Programación de audio
Práctica 1 : Reproducción de buffers de audio

En esta práctica, vamos a aprender a cargar buffers de audio desde un fichero WAV, a crear fuentes de sonido desde las que reproducir el buffer, y a configurar el oyente. Trabajaremos sobre el motor que se desarrolló en la asignatura **Programación 2D**.

En primer lugar, debemos crear una clase que nos permita inicializar el sistema de audio. Llamaremos a esta clase `AudioEngine`, y tendrá la siguiente interfaz:

```
class AudioEngine {
public:
	static AudioEngine& Instance();

	void Init();
	void Finish();
protected:
	AudioEngine(void) : device(NULL), context(NULL) {}
	~AudioEngine(void);
private:
	static AudioEngine* engine;
	void* device;
	void* context;
};
```

Esta clase sigue el patrón singleton, y permite, mediante los métodos `Init` y `Finish`, inicializar y finalizar el sistema de audio. Antes de utilizar audio en nuestro juego, el método `Init` debe ser invocado, y antes de salir de la aplicación, se debe llamar a `Finish`.

En `Init`, se debe crear un dispositivo y un contexto de OpenAL, utilizando las funciones `alcOpenDevice` (con parámetro `NULL`) y `alcCreateContext` (con el dispositivo creado en la

llamada anterior y `NULL` como parámetros, respectivamente). A continuación estableceremos el contexto creado como activo con `alcMakeContextCurrent`.
En el método `Finish`, se deben eliminar el contexto y el dispositivo (en ese mismo orden) con las funciones `alcDestroyContext` y `alcCloseDevice`.

Una vez creada esta clase, implementaremos una clase para el manejo de buffers de sonido, con la siguiente interfaz:

```
class AudioBuffer {
public:
	AudioBuffer(const String& filename);
	~AudioBuffer(void);

	bool IsValid() const { return alBuffer != 0; }
	unsigned int GetBuffer() const { return alBuffer; }
private:
	unsigned int alBuffer;
};
```

En el constructor, fijaremos el valor de la variable miembro `alBuffer` a 0, y a continuación obtendremos del fichero WAV los datos necesarios para cargar la fuente de sonido. Para la lectura de datos de un fichero, podemos utilizar la clase `File` que se proporciona con el enunciado.

Para poder hacer esto, es necesario conocer la estructura de un fichero WAV. En primer lugar, encontramos una cabecera con los siguientes datos:

Dato	**Bytes**
ChunkID (“RIFF”)	4
ChunkSize	4
Format (“WAVE”)	4
SubChunkId (“fmt “)	4
SubChunkSize	4
AudioFormat	2
Channels	2
SampleRate	4
ByteRate	4
BlockAlign	2

Dato	Bytes
BitsPerSample	2
ExtraParamsSize	2
ExtraParams	X

Los dos últimos elementos sólo aparecen si el valor de AudioFormat es distinto de 16. En ese caso, debemos leer el valor de ExtraParamsSize y saltarnos ese número de bytes.

A continuación, debemos leer la cabecera de los siguientes bloques de datos (4 bytes), buscando uno que coincida con el string “data”. Mientras el bloque no tenga ese identificador, nos lo saltaremos leyendo su tamaño (4 bytes) y saltándonos el número indicado de bytes.

Una vez encontrado el bloque “data”, leeremos su tamaño (4 bytes), reservaremos memoria para ese tamaño, y leeremos ese número de bytes dentro de la memoria reservada.

Una vez hecho esto, generaremos el buffer de OpenAL y, con la función `alBufferData`, rellenaremos la información de dicho buffer, con los siguientes parámetros:

- `buffer`: El identificador del buffer generado en el paso anterior.
- `format`: Si el valor de Channels leído del fichero es 1, el formato será AL_FORMAT_MONO16. En caso contrario, será AL_FORMAT_STEREO16.
- `data`: El puntero a la memoria reservada donde hemos cargado el bloque “data”.
- `size`: El tamaño del bloque “data”.
- `freq`: El valor SampleRate leído del fichero.

También necesitaremos una clase AudioSource que represente una fuente de sonido. Tendrá la siguiente interfaz:

```
class AudioSource {
public:
	AudioSource(AudioBuffer* buffer);
	~AudioSource();

	void SetPitch(float pitch);
	void SetGain(float gain);
	void SetLooping(bool loop);
	void SetPosition(float x, float y, float z);
	void SetVelocity(float x, float y, float z);

	void Play();
	void Stop();
	void Pause();
```

```
	bool IsPlaying() const;
private:
	unsigned int source;
	AudioBuffer* buffer;
};
```

En el constructor, crearemos la fuente de OpenAL (que será eliminada en el destructor), y llamaremos a los métodos que establecen los valores de la fuente pasándole valores por defecto (sin reproducción en bucle, pitch y gain valdrán 1, y la posición y velocidad serán 0 en todas sus coordenadas). Por último, indicaremos el buffer a utilizar con `alSourcei`.

El resto de métodos llamarán a las funciones apropiadas de OpenAL según aparece en las diapositivas del tema 1.

Por último, tenemos que crear una clase Listener que represente a nuestro oyente. Su interfaz es la siguiente:

```
class Listener {
public:
	static Listener& Instance();

	void SetPosition(float x, float y, float z);
	void SetVelocity(float x, float y, float z);
	void SetOrientation(float x, float y, float z);
protected:
	Listener();
	~Listener() {}
private:
	static Listener* listener;
};
```

Esta clase sigue el patrón singleton. En el constructor, llamaremos a los métodos que establecen las propiedades del oyente, pasándole 0 en todas ellas. Los otros métodos se implementarán llamando a las funciones de OpenAL de acuerdo con la información de las diapositivas del tema 1.

Una vez hecho esto, probaremos que la reproducción de audio funciona cargando un buffer con el fichero **“data/music.wav”**. Generaremos una fuente de sonido, y reproduciremos el buffer generado.

Debemos abrir una ventana con la clase `Screen` del motor 2D, para poder recibir entrada del teclado. Cuando se pulsen las teclas de cursor, se debe hacer lo siguiente:

- Arriba: Aumentamos el pitch.
- Abajo: Disminuimos el pitch.

- Izquierda: La fuente se desplaza a la izquierda.
- Derecha: La fuente se desplaza a la derecha.

La calificación máxima de esta práctica es de **3 puntos**.