API de sonido - SDL

Alfonso Pérez González (alpegon3.at.fiv.upv.es)

Introducción

Para la realización de esta práctica se pedía la implementación de una API de sonido que funcionase utilizando la librería Simple Directmedia Layer (SDL).

SDL es una librería multiplataforma diseñada para proporcionar acceso a bajo nivel al sonido, teclado, joystick y hardware 2D y 3D. SDL se puede ejecutar en Linux, Windows, Windows CE, BeOS, MacOS, Mac OS X, FreeBSD, NetBSD, OpenBSD, BSD/OS, Solaris, IRIX, y QNX.

En SDL la gestión del sonido se realiza a muy bajo nivel y hay que conocer bien el hardware que se está utilizando y los formatos de audio que se quieren reproducir, por lo que para esta práctica usaremos la librería SDL_mixer[1]. Dicha librería simplifica enormemente el manejo del sonido, tanto en la carga como en la reproducción de sonidos y música desde diferentes fuentes y formatos utiliza internamente la librería SDL.

Dentro del juego existe una clase que gestiona el sonido, la clase SoundsManager y será dentro de ésta donde definiremos los nuevos métodos a utilizar por la librería. Para definir el API que se quiere usar se tiene que cambiar la función *#define* de la clase *Sound.h*.

Cambios realizados

A continuación se pasan a describir los cambios realizados en las clases del juego. Todavía existen bastantes clases con llamadas directas a la librería FMOD, por lo que se ha decidido mantener estas llamadas para cuando se quiera utilizar la librería FMOD, definiendo éstas entre los *#ifdef CSND_FMOD375 #endif* correspondientes. SDL gestiona todo a través de la interfaz de *SoundsManager*, por lo que todas las llamadas que se hagan relacionadas con el sonido serán a través de dicha interfaz.

Se indicará el fichero modificado y los cambios que se han realizado en dicho fichero.

SoundsManager.h

- Se ha añadido una nueva definición con el número máximo de rutas de canciones que se pueden tener cargadas en memoria (`#define CSM_MAX_MUSIC_TRACKS 4`).

[1] http://www.libsdl.org/projects/SDL_mixer/

- Se han comentado las variables y las funciones que utiliza FMOD dentro del `#ifdef CSND_FMOD375` correspondiente y se ha definido una nueva variable (`Mix_Music *music`) para gestionar la música con SDL.
- Se ha añadido una nueva estructura para gestionar las rutas de las músicas del juego `char MusicPath[CSM_MAX_MUSIC_TRACKS][CSM_LONG_PATH_NAME]`.
- Se han añadido una nueva serie de funciones relativas a SDL para gestionar la música, a continuación se numeran las funciones y se muestra una breve definición:
    - `inline void Play (const unsigned char s, int vol)`: reproducirá un sonido s, con un volumen vol a través de la clase Sound.
    - `void LoadMusic(const char* file)`: carga el archivo de música que se le pasa como parámetro, pero no lo reproduce.
    - `void MusicFadingIn(void)`: reproduce el archivo de música previamente cargada en memoria. Utiliza un timer de 2s para ir aumentando progresivamente el volumen del sonido.
    - `void MusicFadingOut(void)`: para el archivo de música que actualmente se está reproduciendo. Utiliza un timer de 2s para ir disminuyendo el volumen progresivamente.
    - `void SetMusicPath(char ** musicPath)`: establece las rutas de los ficheros de música que se repoducirán durante el juego.
    - `void PauseMusic(void)`: pausa la música que se está reproduciendo actualmente.
    - `void ResumeMusic(void)`: continúa la repoducción de la música previamente pausada.
    - `void StopMusic(void)`: para la música y libera los recursos reservados.
    - `void NextMusicTrack(void)`: carga en memoria el siguiente archivo de música de la lista de archivos cargada anterioremente. Reproduci la lista en forma de bucle, es decir, cuando termine con el último archivo vuelve al primero.
    - `static char *ErrorMsg()`: devuelve el último error ocurrido dentro del Mixer de SDL en forma de una cadena de texto comprensible.
- Se han creado nuevos callbacks para que se puedan llamar a las funciones de repoducción y parada de música
    - `void MusicFadingIn(void *dummy);`
    - `void MusicFadingOut(void *dummy);`

SoundsManager.cpp

- Dentro de la función `void CSoundsManager::Init(void){}` se han definido las nuevas variables de inicialización que utilizará SDL, y se inicializa el Mixer de sonido.
- Se ha creado una nueva llamada para cerrar el sonido a través de la API de SDL en la función `CSoundsManager::~CSoundsManager()`. Se cierran los sonidos llamando a `Sound.at().close()` y posteriormente se llama a `Mix_CloseAudio()`.
- Se han introducido entre los correspondientes `#ifdef CSND_FMOD375` y `#elif CSND_SDL` todas las funciones que se definieron anteriormente en el fichero SoundsManager.h. Llamando así a la función adecuada según se seleccione una API u otra.

- Todas las nuevas funciones definidas están comentadas para que puedan ser parseadas por Doxygen.

Sound.h

- Se ha creado la definición CSND_SDL y se les ha dado un valor a ésta y las demás para que el compilador no genere problemas.
- Se ha incluído la librería *SDL_mixer.h*, necesaria para gestionar el sonido en SDL, en principio la librería principal de SDL no es necesaria puesto que no se usará ninguna de sus propiedades, solo las que se encuentran dentro del mixer.
- Se han definido nuevos valores para el volumen, ya que el volumen máximo en SDL es 128.
- Se ha creado el nuevo tipo de sonido que reproducirá la librería de SDL (`Mix_Chunk *Sound`). En principio SDL reproduce música y efectos por separado, pero sólo los efectos se pueden escuchar por varios canales a la vez, por lo que se ha definido el sonido como un tipo Mix_Chunk (ver más adelante la sección de problemas).
- Se ha definido una nueva función Play que cargará los sonidos en la clase CSound y se ha redefinido la llamada a la función getStream ya que SDL devuelve un tipo distinto de dato.

Sound.cpp

- En la función `Init(char *name, int vol, char *path)` se ha añadido el método (`Mix_LoadWAV(Path)`) para cargar los sonidos en SDl.
- En la función `close()` se ha añadido la función que cerrará y liberará los recursos del sonido actualmente cargado (`Mix_FreeChunk(Sound)`).

SIGame.cpp

- En la función `bool CSIGame::Initialize (void)` se inicializa el *SoundsManager* se cargan las rutas de música y se lanza el hilo que gestionará el inicio de la música:

```
// Load the sound manager
SoundsManager.Init();
// Start the game music
char **AuxM=new char*[CSG_MAX_MUSIC_TRACKS];
for(int i=0;i<CSG_MAX_MUSIC_TRACKS;i++)
      AuxM[i]=CGS_MusicFiles[i];
// Loads the music path
SoundsManager.SetMusicPath(AuxM);
// Loads the first music song
SoundsManager.NextMusicTrack();
// Starts the music
_beginthread( MusicFadingIn, 0, NULL );
```

Si_main.cpp

- Ya que SDL gestiona de una manera distinta el sonido que la librería FMOD y lo hace todo a través del *SoundsManager* en la función `void ManageKeyboardInput()` se han añadido nuevas llamadas a métodos de la función *SoundsManager*.
- Con la tecla M se enciende y apaga la música:

```
	if (Window.Keys->keyDown [0x4d] == TRUE
|| Window.Keys->wivik[0x4d] == TRUE)
	{
		static float key_timer_music_on_off;
		if (!Keyboard.KeyWait(CSIG_music_on_OFF)) return;
	// 200 ms between change

		if (SoundsManager.music_on){
			SoundsManager.PauseMusic();
		} else {
			SoundsManager.ResumeMusic();
		}
	}
```

- Con la tecla espacio se cambia de canción:

```
if (Window.Keys->keyDown [VK_SPACE] == TRUE
|| Window.Keys->wivik[VK_SPACE] == TRUE)
	{
		Window.Keys->keyDown[VK_SPACE] = FALSE;
		if (SoundsManager.music_on){
			SoundsManager.StopMusic();
			SoundsManager.NextMusicTrack();
		}else{
			SoundsManager.NextMusicTrack();
		}
	}
```

GameSounds.h

- Se ha definido una nueva estructura para almacenar las rutas de los archivos de música.

GameSounds.cpp

- En esta clase se especifican los nombres de los ficheros de sonido que se cargarán para el juego de manera estática. Se han cambiado los nombres porque SDL no gestiona mp3. Además se han añadido los nombres de los ficheros de sonido que cargará el juego.

si_Basecode.cpp

- En la callback `DialogAboutProc` se han cambiado los nombres de las funciones a los que llaman los hilos para apagar y encender la música y también se ha cambiado la variable que se consulta para saber si la música está en funcionamiento o no. Ahora

para saber si la música se está reproduciendo se llama a `SoundsManager.music_on`.

- Dentro de la callback `DialogStartupProc` en el caso en el que se cierra la ventana `case WM_CLOSE`, se han añadido las llamadas a las funciones de parar el sonido y detener la música.

Problemas surgidos

Los formatos que SDL puede reproducir dependen del tipo de variable que se gestiona:

- Cuando es del tipo Mix_Chunk, se pueden cargar ficheros del tipo ficheros WAVE, AIFF, RIFF, OGG, y VOC.
- Cuando se trata con un tipo Mix_Music, se pueden cargar ficheros del tipo WAVE, MOD, MIDI, OGG, MP3, FLAC

Como se puede observar, sólo los tipos Mix_Music pueden gestionar ficheros mp3. El problema es que la música en SDL se gestiona a través de un canal único y no todos los eventos de sonido pueden ser del tipo Mix_Music ya que sólo se puede reproducir uno a la vez. El Mix_Music está pensado para reproducir la música de fondo del videojuego, no sus efectos sonoros.

Por otro lado están los Mix_Chunk, este tipo de fichero están pensados para ser utilizados por los efectos de sonido del juego, pueden agruparse en canales, se puede elegir el canal de reproducción pueden reproducirse varios a la vez, etc. El problema está en que no pueden cargar los ficheros del tipo .mp3, por lo que para que la librería de sonido de SDL funcione correctamente, los sonidos que se incluyan en el juego no podrán tener este formato.

Teniendo en cuenta esto, se convirtieron los archivos de sonido del juego de mp3 a ogg, y se cambió la definición de los nombres de sonido en el fichero GameSounds.cpp.