**Spatial Audio in Game Design worksheet**

**Software Versions:**

Unity – 2022.3.4f1 (student licence can be acquired [here](https://unity.com/products/unity-student).)

Wwise – 2022.1.6.8263 (Audiokinetic launcher can be downloaded from [here](https://www.audiokinetic.com/en/download/).)

Visual Studio – 2022 (Community version can be downloaded [here](https://visualstudio.microsoft.com/downloads/).)

**Github Setup:**

Github allows you to make an online repository to store your gaming projects, store multiple versions and collaborate with other developers.

1. Please access the following link to the Github website or alternatively, you can search for Github in the browser: [GitHub: Let’s build from here · GitHub](https://github.com/)
2. Click on “Sign up” in the top right of the screen
3. Follow the steps to create an account
4. Once your account has been made and you have logged in, click on the Create Repository button on the left hand side

To make the repository:

* Make sure your account is the “Owner” and create a repository name. (this can be anything you want it to be, but please note that only the hyphen special character can be used and will also replace spaces)
* Add a description of the project (this is optional, but always good as a brief description to let other people who are viewing your repo know what your project is about)
* Select whether you want it to be public or private
* There is the option to add a README file, which allows for a longer description of the project
* Add a .gitignore from the dropdown menu called Unity (we will do something with this later)
* You can choose a licence but for the purposes of the workshop, this isn’t required
* The repo can then be created

We will now download a copy of the repo to the PC:

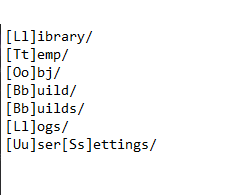
* Load up Github Desktop and login to the account you just created
* On the left hand side is an “Add” button with the option to clone the repo (you can also make repos from the Desktop app, but I wanted to show both the web browser and desktop versions of Github)
* Select a path to clone the repo too (I would recommend the H Drive)

We will come back to Github later!

**Unity Project Setup:**

I have taken the opportunity to make a Unity project which is set up with a small pre-made scene, which we will use to demonstrate different types of spatial audio techniques in conjunction with the audio middleware Wwise:

* Use the following link to access the repository for the project:
* Click on the green “Code” button and download a zip file of the project
* Once downloaded, copy the project folder into your newly created Github repo on the PC and extract it using 7Zip (or another type of decompressor)
* Open the .gitignore file and make sure the file types at the top look like the screenshot below. This means that when the project is pushed to Github, it won’t load unnecessary files which are always made when the project is loaded



Let’s do our first push of the project to Github, so you have it saved somewhere online before we start editing anything!

* Go back to Github Desktop; you should now see that the left hand side has been populated with a number of files
* Towards the bottom of the left hand side is a Summary section, which requires a brief summary of what changes were made to the project. As this is the initial push, I would recommend something like “First push”
* Below is a section where you can add further comments
* Once populated, click “Commit to main”
* Finally at the top is a section that will say “Push to origin”. When clicked, this will push the project (and any changes made to the project, such as the amended .gitignore file) to the repo
* Go back to the desktop browser, refresh the page and you should see that it has been updated with a timestamp on when changes occurred

**Wwise Setup:**

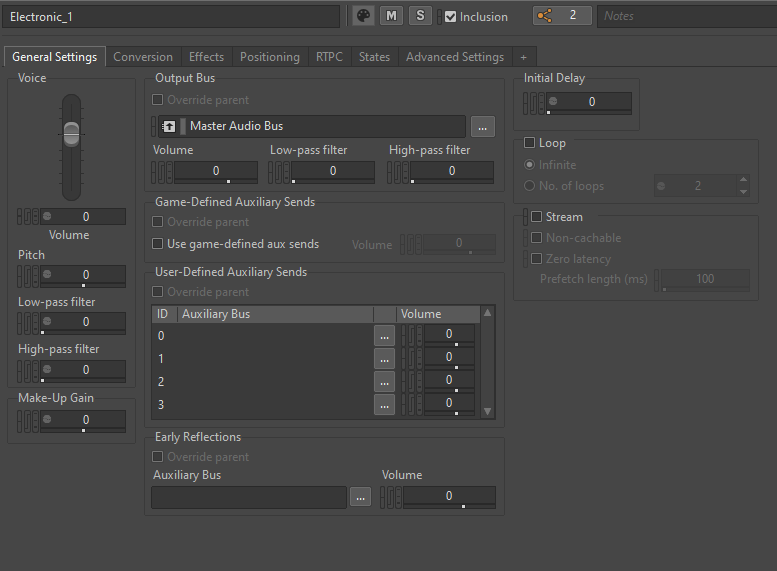
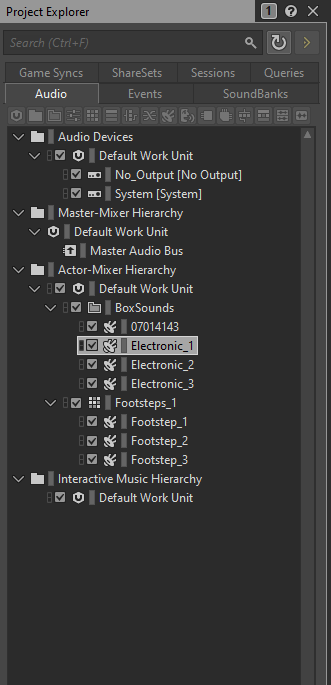
Wwise is an audio middleware with some similarities to DAWs like Pro Tools (although layout is different). It can assist with adding spatialization, effects and more to your games. When combined with Unity, it brings its own scripts for use in the game engine which we will get into later.

To setup Wwise:

* Open the AudioKinetic Launcher and click on the Unity tab
* The launcher should be able to locate your Unity projects on the PC automatically
* Click on “Integrate Wwise in Project…” button
* From the dropdown, locate the version of Wwise installed (2022.1.6.8263). This is needed for the SDK
* For deployment platforms, you will want to select which platform you intend to deploy too. In this instance, Windows will probably be the most appropriate (if it is not already selected)
* I would recommend leaving the Wwise project path as the default and it will create a new Wwise project
* Click on integrate at the bottom and it will create a new Wwise project linked to your Unity project. This can take some time

**Let’s take a look inside the Wwise project and add our first audio sample:**

* After Wwise has been integrated into the project, click on the sideways diamond shape next to the project name on the right. It will ask you which version of Wwise you want to launch (this should be 2022.1.6 or 2022.1.6.8263, both are the same)
* On the left hand side of the project window is the Project Explorer. In this, there are a variety to tabs associated with different things that we can do with Wwise. To begin with, we will look at the Audio and Events tabs
* Click on the Audio tab and locate “Actor-Mixer Hierarchy”. To import your audio into Wwise, you can either right click on the Default Work Unit and select import audio (Add Files or Add Folders then click import) or you can drag and drop it from the file explorer onto the Default Work Unit
* This will populate a number of the windows on the right hand side, such as the Transport Control, Source and General Settings, etc.
* Click on the play button in the transport control and make sure your audio plays. You can adjust the volume using the “Voice” slider in the general settings if required
* Make sure that the “Loop” box is ticked, so that we can hear always hear the audio in Unity (we can turn this off later)
* When you’re happy with the volume, right click on the audio clip in the Project Explorer and go to New Event -> Play
* Click on the Events tab and you will see your new play event. Again, click on the play button in the Transport Control and we’ll make sure it’s working
* At the top of the screen is a dropdown called “Layouts”. Click on this and change it to the SoundBank layout
* In the project explorer or the SoundBank Manager, right click on the Default Work Unit and create a new sound bank. I would personally call this “Master”, but you can name it anything you want to
* Switch to the Events tab in the Project Explorer and drag your new play event onto the sound bank you’ve just created
* In the Soundbank Manager (main screen not project explorer), make sure that the Default Work Unit and any sound banks have been ticked, as well as the Windows platform and languages (if you’re working on a Mac, this can be selected too)
* Important! Make sure that you save your project and click “Generate Checked” whenever you want to upload something from Wwise to your Unity project, as it may not transfer over if this step is not followed

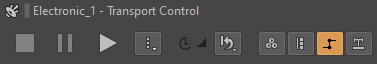
****

Project Explorer

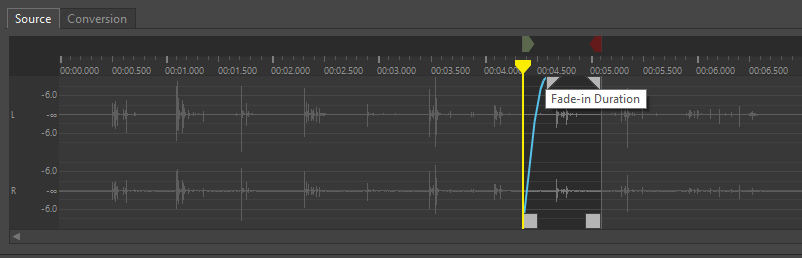
General settings tab

Source and Transport Control

**A screenshot of a video editing

Description automatically generated**

* It is also possible to fade in/out your audio with the top sliders in the source window (screenshot below) as well as trim the start/end time of your audio clip



**Unity Project:**

Let’s take a look at Unity and get a brief introduction to the layout and have a play with the scene!

* To open the project, click on the Unity symbol next to your project in the AudioKinetic launcher (you may have to browse the file explorer through the launcher to locate the project if it does not automatically detect it) or go into the file directory where the project was saved, locate the scene file named “Playground” and double click it (if the scene has not already been loaded on starting the project)
* When the project has been loaded, take a look around the view. You’ll see a scene window which contains a premade scene with a variety of different structures and cubes
* On the right side is the inspector, which contains the parameters for the game objects, scripts, etc., when they are selected. You can change parameters in this view which are public
* On the left side is the Hierarchy, which lists the objects currently applied to the scene
* At the bottom are usually two windows, Project and Console, but there will be a third called Wwise Picker
* The Project window is where you will find a variety of folders from things imported into or created in Unity. This can be objects, materials, scripts, etc. It is important to keep your project well organized and this can help greatly
* The console window will mainly advise you of any warnings or errors which occur with your project. These can be scripting issues, dependency warnings and more. Additionally, you can add your own testing messages within scripts which can be located in the console
* At the top of the scene window are play and pause buttons. Press the play button and have a run around the environment (W for forward, A for left, S for backward, D for right, Spacebar is jump and the mouse will move your view around)

A screenshot of a computer

Description automatically generated

This is a screenshot of a Unity project with a few labels to describe where sections are

Scene

Inspector

Hierarchy

Project window (contains file structure and you can switch to the Console window and Wwise picker)

Let’s add some audio to the scene in Unity:

* Make sure that both your Unity and Wwise projects are open at the same time. You will notice that there is a new tab at the bottom which we mentioned before, Wwise Picker
* Click on this and you should see a number of folders. At the moment, we are particularly interested in the “Events” folder. Make sure there is an event called “Play\_(whatever you called the audio in Wwise)” contained within
* Select one of the cubes around the play area. In the cube’s inspector, click on “Add Component” and in the search bar, look for and add the AKAmbient script
* Two scripts will be added to the object; AKAmbient and AKGameObj. Both of these scripts are required for audio to play with Wwise
* Click on “Add Rigidbody” and make sure that the “Trigger On” section is set to Start only
* Add the audio in the name section by selecting it from the dropdown menu
* Finally, click on the WwiseGlobal object in the hierarchy and make sure that the AkInitializer and AkBank scripts are attached. In the AkBank script, make sure that the “Load On” dropdown is set to awake and the sound bank you made in Wwise is selected next to name
* Press play and you’ll be able to hear the audio in the game scene
* There isn’t currently any spatialization to the audio, so let’s add that in!

**Spatialization:**

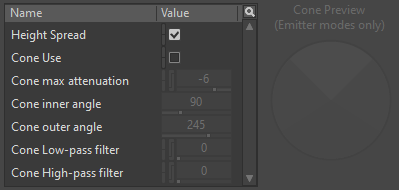
* Go back into Wwise and make sure you’re on the Designer layout
* Go back to the audio clip you imported and make sure it’s selected
* Click on the Positioning tab and from the dropdown next to 3D Spatialization, change this to “Position”
* Make sure attenuation is checked and click the symbol which looks like
* Create a new attenuation and call it whatever you like (I usually name this to something relevant to the object I am using it on / for)
* Click on the button which looks like the following
* There are a number of different parameters which can be changed here, but we will stick with the top one, Distance. Change the Max distance value to 25
* It is also possible to modify the attenuation curves themselves. If you right click on the curve within the graph, there are a number of preset curve types you can choose from. Alternatively, if you double click on the curve, it is possible to add your own points and adjust them manually. There is also the option to combine both of these
* Switch back to SoundBank layout, save your project and generate the selected files again
* Go back into Unity and play the scene; approach your cube and notice that the sound coming from the cube depends on the position and distance between the object and the player. (A small note; the distance value in order to hear the audio for Unreal Engine, is much greater than Unity)

We can also add some further directionality by changing the 3D Spatialization to “Position + Orientation”. When we play audio with the 3D spatialization set to “Position” only, this is actually playing in mono. By adding orientation, the audio is played in stereo instead. It can sometimes be difficult to determine if this is occurring and will normally only be able to heard from the front of the object. In Unity, the front of the object is considered wherever the blue arrow is pointing when the object is highlighted (Z-axis). It is possible to amend the front of the object through code, should you wish to do so. (We will not look at this today, but please feel free to experiment with this yourselves.)

**Cone Attenuation:**

By adding attenuation, we can get an omnidirectional output of the audio, which will change volume based on the distance from the listener to the emitter. We can further effect this by altering the angles at which the audio plays.

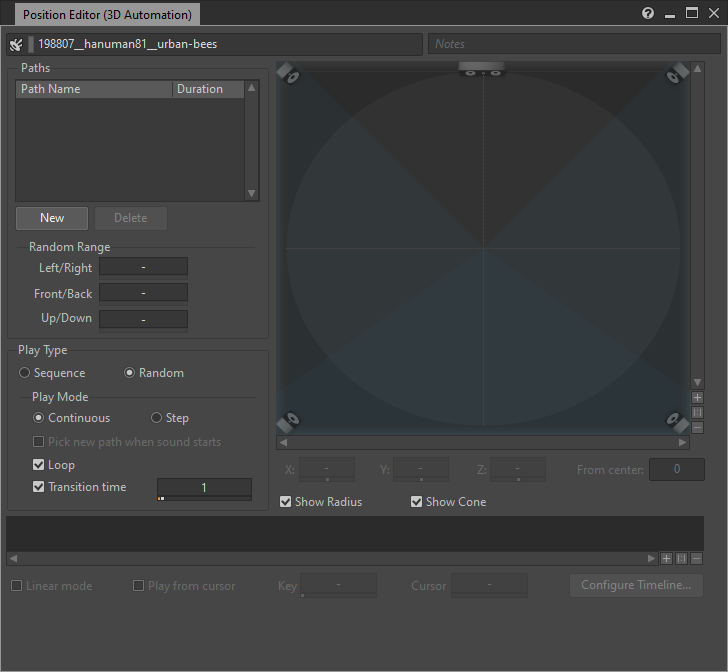
* If we go back to Wwise and open up the attenuation window of the audio we just added spatialization too, on the right hand side of the window is the option for “Cone Use” and a cone preview (circular shape cut into different angles)



* If you click the tick box next to “Cone Use”, it will illuminate the parameters below it, as well as the cone preview
* Firstly, we’ll use the cone preview to see what happens when the cone is used without any of the settings being changed. Locate the transport control (usually at the bottom of the main window) and play your audio. Click on the small circle within the cone preview (highlighted orange) and move it around the circle. The lightest shade of grey is the front of the object and the darkest shade of grey is behind
* Notice how when you move the circle around and the line into the different shades/angles, the volume changes by both the distance and position in the preview. This is a great way to test out your audio before implementing it into the actual game engine
* Stop the audio and let’s make some changes to the values. Try changing the Cone max attenuation to a different value (by increasing the value from the default -6, it will make the sides and rear louder; with 0 having no effect. Alternatively, decreasing it further will have the opposite effect and reduce or remove the audio entirely)
* By changing the cone’s inner angle value, this will increase or decrease the angle at which the loudest volume is heard. It is possible to have it so that the inner angle is 360 degrees, but this provides the same effect as though the cone was not used
* By changing the outer angle, this effects the rear angle of the emitter. Note that when you change these angles, the middle angles are also effected which can increase or decrease their value as well, although there is no direct way to control this
* Finally, you are able to use low and high-pass filters which effect the sides and rear of the cone
* Have a play around with these settings and we will see them in action in Unity. (Remember to save your project and generate the files before switching over!)
* As we have used the same audio from the previous example, no further setup is required and we can simply play the scene and listen to how the audio is effected based on the player/emitter positioning

**3D Automation:**

So far, we’ve had quite static spatialization using just emitters. But what if we wanted to add a sense of movement? We’ll take a look at adding some automation to the 3D position.

* First, we need to add 3D spatialization and some attenuation to our audio. You’ll notice in the project explorer that there is a folder called “Bee\_Automation” which contains a sound file called “Bee\_1”. (If you need a reminder on how to set up 3D spatialization or attenuation to audio, you can click [here](#Spatialization))
* At the bottom of the positioning window, you’ll see a section called 3D Position which is currently set to “Emitter”. For the purposes of this demonstration, we’ll want to set it to “Listener with Automation”
* When this is selected, the Automation button will highlight to the right hand side. Click on it and will produce a window like that on the right
* Here we can add Paths, which are a series of nodes which the audio will follow based on the duration set for the path and how they have been spaced along the timeline at the bottom
* The initial node is set in front of the character and can be move around by left clicking and dragging it around the space with your mouse
* To add additional nodes, double click in the speaker setup. Add a few nodes and position them around the space to your liking and then play the audio using the transport control. Whilst the audio file itself is quite large, the movement will only take 5 seconds to full complete and then remain at the position of the final node

We have the capability to use parameters from Unity to change the audio or effect it in different ways. In the next section, we will look at accessing game parameters to change the audio to a different one and then back again.

**Changing States:**

* First, let’s go into Wwise and select two pieces of audio and place them both in their own folder for clarity purposes. Similar to what we initially did, we’ll add some 3D spatialization with “Position” and add the attenuation we made to both pieces of audio (for a reminder on how to set up the audio, click [here](#AudioWwise) and for spatialization click [here](#Spatialization))
* A screenshot of a computer

  Description automatically generatedCreate a play event for just one of the pieces of audio and then find and click on the play event
* You’ll notice on the right hand side that there is a window which allows you to Add, Remove and Browse. Click on the Audio tab again and drag and drop the other audio file into the window
* Switch to the “Game Syncs” section in the Project Explorer and locate the States folder
* Within the Default Work Unit, right click and create a new state group and give it a name (perhaps something relevant to the task such as SoundSwitcher but it can be anything)
* Then right click on the new state group and add two states, On and Off
* We will also change the transition time to 3, so that there is a gradual change in the audio playing
* Go back to the Audio tab and pick one of the audio files. Switch to the “States” tab and add the new state group you’ve just made. Reduce the “Voice Volume” to -108 for the On switch. Do the same thing with the other audio file but instead reduce the Off voice volume to -108
* Switch to the SoundBank layout, add the play event to the Soundbank, save the project and generate the files again

We’ll now switch back to Unity:

* You’ll notice there is a flattened cube (State Cube) near one side of the scene. We will use this to change our audio states
* In the hierarchy, click on the “StateCube”. You’ll notice that a number of scripts have already been attached to this object. We have the usual AkGameObj and AkAmbient scripts, but some additional ones; Ak Trigger Enter, Ak Trigger Exit and two AkState scripts
* The Ak Trigger Enter script is used when a certain object enters the bounds of a collider, which is attached to that object. The Ak Trigger Exit script is used when a particular object leaves the bounds of a collider. The Trigger Object has been populated as the Capsule. We utilize these scripts in conjunction with the AkState scripts
* In the AkState scripts, you’ll notice that there is a “Name” section. We can populate this with the switch group we made earlier and select the “On” option for one and the “Off” option for the other
* Next we need to select a trigger for this. As mentioned previously, we will use the Enter and Exit scripts to do so. From the “Trigger On” section of the script where On was selected for the name, select “None” from the dropdown menu and then go in again to select AkTriggerEnter. We first select None because if we simply select AkTriggerEnter, it will include any prepopulated choices, which we do not want
* Repeat the same as above with the Off script, but select AkTriggerExit instead
* We need to set up one final AkState script. This is located on the WwiseGlobal object in the hierarchy. We need to set the initial state when is played when the scene / game begins. The Trigger On has already been set to Start, so we just need to add the Off variation of our Switch Group
* If we take a look at the AkAmbient script, it has been set up slightly differently. Instead of the Position Type being Simple\_Mode, it has been set to Large\_Mode. This contains different points where audio can emit from, but still retain its spatialization

(At this point, don’t forget to add the play event to the Name section!)

* In this instance, the emitters have been placed in the 4 corners of the object. When the player stands on the outside of the cube, it will provide directionality of the audio still. When the player moves onto the cube, the audio will now change and the player will be effectively surrounded by the new audio
* Play around with the placement of the emitters or add more to see what happens

**Efficiency with Unity and Wwise:**

We’ll now add an object to the scene and make it into a prefab:

* Right click in the hierarchy and select a 3D object (I would recommend a sphere, cube, capsule or cylinder)
* Make sure the position is set to 0, 0, 0 in the transform position (located at the top of the inspector when the new object has been selected)
* Play around with the Scale settings until you get the size and shape you like
* Add an AKAmbient script to the object, add a rigidbody and select some audio to add to it
* Drag the object from the hierarchy into the prefab folder in the Project tab at the bottom.
* You can now delete the object from the scene.
* The prefab object can be added to the scene as many times as you like and it will have the settings and additional scripts which were implemented, prior to it being saved as a prefab. This is a great time saver if you have a lot of one type of object you want to implement with the same settings

Whilst we can make prefabs of objects that play the same audio and displays efficiency from a construction point of view, we can also make the demand on the CPU more efficient.

* Add 3 more objects spaced around the playground as you wish
* Select all of the objects by using Shift + left click or CTRL + left click (one at a time)
* Add the AKAmbient script as usual, add a rigidbody and select audio to add to it
* On this occasion, we will change the Position Type to be “Multi Position\_Mode”
* When we play the scene, the audio won’t be too dissimilar to what was implemented in the previous section. However, as we are not creating multiple instances of the same audio being played (in this way, we only have one iteration of the audio playing from multiple positions) it reduces the impact that it has on the CPU

Important – Whilst it is possible to create prefabs of objects with the Multi Position\_Mode, this does ultimately flag up errors in the console window. The audio will still play without issue in the game engine, but issues might arise when building your games to the device.

**Random Container (Wwise) and Custom Scripting (Unity):**

You will notice that within Wwise there was some audio already implemented in the form of footsteps and when you play the scene, footsteps can be heard when the character moves around. We will take a closer look at how this has been achieved:

* In Wwise, you’ll notice that the footstep audio is encased within a different type of container. This is a random container, which will randomly play the different audio files when it is triggered
* You can add as many audio files to the random container as you would like
* You will notice that there is a new addition to the General Settings tab; the Play Type box
* In this box, we can set the Random type to Standard or Shuffle, as well as setting it so that it does not repeat the same audio clip after it has already played

We’ll take a look at how this has been incorporated into Unity scripting:

* Go to the Player Capsule and look in the inspector for the script “First Person Controller”. At the bottom of the script’s details in the inspector, you’ll notice a section called “Wwise Events”. We can access Wwise attributes from premade/our own custom scripts by adding them to the code
* Right click on the script’s name and click “Edit Script”. We will take a look at how the footsteps have been incorporated and later on, take a look at making our own custom script to play some audio
* Once visual studio has loaded, take a look at the script. All of the details contained within are used to make the player capsule move around the playground. Whilst it might seem like a lot, we will only be looking at a few sections for the audio implementation

At the top of the script (Class) are where a number of different parameters have been created for use in moving the player. Some examples are floats and bools which will be described in more detail later, when we are creating our own script. (Descriptions of parameters can be found in the custom script named “CustomScript” which we will use later.)

* Towards the bottom of this section, you will see a section called “Wwise” which is written in green with // in front of it. The // symbol allows us to add comments to our scripts for descriptive purposes and allow others to understand the intended purpose of our scripts/particular sections
* A computer screen shot of a program

  Description automatically generatedLocate the “Move” method. This will be written as “private void Move()” further down the script. We will focus on the second highlighted in the screenshot below
* In this section, we utilize a number of “if” statements, which are used to check the status of certain parameters and action something if those conditions are met
* For example, the if statement (!footstepIsPlaying && !isJumping) will check to see if the Boolean values are false and if they are, they will play the associated audio file from the object, change the value of footstepTime to be equal to Time.time and change the Boolean footstepIsPlaying to true
* If the conditions in the “if” section are not met, then it will move onto the “else” section

I have not gone into too much detail with this script but if you have any questions about it, please feel free to ask!

We will now take the opportunity to create our own script and access Wwise from it:

* In the Project Explorer, there is a folder called Scripts. Go into this folder and you will see a file called “CustomScript” which is one I prepared beforehand
* Double click on the script and it will open Visual Studio
* You’ll notice there are some parameters which have been added at the top with some comments in green, just to describe

We’ll add a few parameters to get something working in the Unity scene:

* First, let’s add the float fTimer to the Start method and assign it to zero. We do this because, as we intend to use this as a timer, it will need to start at zero from the very beginning

private void Start()

{

fTimer = 0;

}

* Next, we will update the timer in the Update function, incrementing it using Time

void Update()

{

fTimer += Time.deltaTime;

}

* Next, we will populate the function which will cause the audio to be played. To explain what is occurring further; iRandom is used to pick a random number between 0 and the amount of objects which are stored within the headSpace array. It will then check to see if a button on the keyboard has been pressed (in this example M, but you can change it to a different key) via an “if” statement. In the next if statement, it will check to see if the value of fTimer is greater than that of fCountdown. If it is, it will play the event assigned to myAudioOne and randomly select one of the objects placed in the headSpace array, produce a message in the console to advise that the button has been pressed and reset the value of fTimer back to 0 for the next iteration to begin

public void GenerateAudio()

{

iRandom = Random.Range(0, headSpace.Length);

if (Input.GetKeyDown(KeyCode.M))

{

if (fTimer > fCountdown)

{

myAudioOne.Post(headSpace[iRandom]);

Debug.Log("M was pressed");

fTimer = 0;

}

}

}

* Finally, we need to call the GenerateAudio method in the Update method, so that it can be checked each frame and update the random number selected for iRandom

void Update()

{

fTimer += Time.deltaTime;

GenerateAudio();

}

* Save your script and return to Unity. Play the scene and press the button you assigned in the script. (The script is already attached to the Capsule from the hierarchy, within the PlayerCapsule.) Currently, the countdown is set to 2 seconds, so the event will only be playable after 2 seconds each time the button is pressed. Play around with the values, add more head spaces, etc. and experiment with it

**Buses:**

Much like in your favourite DAWs, it’s possible to bus effects onto your audio / events in Wwise. We’ll take a look at adding a reverb bus and applying it to our scene in Unity.

* Go back to Wwise and make sure you’re on the Audio tab in the Project Explorer
* In the explorer is a section called “Master-Mixer Hierarchy” which already includes a Master Audio Bus
* We can add additional Audio Busses to this by right clicking on it and adding an Audio bus. When this is added, right click it and rename this to something else (perhaps Reverb, but you can name it whatever you like)
* We can then add Auxillary busses to this in the same fashion, right click and add an Auxillary Bus and rename this to “Tunnel Bus” as we intend to add reverb to the tunnel in the playground (for now!)
* In the General Settings tab, make sure that the “Use game-defined aux sends” box is ticked

Let’s change over to the Effects tab. In this tab we can add a number of different effects to the aux bus to be applied to our audio events in Unity.

* We will start with something simple; click on the symbol to add an effect and locate “Matrix Reverb”
* You can create a new type of reverb, use a default (custom) effect or use some premade effects already implemented into Wwise. We will start with a premade effect called “Tunnel1”
* As usual, save your project and generate the selected files in the Soundbank layout

In order make use of this, we will move back to Unity and utilize the tunnel towards the rear of the playground.

* Click on the tunnel object in the Scene view or locate it in the hierarchy (Environment -> Greybox -> Tunnel\_Prefab)
* You’ll notice that the object already contains a number of box colliders for the walls and roof of the tunnel. We will need to add one more box collider to the structure. Ideally, we would like it to cover the interior of the tunnel (the section your character is likely to walk through) for the reverb effect to occur. On this occasion, it will be left to your judgment on how best to adjust this
* You can adjust the size of the collider by using the Edit Collider button (this is located in the top most collider and an image is provided on the next page) which allows you to manually adjust the collider’s size in the scene by clicking on the small green squares on each side. Alternateively, if you hover over the X, Y and Z on the size attribute on the collider and use a slider to change the value (as well as having the ability to manually change the values, as well)

A screenshot of a computer

Description automatically generated

* Make sure that the “Is Trigger” box is ticked for your new collider only
* Add a rigidbody to the object
* Next, we need to add an AkEnvironment script to the object. We use this to add an environmental effect, such as reverb, to an object
* The only thing you need to change/add to the script is in the “Name” section, we need to add the tunnel reverb aux bus we made in Wwise
* When you now play the scene and move through the tunnel, you should notice that a reverb effect has been added to the footsteps of the player and when you leave the tunnel, they return back to normal

**Ducking:**

Now that we’ve been introduced to busses in Wwise, I wanted to quickly show you the auto-ducking feature that can be used. In this instance, we’ll retroactively use the feature to create two busses (one for the footsteps and another for our sound boxes) and have it so that when the player is moving and the footstep sounds are playing, the sound boxes with be quieter.

* As before, we’ll add two new audio busses to the Master Audio bus to represent the footsteps and sound boxes. You can name these whatever you like, as long as they can be identified for their intended purpose
* Click on the bus created for the footstep audio and click on the “Auto-ducking” tab located with general settings, effects, etc.
* We’ll leave the “Recovery time” and “Maximum ducking volume” as they are. Click in “Insert” and locate the sound box bus that we made, adding it to the Busses section
* You’ll notice we can adjust the volume of the sound box bus and the audio fade out/in. Change the volume to -96 for the time being and the fade out/in values to 1 (these can be changed later if you want to)
* Locate the audio you want to assign the busses to (for the footsteps, we’ll want to select the top level container) and assign the relevant bus in the “General Settings” tab
* Save your Wwise project and generate the selected files as usual
* Go into Unity and play the scene. You’ll notice that when you walk near the boxes which the bus has been applied to, they will be silent as you move around and when you stand still, the audio will fade in until you move again
* Play around with the volume / fading for the auto-ducking and see if you can get a nice balance you’re happy with

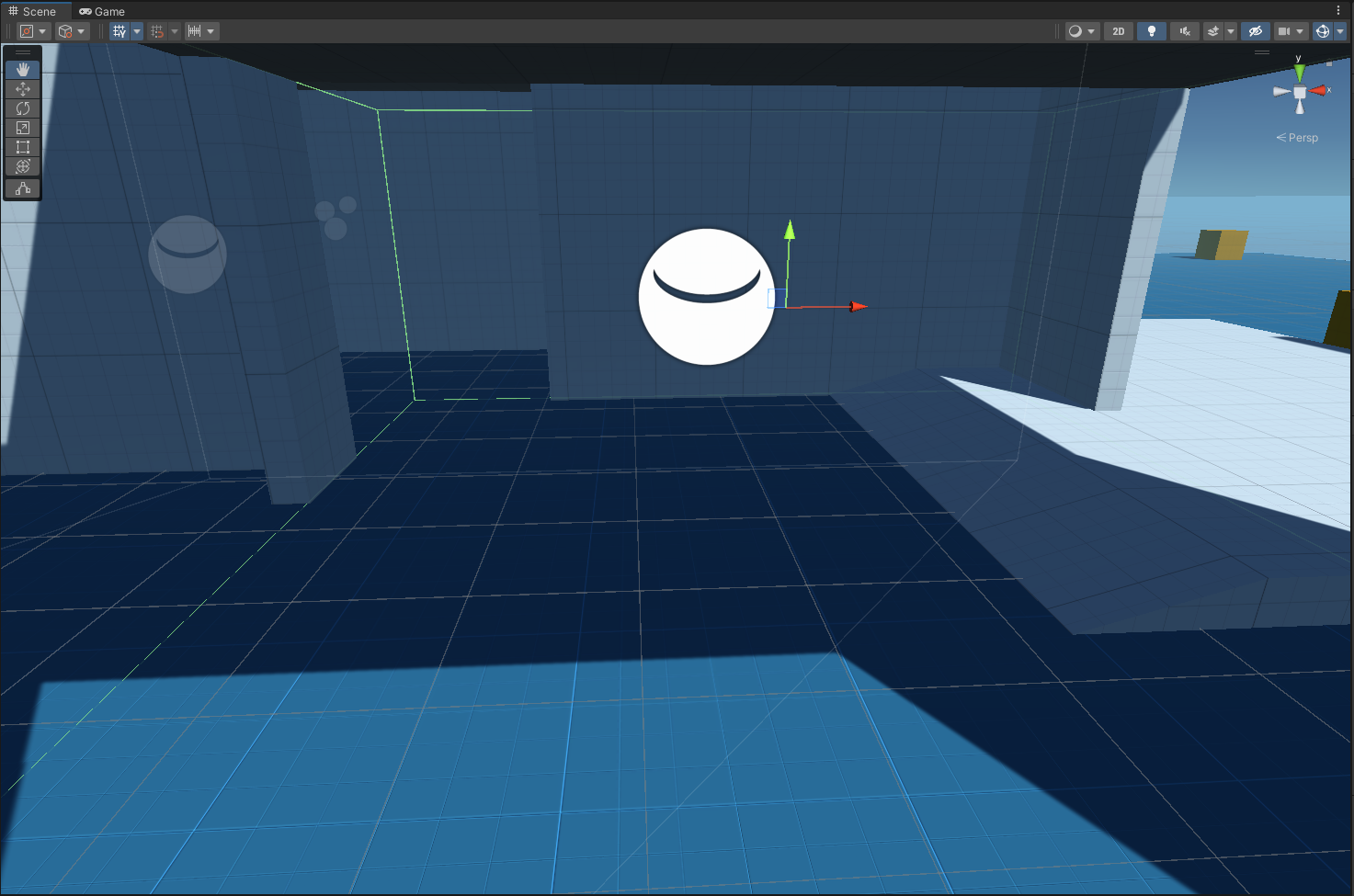
Now that we have looked at some basic approaches to bussing, we’ll make it slightly more complicated. We will look at having two different types of reverb and the ability to smoothly transition between them from one interior location.

* Let’s add two more reverb aux busses in Wwise and select some different matrix reverb effects to make sure there is a distinguishable difference (alternatively, feel free to make your own and have a play around with how it sounds)
* Remember to tick the “Use game-defined aux sends” tick box
* Save the project and generate the selected files in the SoundBank tab

Moving back to Unity, we’re going to use the larger building in the centre of the playground, but use the section underneath for this implementation.

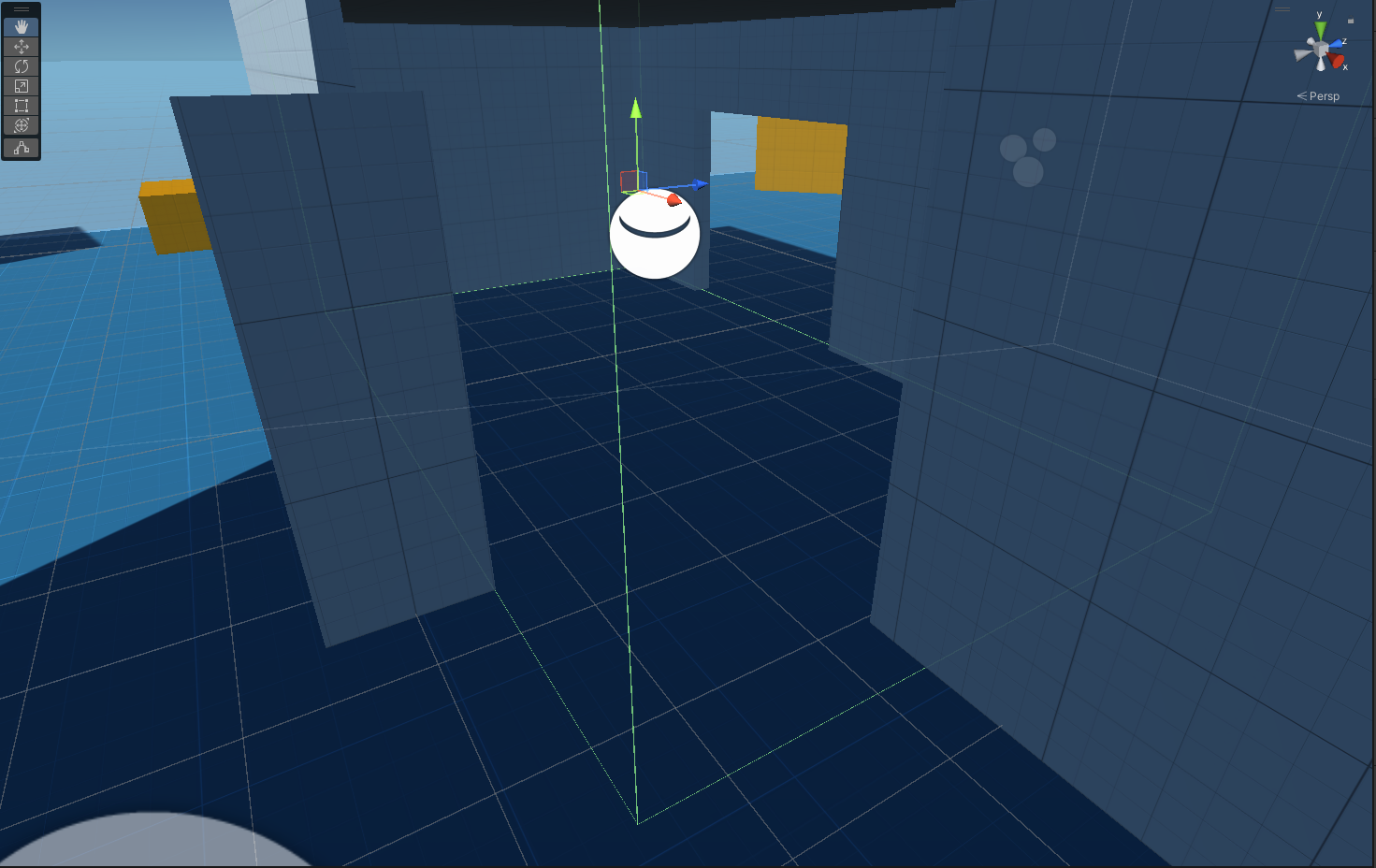
* First, we will create an empty game object and set the position to (0, 0, 0) and name it something appropriate
* We will then make 3 child objects attached to this. We are going to use these to set up a number of colliders which act like trigger zones, rather than having physical objects the player interacts with
* You’ll want to position two of the objects in the centre of a room (one for each) and the third object in the space between the two rooms
* Attach a box collider to each of the objects and set it as a trigger (tick the “Is Trigger” box)
* On this occasion, I will leave it to your own judgement on how to set up the colliders but here is some guidance; the two room colliders need to be set so that they cover each of their respective rooms (it doesn’t matter if they overlap in the section where they cross over). For the third collider, you will want it to overlap both of the other colliders between the archway
* If they are not already turned on, you may want to have “Gizmos” enabled (I have provided some screenshots on the following pages to show what the colliders could look like and where the gizmo button can be located)

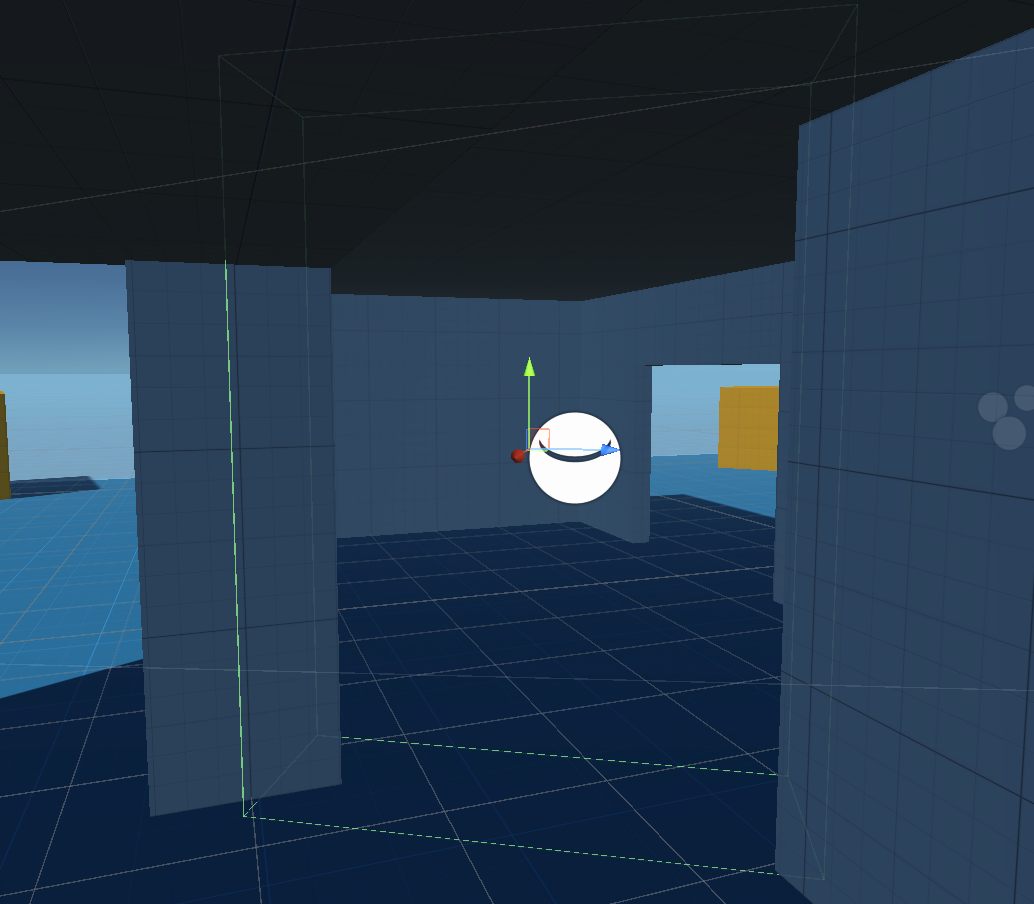
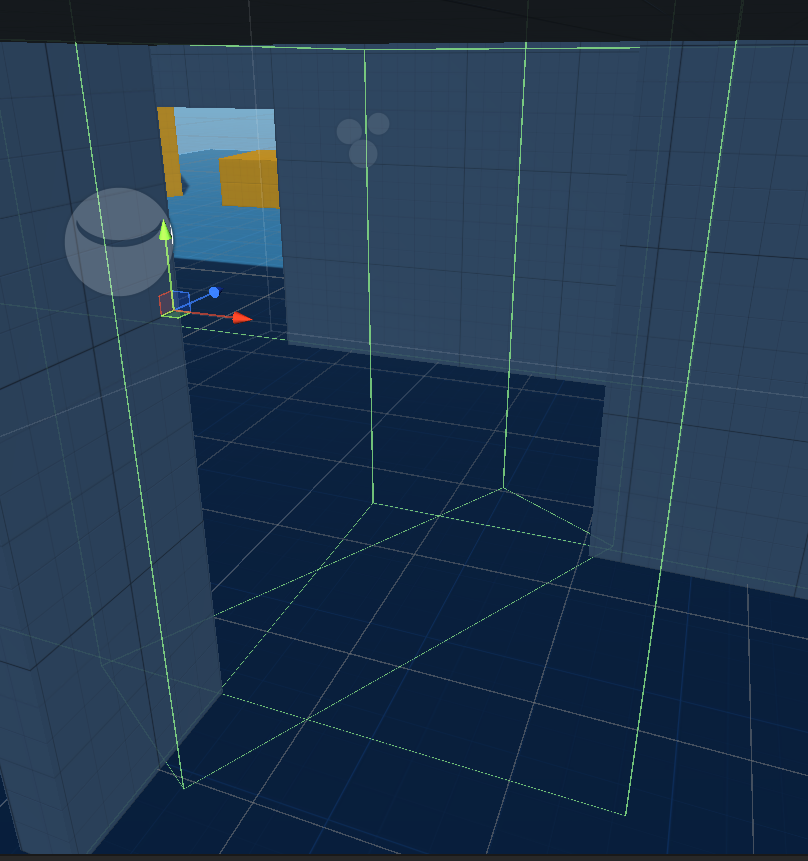
I’ve provided a series of screenshots on the following pages to give an idea of what we intend to make, but please feel free to try your own interpretation or play around with it!



You can click this button to switch from the object movement input to object rotation

This is the Gizmo button





* For the specific room objects, we want to add the AKEnvironment scripts to these and select the reverb bus we want to associate with each room
* On the transitional object, we want to add an AKEnironmentPortal object
* You’ll notice there are a couple of dropdown menus for the environments and the axis. In order to populate this script, change the axis from X to Y and then back to X which will automatically populate the environment objects (you cannot simply select the environments from the dropdown as this is not an option)
* We want this to be on the X axis as this is the axis on which the player will move between the two rooms in the playground
* Play the scene and walk through the rooms, notice the differences in the reverbs but also that when you pass through the centre, the transition between reverb effects is pretty much seamless

This concludes the worksheet. These are just a few examples on how to integrate audio spatially into games.

If you have any questions or need any assistance with the worksheet or any other game engine / Wwise related things, please feel free to contact me on the following email address and I will get back to you, as soon as I can:

[**ian.corkill@dmu.ac.uk**](mailto:ian.corkill@dmu.ac.uk)

Thank you for taking the time to attend the workshop and I wish you the best of luck for the future!