar valores de las variables y verificar que se está jugando bien. El jugador se tiene que guiar por el ruido.

Diseñad vosotros mismos los sonidos y realizad efectos en tiempo real sobre ellos, como eco o reverberación.

Condiciones de juego

- Nuestro personaje simula que tiene que poder caminar hacia delante y hacia atrás, y poder girar a 90 grados. Por lo tanto, hay que guardar del personaje la casilla de la tabla en la que está y la dirección a la que está mirando (cuatro direcciones).
- Cada vez que cambie de casilla, reproduciremos un sonido de un paso.
- El juego se controla con tres teclas: avanzar, girar a la derecha y girar a la izquierda.
- El monstruo roncará de manera constante. Y el salto de agua también sonará constantemente.
- Tenéis que controlar la dirección y la intensidad del sonido del monstruo y la cascada en función de dónde está el personaje. Supondremos que las paredes no paran el sonido del monstruo ni de la cascada.
- Si el personaje intenta ir hacia una casilla donde hay una pared, se debe oír el sonido de ese golpe contra la pared.
- Cada vez que el personaje choque contra una pared, el monstruo cambiará a una casilla adyacente y que no tenga pared, aleatoriamente.
- Si el monstruo o el jugador están en la misma casilla, el primero se zampará al segundo y el juego se acabará. Entonces pararéis todos los sonidos y pondréis una musiquita de derrota.
- Si el jugador llega a la casilla de salida, gana y el juego se acaba. Pararéis todos los sonidos y pondréis una música de victoria.

Sonidos que hacen falta:

- Monstruo que ronca.
- Choque con la pared.
- Salto de agua.
- Monstruo que se come al jugador.
- Paso del jugador.
- Paso del monstruo.
- Música de victoria.
- Música de derrota.