Audio

1. Introduction

This tutorial teaches the basics to use Audio effects in your game.

2. Setting up the Windows project

Download the **`600098-Initial_ECS_Framework_withAudio.zip`** file and extract it to a suitable location on your file store. Open the Visual Studio project. Run the code and you should hear a sound file.

Use the cursor keys to move the camera and the sound should appear to move in relation to the camera’s position and orientation.

3. Implementing Audio

Look at the following code in **`SceneManager`**:

```
var device = ALC.OpenDevice(null);
ALContextAttributes att = new ALContextAttributes();
var context = ALC.CreateContext(device, att);
ALC.MakeContextCurrent(context);
```

The audio context needs to be created in an appropriate location in the code. In the example code, it is created in the constructor of **`SceneManager`**. So that the audio context is available in any Scene.

Look at the following code in **`GameScene`**:

```
using OpenTK.Audio.OpenAL;
using System.IO;
```

These allows us to access the audio functionality of OpenAL and load sounds.

Look at the data member declarations in **`GameScene`**.

- The **`sourcePosition`** is a ***`Vector3`*** and is used to store the position of the audio source that is sounding out an audio effect.

- The **`audioBuffer`** is an **`int`** and is used as a handle to the stored audio effect.

- The **`audioSource`** is an **`int`** and is used as a handle to a source sound. Any number of sound sources can be created from a single sound buffer.

Look at the audio code in the constructor in **`GameScene`**.

- We create a buffer to store the audio clip (**`audioBuffer`**). This buffer can be used many times to create individual instances of the same audio clip without having to load in the audio file again or to create more buffers for this particular audio clip.

- We then create the audio source **`audioSource`** based on the audio clip in the buffer **`audioBuffer`**. We set the audio source **`audioSource`** so that it loops, set its position in the scene, and then play the audio source.

Look at the **`Update()`** method in **`GameScene`**.

- We update the listener’s Position and Orientation based on the camera.
- We find the Moon Entity and update its position (move right).
- We update the Audio Source’s position based on the Moon’s position.

Look at the **`Close()`** method in **`GameScene`**.

- We stop the audio source and then dispose of the audio source and audio buffer.

4. Exercises

Please attempt these exercises, but if you get stuck or you are confused then ask for help during the scheduled lab times.

1. Move the code for loading of the audio into the **ResourceManager** so that we only load any audio file once even if we request it to be loaded more than once.

   The **ResourceManager** needs to load the sound file into an Audio Buffer. It also needs to create a Dictionary to hold the handle to the Audio Buffers (int) to make sure that only one Audio Buffer is created, so that the same sound cannot be loaded twice. The Audio Buffer can then be requested by another class to create an Audio Source.

2. Create a **ComponentAudio** that can be used to manage the Audio Source etc. Use this **ComponentAudio** to add a sound to an **Entity**.

   You can do this by moving the remaining audio code for the Audio Source to the constructor of your **ComponentAudio**.

   You will also need a method to set the position of the Audio Source (and perhaps one for the velocity), e.g.

   ```
   public void SetPosition(Vector3 emitterPosition) {}
   ```

   You will also need methods to allow the Audio Source to be played and stopped, and a method to clean up on close.

   You also need to remember to add an entry into the **ComponentTypes** enum in the IComponent.cs file.

3. Create a **SystemAudio** that will handle the audio updating to any **ComponentAudio**.

   You have a **ComponentAudio** that needs to be told where its **Entity** is by getting the **ComponentPosition** of its Entity so that it can update the position of the Audio Source.

This **SystemAudio** needs to set a MASK so that it can handle any **Entity** that have both **ComponentPosition** and **ComponentAudio**.

It then needs to retrieve the position from the **ComponentPosition** and pass this to the **SetPosition()** method of the **ComponentAudio**.

4. When the **GameScene** ends, the Close() method is called. This method should be used to cleanup any resource. Any audio that is currently playing will continue to play even if the **GameScene** ends because the Audio Context will continue. Therefore the Close() method should notify all Managers to cleanup also. The **EntityManager** should notify all components in each **Entity** that they should close (put a Close() or similar method in all components). In the **ComponentAudio**, you can stop the sound.

5. Add a new audio file to your project (place it into a folder called Audio). Make sure that your sound file in **MONO**. 16bit works well too.

6. Change the code so that you use your new sound instead of the provided buzz sound.