David Abbott

Using FMOD Studio with Unity

Bachelor of Business Administration

Autumn 2018

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Tiivistelmä

Tekijä: Abbott David

Työn nimi: FMOD Studion käyttö Unityn kanssa

Tutkintonimike: Tradenomi, Tietojenkäsittely

Asiasanat: FMOD Studio, äänimoottori, väliohjelmisto, Unity, äänisuunnittelu

Tämä opinnäytetyö käsittelee FMOD:n käyttöä yhdessä Unity pelimoottorin kanssa. Opinnäytetyössä annetaan ohjeet videopelien perusäänijärjestelmien sekä joidenkin enemmän kehittyneiden järjestelmien luomiseen kuten äänijärjestelmä askeläänille. Tälle opinnäytetyölle ei ole ulkopuolista tilaajaa. Opinnäytetyön lähteet koostuvat FMOD-käyttöoppaasta sekä muista aiheeseen liittyvistä verkkosivuista. Tässä työssä on käsitelty FMOD Studion ja FMOD Studio API:n tärkeimpiä ominaisuuksia käyttämällä lähdemateriaalia ja kokemustani useista projekteista, joissa olen käyttänyt FMOD Studio ohjelmistoa.

Opinnäytetyön teoreettinen osa sisältää lyhyen katsauksen FMOD:n ominaisuuksista, lisenssoinnista ja miten se tukee eri alustoja, tiedostomuotoja ja pelimoottoreja. Empiirisessä osassa on esimerkkejä useimmista FMOD Studio -ominaisuuksista ja siitä, miten niitä voidaan käyttää yhdessä Unity-pelimoottorin kanssa.

Ensimmäisessä kappaleessa esitetään opinnäytetyön tavoitteet ja viimeisessä kappaleessa esitetään opinnäytetyön johtopäätökset, jossa myös mainitaan eräitä aiheita, jotka jätettiin pois opinnäytetyön aihealueen rajoittamisen vuoksi.

Abstract

Author: Abbott David

Title of the Publication: Using FMOD Studio with Unity

Degree Title: Bachelor's Degree in Business Information Technology

Keywords: FMOD, audio engine, middleware, Unity, sound design

This thesis covers the use of FMOD Studio in conjunction with the Unity game engine to create the basic audio systems most needed in a typical video game, as well as some of the more advanced systems that might be needed such as a footstep audio system. This thesis was not commissioned by an outside party. The sources of the thesis consist of the FMOD user guide, as well as various other websites related to the topic. Using the source material and my experience working with FMOD Studio on multiple project, the most important features of FMOD Studio and the FMOD Studio API have been covered in this thesis.

The theoretical section of this thesis contains a brief overview of FMOD, what its features are, the licensing terms under which it can be used and the support for different platforms, file formats and game engines it provides. The empirical section contains in-depth explanations and examples of most of the features of FMOD Studio and how they can be used in conjunction with the Unity game engine.

The first chapter contains the objectives and goals of this thesis while the last chapter contains the conclusions of the thesis including certain topics that were left out due to limiting the scope of the thesis.

Contents

List of Symbols

API – A software intermediary that allows two applications to communicate with each other. The acronym stands for “Application programming interface”.

Bus – A channel that collects together multiple audio signals for overall processing.

Digital audio workstation – A type of application commonly used when recording audio.

Dry & wet levels – Determines the ratio between the original and the affected signal.

Envelope generator – Produces a rising and falling signal that shapes the audio.

Fader – Any knob or button that slides along a track

Audio mixing – Combining multiple sounds in a way that is pleasing to listen to.

Pitch modulation/shifting – Raising or lowering the pitch of a sound.

Plug-in – A way of adding a new feature to existing software.

Return – Used to affect an audio signal with different types of audio effects.

SFX – Sound effects.

Send – Used to send a signal to a return.

VCA – A Voltage controlled amplifier is used to control volume levels within FMOD Studio.

Polyphony – The number of sounds allowed to be playing back at the same time.

Timecode – The playback position depicted in hours, minutes and seconds.

1 Introduction

The objective of this thesis is to introduce the FMOD Studio software along with its API, explain what it is capable of, teach the basics of how FMOD Studio functions and then demonstrate some of the more advanced features by using it in conjunction with the Unity game engine. This thesis assumes that the reader has the knowledge required to use Unity and therefore it will mainly focus on FMOD and the C# programming needed to achieve certain effects in-game. The thesis is written in a guide-like fashion where the reader can create their own Unity project and follow along with the instructions to learn how to implement sound effects, adaptive music and advanced systems designed to handle the audio for footsteps and other more complicated SFX.

2 What is FMOD?

2.1 General overview

FMOD is an audio engine developed by Firelight Technologies for use in software development. It can play sounds in many different formats on a diverse array of operating systems. It is supplied to the end user as a programmer's API and an authoring tool called FMOD Studio which is similar to a digital audio workstation. (Wikipedia, n.d.)

FMOD has been used in multiple highly regarded video games including BioShock, Deus Ex: Human Revolution, Alan Wake, Subnautica and more. (Firelight Technologies, n.d.)

2.2 Features

FMOD consists of the following technologies: (Wikipedia, n.d.)

- FMOD Studio – An audio creation tool for games, designed like a digital audio workstation.
- FMOD Studio run-time API – A programmer API to interface with FMOD Studio.
- FMOD Studio low-level API – A stand-alone programmer API, with a simple interface for playing sound files, adding special effects and performing 3D sound.

The FMOD sound system has an advanced plug-in architecture that can be used to extend the support of audio formats or to develop new output types, e.g. for streaming. (Wikipedia, n.d.)

Some of the features included in FMOD Studio are pitch modulation, volume modulation, snapshots for saving mixer settings, support for adaptive music and real-time mixing.

2.3 Licensing

A license is not required to evaluate FMOD or to use it during game development. It is also not needed to distribute a game or other application that is for personal, educational or non-commercial use since the developer of said software will be covered by the End User License Agreement (EULA). (Firelight Technologies, n.d.)

In terms of commercial use, FMOD is available under three different license schemes: (Firelight Technologies, n.d.)

- FMOD Indie License, a bottom level license for games with development budgets of less than $500k USD. This license is free as long as only one game is released per year, otherwise, the license fee is $2000 USD per game.
- FMOD Basic License, a mid-level license for games with development budgets between $500k USD and $1.5m USD. The license fee is $5000 USD.
- FMOD Premium License, a top-level license for games with development budgets over $1.5m USD. The license fee is $15000 USD.

If the product FMOD is being used for is not a game, Firelight Technologies can be contacted for custom licensing options. (Firelight Technologies, n.d.)

2.4 Support

FMOD Studio supports most platforms, audio formats and game engines. FMOD also contains support for AMD TrueAudio and Sound Blaster hardware acceleration. (Connect.creativelabs.com, n.d.)

2.4.1 Platforms

FMOD is written in portable C++, and can thus run on many different PC, mobile and gaming console platforms including: Microsoft Windows, macOS, iOS, Linux, Android, BlackBerry, Wii, Wii U, 3DS, Nintendo Switch, Xbox, Xbox 360, Xbox One, PlayStation 2, PlayStation 3, PlayStation 4, PlayStation Portable, PlayStation Vita, and Google Native Client. (Wikipedia, n.d.)

2.4.2 File formats

FMOD can play back the following audio formats: AIFF, ASF, ASX, DLS, FLAC, FSB (FMOD's sample bank format), IT, M3U, MIDI, MOD, MP2, MP3, Ogg Vorbis, PLS, S3M, VAG (PS2/PSP format), WAV, WAX (Windows Media Audio Redirector), WMA, XM, XMA (only on the Xbox 360), as well as raw audio data. (Wikipedia, n.d.)

2.4.3 Game engines

FMOD has been integrated as a primary sound-effects system into the following video game engines:

- Caffeinated3D from Gajatix Software Ltd. (Wikipedia, n.d.)
- Unity from Unity Technologies, although the full implementation of FMOD in this guide surpasses what can be done with the basic implementation that exists in Unity. (Wikipedia, n.d.)
- Unreal Engine 3 from Epic Games (Unreal Technology, n.d.)
- Unreal Engine 4 from Epic Games (Unreal Technology, n.d.)
- CryEngine from Crytek (Crytek.com, n.d.)
- Torque Game Engine from GarageGames (Wikipedia, n.d.)
- BigWorld Technology from Bigworld Technology (Bigworldtech.com, n.d.)
- Scaleform from Scaleform Corporation (scaleform, n.d.)
- Havok Vision Engine (havok.com, n.d.)
- Source from Valve Corporation (Valve Corporation, n.d.)
- HeroEngine from Idea Fabrik Plc. (HeroEngine wiki, n.d.)

3 Setup

3.1 Downloading the necessary files and installing FMOD Studio

After registering for a free account at http://www.fmod.com, click on the Download button in the header of the page. The latest version of FMOD Studio will be available for download here as well as the integration package for Unity. The version of the integration package must match the version of FMOD Studio. Download both and then run the installer for FMOD Studio and follow the instructions. The application can be installed in any location.

3.2 Setting up our Unity project

Create a new Unity project and open the Asset Store by clicking on “Window - Asset Store”. Search for the free Standard Assets by Unity Technologies in the Asset Store. Download the package and import all files into the project. Please note that the Standard Assets pack might have deprecated elements that could cause warnings to appear but they should be safe to ignore.

Navigate to “SampleScenes – Scenes” and open up the CharacterThirdPerson scene. Then use the hierarchy window and navigate to “Cameras – FreeLookCameraRig – Pivot – MainCamera” and remove the Audio Listener component. After this, click on Add Component, navigate to the FMOD Studio category and click on FMOD Studio Listener.

Also, delete the GeometryDynamic object in the hierarchy and then navigate to the GroundObstacles child object under “GeometryStatic - GroundObstacles” and delete that too.

Save the scene.

3.2.1 Integrating FMOD with Unity

The FMOD integration package comes in the form of a unitypackage file. To import it into the project, click on “Assets” in the Unity toolbar and then click on “Import Package -

Custom Package” in the drop-down menu. Select the package from the hard disk and click on “Open” after which click “Import” to import all files into the Unity project.

3.3 Creating an empty FMOD Studio project and setting the project path within Unity

Open FMOD Studio and click on “New Project”. Then click on “File - Save As…” and write in a name for the project. Navigate to a save location and then click “Save”. Now click on “File - Build All Platforms” and then “File - Save”.

If the Unity project is open, there should be a message in the Console that reads “FMOD Studio Project path not set”. To rectify this, the project path must be set within Unity. To do this, click on the new “FMOD” tab in the Unity toolbar and then click “Edit Settings” in the resulting drop-down menu. The settings will show in the inspector. Leave the connection type on Project and click on “Browse”. Find the FMOD Studio project folder created earlier and select the fspro file inside.

4 The basics

4.1 Platform and audio quality settings

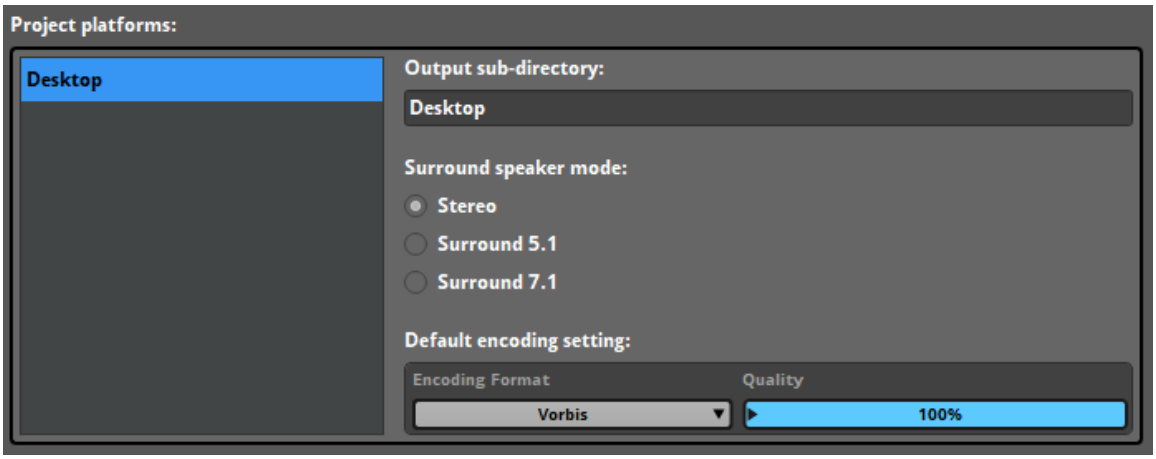

Figure 1. Project platform settings.

By clicking on “Edit” and then “Preferences” and then clicking on the “Build” tab within the window that appears, it is possible to adjust the settings for which platforms are needed, if surround sound is needed, the encoding format for each platform and the quality of the audio for each platform. More platforms can be added to the list by right-clicking the darkened area on the left-hand side of the settings.

For the purposes of this guide, the settings should be adjusted to reflect the figure above.

4.2 Creating events

To add basic sound effects or ambient sounds to a scene, the audio files must first be added as events in FMOD Studio. To do this, open FMOD Studio and then open the empty project created earlier.

Figure 2. The New Event and New Folder buttons.

To create a simple 2D or 3D sound effect event to be played in-game, left-click on the “New Event” button in the bottom left corner of the user interface and then select either a 2D or 3D event from the list that appears. The difference between a 2D and a 3D event is that 2D events are not placed in the world which makes them particularly suitable for UI sounds and music. 3D events are used for sounds emitted by objects or characters within the world.

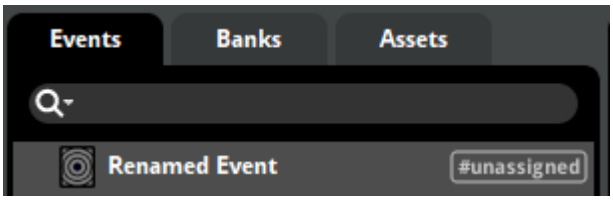

Figure 3. A newly created and renamed event.

When an event is created, it will be added to the list on the left side of the UI and it can be renamed straight away using the keyboard. If for some reason the event isn’t high-lighted, double-click on the event to rename it.

Next to the name of the event, a hashtag that reads “#unassigned” can be seen. This means that the event hasn’t been assigned to a bank. A bank is a file built by FMOD Studio that contains the audio information that is used by Unity to play the events creat-ed. Multiple banks can be created (e.g. Guns, Creatures, Dialogue, Music) and they can be unloaded or loaded from memory within the C# code written in Unity. For now, stick with the pre-existing master bank by right-clicking on the Event created earlier and then hovering over “Assign to Bank” in the drop-down menu to reveal the different banks available. Click on “Master Bank“ to assign the event to it.

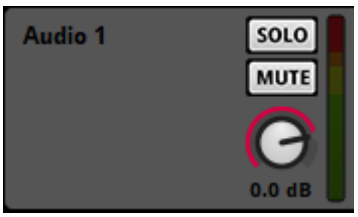

Figure 4. The Audio 1 track.

One empty audio track named “Audio 1” should be visible in the event editor. If it isn’t, select the event in the list by clicking on it. Double-click on the name of the track to re-name it. Additional audio tracks can be created within the event by right-clicking on an existing track or the master track and then clicking “Add Audio Track” in the resulting drop-down menu.

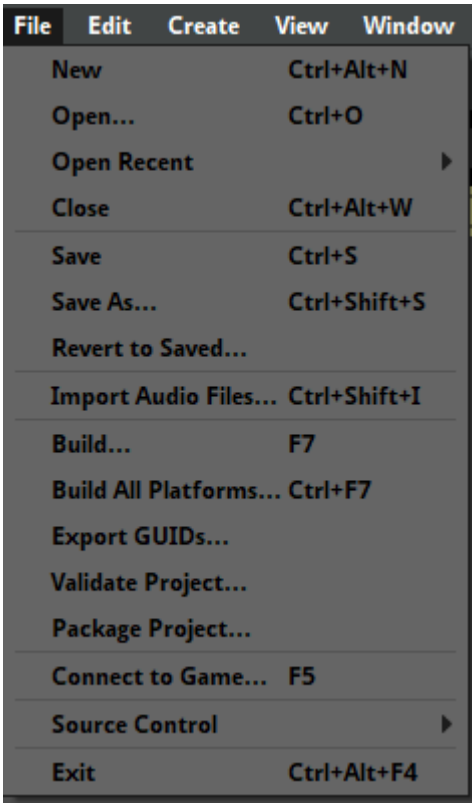

Figure 5. The File menu.

After creating or editing an event, the bank containing it must be rebuilt in order for Unity to be able to access the event. This can be done by clicking on “File” on the toolbar and then “Build” or “Build All Platforms” in the drop-down menu. Finally, it’s a good idea to save the FMOD project using the “Save” or “Save As…” buttons.

4.3 Editing events

4.3.1 Instruments

Instruments are trigger regions placed on audio tracks within events and when the playback position in the event reaches a trigger region, the audio placed within the instrument will play. If a fade-in or fade-out is required, simply drag the top left or right corner of the trigger region to create a fade.

By right-clicking on an audio track, a list of instrument options will appear and by right-clicking on an instrument, the settings for it will appear at the bottom of the FMOD Studio UI:

- Add Single Instrument

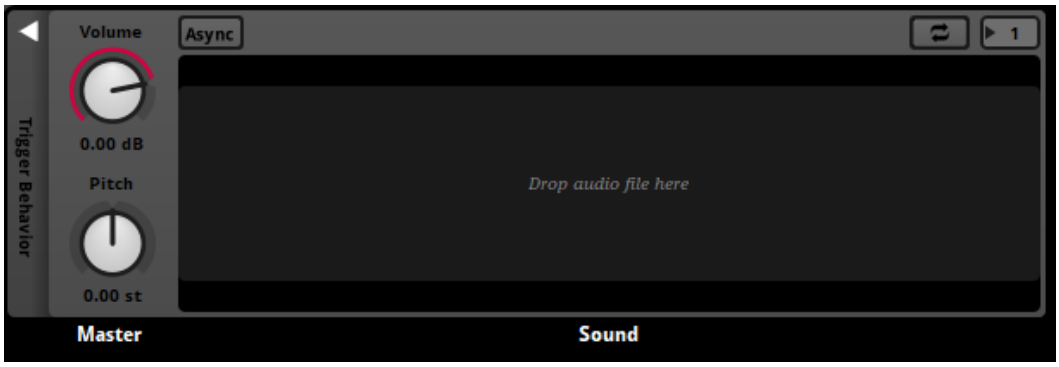

Figure 6. Single Instrument settings.

Used for a single audio file. The audio file can be added by dragging and dropping it into the area named “Sound”.

  - The volume and pitch controls are for adjusting the volume and pitch of the instrument.

  - By clicking on the “Async” button, the instrument will be in asynchronous mode which means that the audio file within the instrument will finish playing even if the length of the instrument is not as long as the audio file.

  - The button depicting two arrows is used to loop the instrument while the number field next to it will determine how many times the instrument should loop.

- Add Multi Instrument

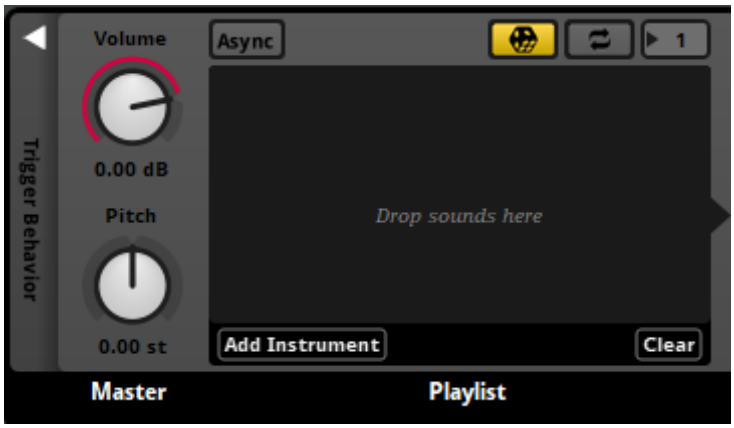

Figure 7. Multi Instrument settings.

A playlist of single instruments, event instruments or programmer instruments.

  - The die icon is lit by default which indicates that the order of playback is randomized.

- Add Event Instrument

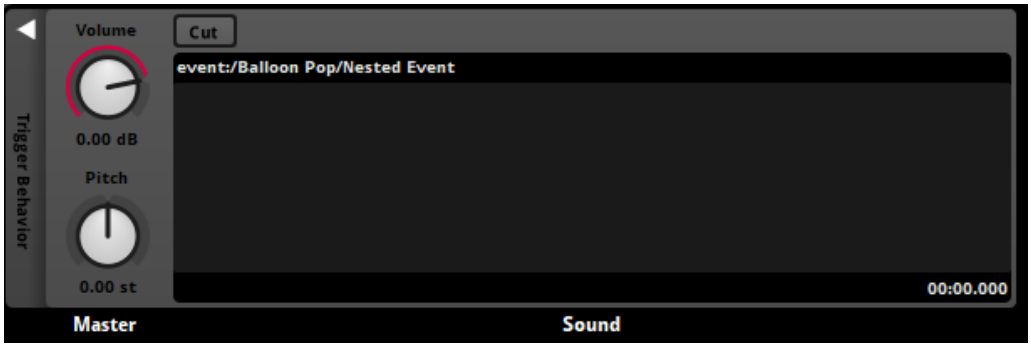

Figure 8. Event Instrument settings.

Used to make nested events. By double-clicking on the instrument, FMOD Studio will open up the editor for the nested event. When playback is triggered, the output will be routed to the parent event instead of directly to the mixer.

  - The event instrument is asynchronous but if the “Cut” button is enabled, the instrument’s output will be cut short when the playback position reaches the end of the instrument’s trigger region.

- Add Scatterer Instrument

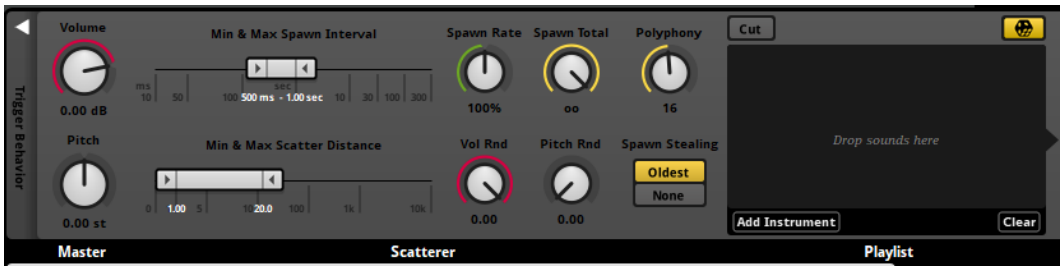

Figure 9. Scatterer Instrument settings.

A playlist of multiple audio files but with spatial and temporal randomization which means that the sound will trigger within different locations in 3D space.

  - The sliders control the time between sounds played and the distance between sounds in 3D space.

  - The dials control the total amount of spawned sounds, the rate at which they are spawned, the polyphony of said sounds and randomization of the volume and pitch of the sounds.

  - The spawn stealing setting is set to “Oldest” by default which means that if a sound is spawned and the polyphony limit has been reached, the oldest sound still playing will be cut short and be replaced by the new sound.

- Add Programmer Instrument

  Passes a callback to the game’s code when triggered which allows for the audio file to be specified within the code. A placeholder audio file can be assigned using the settings for this instrument.

- Add Plug-in Instrument

  Includes extra instruments based on the FMOD Studio plug-ins that have been installed. The AudioMotors2 and AudioWeather plug-ins are included with FMOD Studio.

- Add Snapshot Instrument

  Allows for snapshot triggering within events. More information on snapshots can be found in chapter 5.6.

4.3.2 Modulation

- Pitch and volume modulation

Figure 10. Modulation controls.

One of the simplest and most useful features of FMOD is the ability to randomize changes in volume and pitch each time an instrument is triggered. This results in more realistic audio for sounds like footsteps or gunshots since the sound effects will never sound exactly the same each time they are played.

This can be done by right-clicking on the volume or pitch controls, choosing “Add Modulation – Random” from the drop-down menu that appears and then adjusting the settings shown in the figure above. They will appear to the right of the existing instrument settings when the modulation is added.

- AHDSR modulation

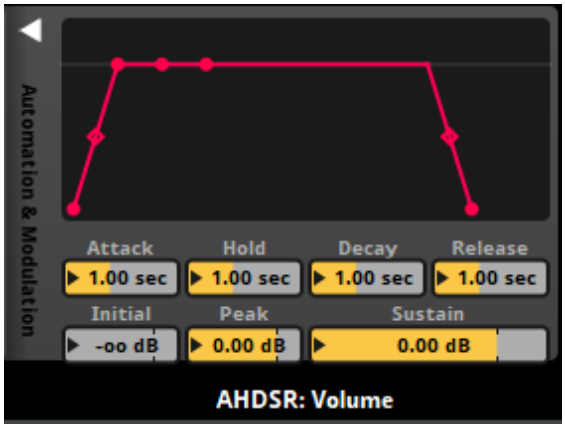

Figure 11. An AHDSR envelope.

The function of an AHDSR envelope is to modulate an aspect of the audio being played back, usually the volume levels over the time it plays.

An AHDSR envelope can be added to an instrument, an audio track or an entire event by right-clicking on a volume control within one of these and then choosing “Add Modulation – AHDSR” from the drop-down menu that appears.

The settings of an AHDSR envelope:

- Attack – This controls how quickly the audio reaches its full volume after it’s triggered. A slow attack will cause the audio to fade-in.
- Hold – This controls the amount of time before the decay portion of the envelope is allowed to begin.
- Decay – This controls how long it takes for the volume level to drop to the sustain level after the initial peak volume.
- Sustain – Instead of controlling time like the previous settings, this controls the volume level of the audio after the decay portion of the envelope is over.
- Release – This controls how quickly the audio fades after it has ended.

It is worth noting that an AHDSR envelope can be used in conjunction with the “Cut” feature found in the settings for some of the instruments.

4.3.3 Parameters

Parameters are one of the most powerful FMOD Studio features as they can be used as trigger conditions for instruments or used in conjunction with automation or logic markers.

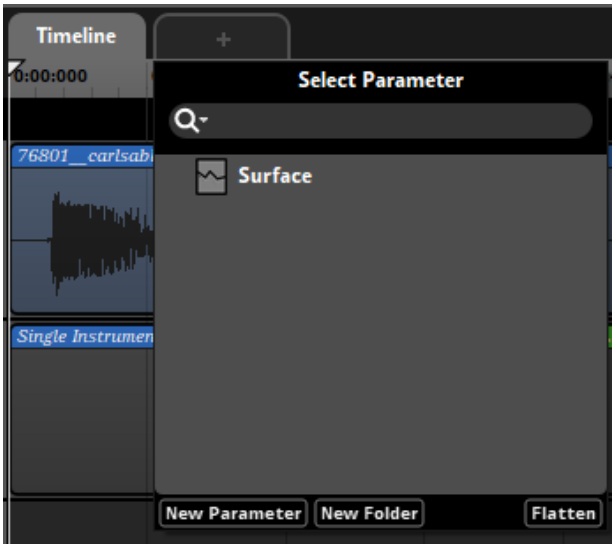

Figure 12. The parameter selection menu.

To add a new parameter to an event, click on the plus symbol next to the word “Timeline” to bring up the “Select Parameter” menu. Then click on the “New Parameter” button within the menu to bring up the menu depicted in the figure below.

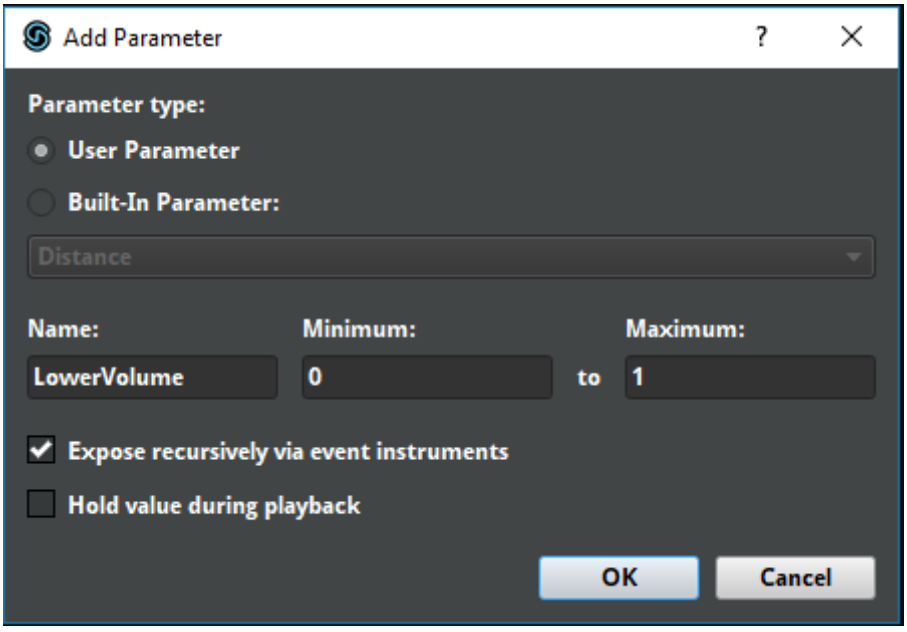

Figure 13. The menu used to add a parameter.

Click on the “Built-In Parameter” radio button and check to see if the desired parameter is already built into FMOD Studio. If not, click on the “User Parameter” radio button and continue on to naming the parameter and setting the minimum and maximum values for the parameter. If the parameter is used by an event instrument, the “Expose recursively via event instruments” checkbox will automatically expose the parameter its parent events. The “Hold value during playback” will prevent the value from being changed while the event is playing back which could be useful in some situations.

Once these settings have been adjusted, click on the “OK” button to confirm them and create the parameter. The parameter will be added as a tab next to the “Timeline” tab and it will be automatically selected for editing. A control for testing the functionality of the parameter will also be added to the right of the timecode as shown in the figure below.

Figure 14. The dial used to test the parameter.

In the following example, the parameter will be used to lower the volume of the audio track by a set number of decibels when its value is 1. When the value is returned to 0, the volume will be restored to its previous value.

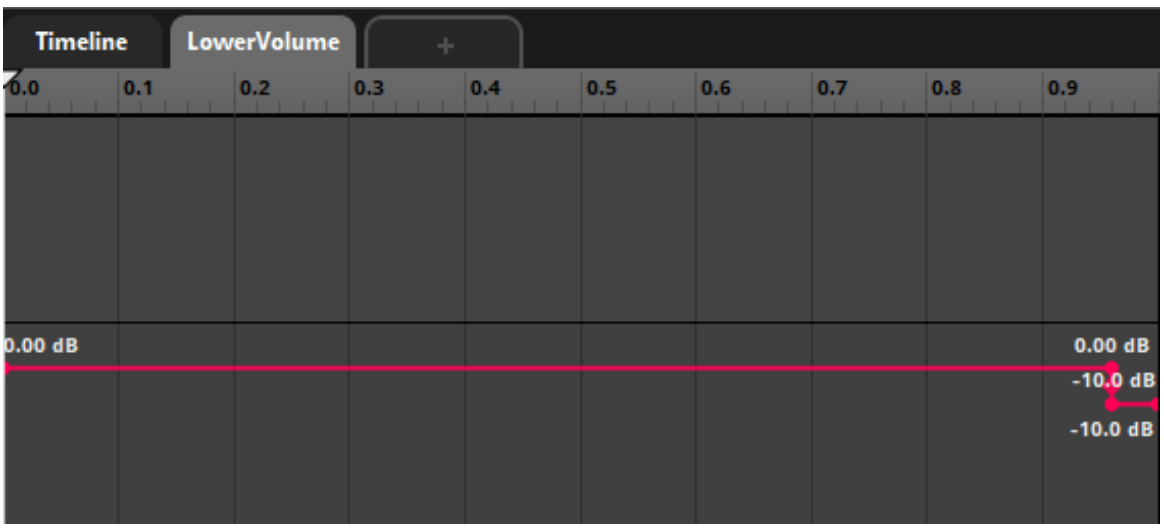

Figure 14. The new parameter used to control automation.

When viewing a parameter, the timeline is replaced with the value range of the parameter. This means that when adding automation to an audio track, the automation curve

can be drawn to fit the parameter values desired. While the parameters tab is selected, right-click on the audio track's volume control and choose "Add Automation" from the drop-down menu. This will create an automation track under the audio track as shown in the figure above. Adjust the automation curve so that there is a volume drop when the value of the parameter is at 1.

The effect of the parameter can be tested by playing the event and then using the control shown in Figure 14 to change the value of the parameter to 1 and then back to 0 while audio is being outputted.

4.3.4 Trigger behaviour settings

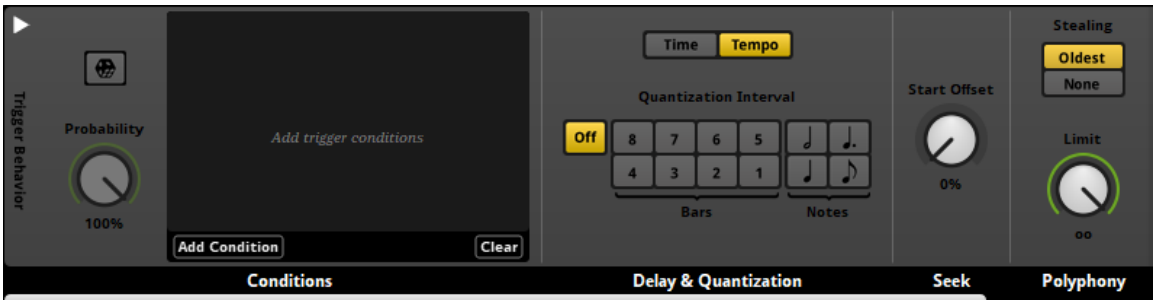

Figure 12. Trigger behaviour settings.

By clicking on the small white arrow in the top-left corner of any of the instrument settings, the trigger behaviour settings for that instrument will open. By clicking on the icon picturing a die, the probability of playback can be adjusted. Under the "Conditions" section, parameters can be added so that the instrument will only be triggered if the parameter is at the desired value. The "Delay & Quantization" section is for setting quantization values if the event has a tempo marker or a delay before the instrument begins to play. Quantization and tempo markers are explained in chapter 8.1. The "Start Offset" control will offset the start point of the instrument.

4.4 Event playback

In Unity, search through the hierarchy for a GameObject to add sound to and select it or alternatively, create a new GameObject for this purpose.

4.4.1 Method 1: Studio Event Emitter

This is the simplest way to add a sound effect or music to a Unity project.

Click on the Add Component button in the inspector and in the drop-down list, click on the FMOD Studio folder and then on FMOD Studio Event Emitter.

Figure 9. The settings for the Studio Event Emitter component.

Under the drop-down lists for Play Event and Stop Event, the most common variants for starting or stopping the event are listed.

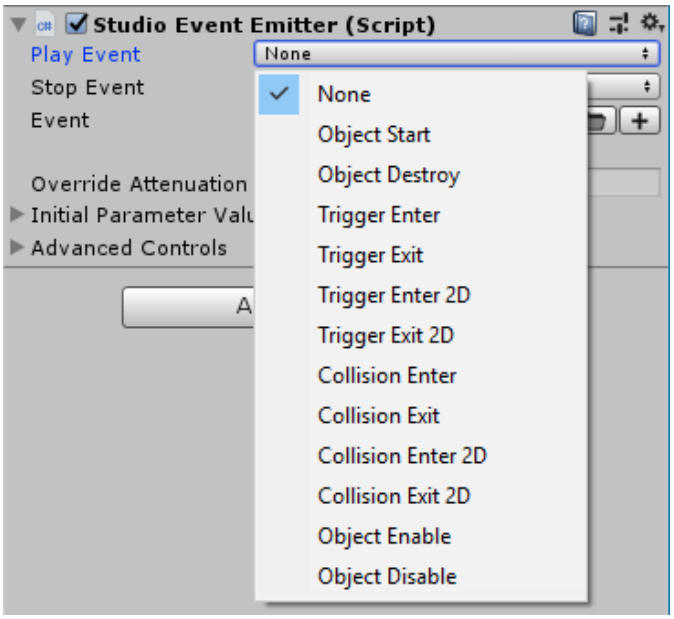

Figure 10. Different options for starting or stopping the event.

- Object Start/Destroy – Triggered as soon as the GameObject exists/is destroyed.
- Trigger Enter/Exit – Triggered when entering/exiting a collision trigger zone.
- Collision Enter/Exit – Triggered when entering/exiting a collision zone.
- Object Enable/Disable – Triggered when the GameObject is enabled/disabled.

The Trigger Enter/Exit and Collision Enter/Exit options also have 2D versions available for 2D colliders and triggers.

The Event field is for choosing which event to assign to the component. By clicking the magnifying glass icon, it is possible to browse through all the events available. The folder icon opens up the same list of events in a separate window and the plus icon is used to create a new event providing that the related FMOD Studio project is open.

4.4.2 Method 2: The FMOD API

This method is more involved but it allows for more complicated scenarios.

While the chosen GameObject is still selected, click on Add Component in the inspector and pick “New script” listed at the end of the drop-down menu. Name the script and then click on “Create and Add”.

4.4.3 Oneshot event

```
//Oneshot
FMODUnity.RuntimeManager.PlayOneShot("event:/Balloon Pop", gameObject.transform.position);
```

Figure 11. Code for a oneshot event.

This type of event will only be triggered once after which it will instantly be automatically released. This also means that the event will not be indexed and so it can’t be affected once triggered.

The arguments for the function include the path to the event as well as the location the event will be played within the game world.

4.4.4 Instanced event

```
//Creating the instance and starting it
eventInstance = FMODUnity.RuntimeManager.CreateInstance("event:/Balloon Pop");
eventInstance.start();

//After starting the instance, changes can be made
eventInstance.setVolume(50f);
eventInstance.setPitch(15f);
eventInstance.setTimelinePosition(1);

//Perhaps most importantly, parameters created in FMOD Studio can be set through code in Unity
eventInstance.setParameterValue("exampleParameter", 5);

//Stopping and then releasing the instance
//The "FMOD.Studio.STOP_MODE.ALLOWFADEOUT" argument will let the SFX fade naturally to a stop
eventInstance.stop(FMOD.Studio.STOP_MODE.ALLOWFADEOUT);

//The "FMOD.Studio.STOP_MODE.ALLOWFADEOUT" argument will stop the SFX immediately
eventInstance.stop(FMOD.Studio.STOP_MODE.IMMEDIATE);

//Releasing the event
eventInstance.release();
```

Figure 12. Code for an instanced event.

This type of event will be indexed which means that changes can be made in code using the variable for the instance. After an instance is no longer needed, it needs to be released manually so that no unnecessary event instances are left causing performance issues.

5 The FMOD Studio Mixer

Open up the Mixer in FMOD Studio by clicking on “Window” in the toolbar and then clicking on “Mixer”. On the left, the routing tab is open by default and all the events created so far are listed within. The main part of the mixer window is the mixing desk which will show all the existing groups, returns and VCAs (Voltage Controlled Amplifiers) as well as the master bus which every audio signal runs through. Note that a return with a reverb effect attached to it exists by default in an FMOD Studio project and it should be visible within the Mixing Desk view.

5.1 Group buses

A new group bus can be created by clicking on “New Group” in the bottom-left corner and then dragging and dropping events into the group. Grouping different types of SFX such as UI sounds, in-game sounds or music is very useful for adjusting the volume of multiple events at once or applying effects such as reverb. Once a group has been created, the group will appear within the Mixing Desk section of the Mixer window.

5.2 Return buses

A new return bus can be created by clicking on “New Return” in the bottom-left corner. Returns are usually used to add FX like reverb or delay to events by adding the chosen FX to the return and then routing either whole groups or individual events to said return. Each group that is visible in the Mixing Desk can be routed to a return using either a Pre-Fader Send or a Post-Fader Send and the amount of signal being routed can also be adjusted which directly corresponds to how much the FX will affect the sound.

To add a Send, click on a group in the Mixing Desk and the signal path for the group will be shown at the bottom of the Mixer window. By clicking on either the minus symbol for pre-fader options or the plus symbol for post-fader options, a drop-down menu will appear where an effect, a send or a sidechain can be added. Click on “Add Send” in the menu and all the available returns will be shown. By clicking on a return in the list, an associated “Send” UI element will be added to the signal path. Use this UI element to

set the amount of signal being sent or use the new setting shown on the group within the main part of the Mixing Desk to achieve the same effect.

5.3 FX

FMOD Studio has multiple effects that can be used to affect audio such as reverb, delay, chorus, EQ, distortion, tremolo and more. The usual method of using FX is to create a return for the effect and then route an audio signal to that return to add the effect to it. This saves processing power since only one effect per return needs to be active.

To add an effect to a return bus, select a bus in the mixing desk and then click on either the minus or plus signs at the bottom of the mixer window to make the drop-down menu appear. Hover over “Insert Effect” in the drop-down menu to make all the choices for different types of FX appear and then choose the one needed. FX can also be added to regular buses or even to individual events in the same way if needed.

5.4 VCAs

A new VCA (Voltage Controlled Amplifier) can be created by switching to the VCA tab in the top-left corner of the Mixer view and then clicking on the “New VCA” button in the bottom-left corner. VCAs are useful for grouping events, group buses and return buses together in order to be able to control the volume of all them together. A common example for games is if there are master volume sliders for dialogue, sound effects and music in the game’s options menu, they should be linked to VCAs controlling the volume of the related events and buses.

5.5 Master bus

The master bus is the final bus which all audio passes through before being sent to the hardware for playback. FX can also be added to the master bus if needed.

5.6 Snapshots

For the next section of this guide, it is necessary to have at least one group with an event placed inside and the group should have a send to the reverb return with the amount of signal being sent set at zero decibels.

5.6.1 Reverb snapshots

Reverb snapshots give us the ability to switch between reverb settings to match different areas within the game in real-time.

First, check the settings of the reverb effect placed on the Reverb Return and make sure that if a certain type of reverb is not specified through the snapshot, no reverb is being added to our sound effects. Open the Mixer window in FMOD Studio and click on the Reverb Return in the Mixing Desk. On the bottom of the UI is the signal chain for this Return.

Find the Wet Level and Dry Level settings on the reverb effect and set them both to zero decibels, this way the reverb will not be applied until a snapshot is active.

To create a snapshot, switch to the “Snapshots” tab in the top left-hand corner. Click on “New Snapshot” at the bottom left and then click on “New Overriding Snapshot” in the list that appears. At this stage, the snapshot can be renamed by typing a name in which should preferably be something that describes the type of reverb or the area this will be used for (e.g. Cave, Hangar, Bathroom).

Select the Reverb return by clicking on it and then right-click on the reverb effect located on the bottom half on the UI. Make sure to right-click on the empty space of the UI element instead of one of the green dials, the full menu will not appear otherwise. In the menu, click on “Scope In” to make the reverb active within the snapshot and to be able to edit its settings. Adjust the settings to something suitable or use one of the presets by opening the previous menu and selecting a preset under “Defaults”. Make sure the Wet Level is set high enough for the effect to be heard. After the snapshot is ready, save and rebuild the banks.

To trigger the snapshot in Unity, create a box collider within the Unity scene and enable the “Is Trigger” function. Add a Studio Event Emitter component to the collider and set

the Play Event to Trigger Enter and the Stop Event to Trigger Exit. Set the collision tag to Player and finally select the snapshot under Event.

5.6.2 Pause snapshot

When entering a pause menu, it is usually desirable to pause the audio of the game as well. A snapshot can be created to achieve this.

Navigate to the Snapshots tab within FMOD Studio's Mixer view and create a new overriding snapshot named "Pause". Right-click on the faders for the groups to be muted and then click on "Scope In" in the menu that appears for each one. Pull the faders down to the bottom so that when the snapshot is active, the sound will be muted for each group.

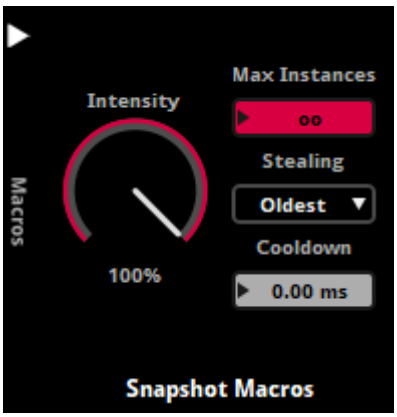

Figure 8. Intensity knob.

Right-click on the Intensity knob located at the bottom-right of the UI under the "Macros" section and click on "AHDSR" under "Add Modulation" in the drop-down menu. The AHDSR modulator and its settings will appear.

Adjust the settings of the AHDSR modulator to have a slow attack and release speed, this will mean that the snapshot's effect on the mix will fade in and fade out accordingly to prevent the audio from cutting out sharply when the game is paused or resumed.

In Unity, click on "Edit" in the toolbar and navigate to "Project Settings" followed by "Input" in the resulting drop-down menus.

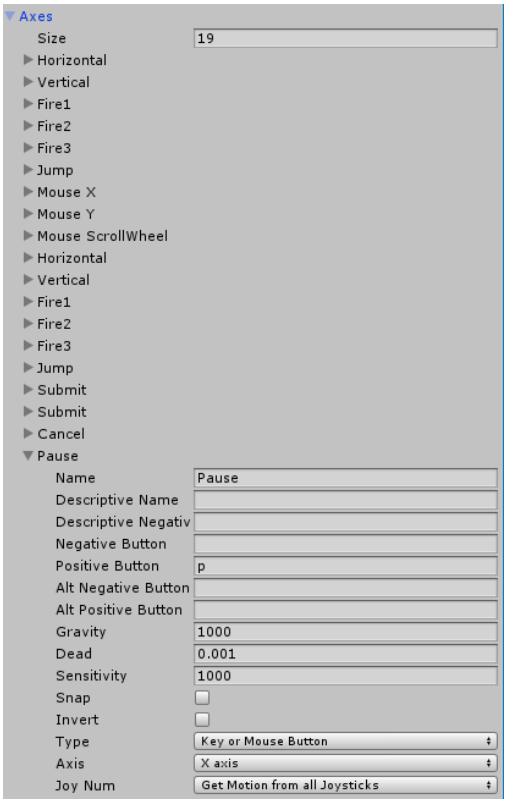

Figure 10. Inputs.

Create a new input by increasing the Size value to 19 from 18 and then rename the new input from “Cancel” to “Pause” and set its positive button to the letter “P”. Clear the “Alt Positive Button” field but leave the other settings alone.

To trigger the snapshot in-game, create a new script named PauseScript and attach it to a GameObject in the scene.

```
public class PauseScript : MonoBehaviour
{
    private bool unPaused = false;
    private FMOD.Studio.EventInstance SnapshotInstance;
```

Figure 11. The variables.

The unPaused bool is used to determine whether the game is paused or not and the FMOD.Studio.EventInstance variable will contain the instance of the snapshot “playing” which will be destroyed when it is no longer needed.

```
void Update()
{
    if (Input.GetButtonDown("Pause"))
    {
        //Switch between paused state and unpaused state
        unPaused = !unPaused;

        if (unPaused)
        {
            //Pause game
            Time.timeScale = 0;

            //Create FMOD instance and trigger pause snapshot
            SnapshotInstance = FMODUnity.RuntimeManager.CreateInstance("snapshot:/Pause");
            SnapshotInstance.start();
        }
        else
        {
            //Unpause game
            Time.timeScale = 1;

            //Disable pause snapshot and release the instance
            SnapshotInstance.stop(FMOD.Studio.STOP_MODE.ALLOWFADEOUT);
            SnapshotInstance.release();
        }
    }
}
```

Figure 12. The Update function.

First, an if-statement is used to check for the player pressing the Pause input (The “P” key) created in the previous step. If the key is pressed, the unPaused boolean is set to true if its current value is false, or false if its current value is true.

A nested if-statement checks the state of the unPaused boolean and acts accordingly. If the game is not paused, the timescale is set to 0 which effectively pauses the game and then an FMOD instance of the Pause snapshot is created and assigned to the SnapShotInstance variable. The instance is then started which triggers the snapshot muting the game’s audio.

Otherwise, the timescale is returned to the default value of 1 and the instance is stopped and destroyed since it is no longer needed until the next time the player pauses the game.

6 Profiling & Live Update

A profiling tool is included with FMOD Studio which allows for debugging and assessing the performance of an FMOD project in real-time by linking FMOD Studio to a game using a feature called Live Update. First, launch the Unity project and run the game. In FMOD Studio, find the button named “Live Update Off” at the bottom edge of the UI and click on it. A window will pop up asking for the IP address of the computer or console you are connecting to. Use the default setting of “localhost” and click on “Connect”. The button previously named “Live Update Off” should now read “Live Update On” and be coloured green showing that FMOD Studio is connected to the game. If the game stops in Unity, FMOD Studio will automatically attempt to reconnect after losing the connection and so when the game is run again, the connection will be reformed. If the Live Update feature is not needed, make sure to turn it off in FMOD Studio so that it isn’t inadvertently running.

6.1 Profiling process

Figure 13. The profiler controls.

First, open the profiler window by clicking on “Window” in the toolbar and then clicking on “Profiler” in the drop-down menu. By clicking on the record button which is the button with a round circle pictured above, the profiling process will start and all the events triggered while playing the game will be recorded for later analysis. The recorded sessions will be stored on the left-hand side of the profiler window under the “Sessions” tab.

In a recorded session, statistics can be viewed either for individual events or the master bus. These include CPU, memory and I/O usage as well as changes in volume levels and the total number of events playing. In the “Overview” tab on the right-hand side of the profiler window, the positioning of events relative to the FMOD Listener in the Unity project is shown along with a list of active snapshots. Also, by clicking on the icon that looks like this: </>, all calls to the FMOD API can be viewed frame by frame.

An important feature to point out is API Playback. By clicking on the “Simulate” button above the timecode, the profiler session will be played back as if the game was being played which allows for the sound designer to change events and then listen to the profiler session to hear the changes made without Unity or a build of the game running. To return back to the audio recorded along with the profiler session, click on the “Waveform” button which is also located above the timecode.

6.2 Using Live Update

When Live Update is active, all changes made to the FMOD Studio project will take effect instantaneously while playing the game. This allows for live iteration and mixing without having to waste time building banks and restarting the game but keep in mind that the FMOD Studio project will still need to be saved and the banks built for the changes to take effect permanently.

7 Object length issue

7.1 What issue?

If an event is attached to an object within Unity, the sound effect will only play at full volume when the player is positioned next to or on top of the centre of the object. The further away from the object the player moves (in either the x or the y-axis) the quieter the sound becomes. This makes sense for most objects but for objects with a considerable length such as a conveyor belt, for example, this poses a problem.

This section covers the solution to this problem of lengthy objects that need 3D sound but at a constant volume along a defined axis.

7.2 Preparation in Unity

For testing purposes, create an object within the scene by clicking on the “Create” button in the hierarchy view and then clicking on “Cube” under the 3D Object -subcategory in the drop-down menu. While the new Cube object is still selected in the hierarchy, find it’s transform in the inspector and change its scale in either the X-axis, Y-axis or Z-axis from 1 to 25 to create an object that is lengthy in nature.

Next, right click on the object in the hierarchy and click on “Create Empty” to create an empty child object. Name the child object StartPoint and set its position so that it is at one end of the parent object. Create another child object named EndPoint and set its position to the other end of the parent object so that a child object is now placed at each end.

Create one more child object named MovingEventEmitter and add the FMOD Studio Event Emitter component to it and in the settings for the component, select “Object Start“ under the Play Event drop-down list and followed by an event in the event list.

7.3 Scripting

Select the MovingEventEmitter child object in the hierarchy if it has been unselected. Click on “Add Component” in the inspector and scroll down to “New Script”. Create a new script with the name of MovingEventEmitterScript.

```
public class MovingEventEmitterScript : MonoBehaviour
{
    private Vector3 playerPos;
    private Vector3 vectorBetweenPoints;
    private float vectorBetweenPointsMagnitude;

    public Transform player;
    public Transform startPoint;
    public Transform endPoint;
    public float speed;
```

Figure 13. The variables.

The variables needed for this script. The public variables are assigned values within the Unity inspector as shown in the figure below.

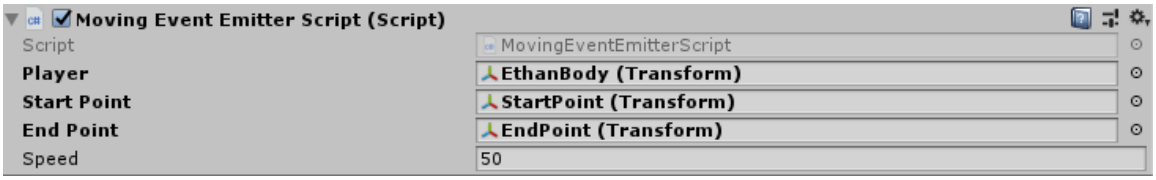

Figure 14. Variables as shown in the inspector.

The player variable is a reference to the transform of the EthanBody GameObject which can be found under ThirdPersonController in the hierarchy.

The StartPoint and EndPoint variables contain references to the transforms of the StartPoint and EndPoint GameObjects created earlier.

Speed is a float variable that will determine the speed of movement between the StartPoint and EndPoint GameObjects.

```
void Start()
{
    //Creates a new vector between startPoint and endPoint
    vectorBetweenPoints = endPoint.position - startPoint.position;

    //Stores the magnitude of the vector
    length = vectorBetweenPoints.magnitude;

    //Normalizes the vector which makes its magnitude 1
    vectorBetweenPoints.Normalize();
}
```

Figure 15. The Start function.

In the Start function, a vector is created between the startPoint and endPoint GameObjects by subtracting the position of the startPoint from the position of the endpoint. The GameObject this script is attached to will move along this vector.

The magnitude of the created vector is stored using the length variable and then the vector is normalized so that the vector only indicates direction, without any information about the magnitude.

```
void Update()
{
    //Keeps track of player's position
    playerPos = player.transform.position;

    //Move this GameObject to the nearest point on the vector to the player
    transform.position = Vector3.MoveTowards(transform.position, NearestPointToPlayer(), speed * Time.deltaTime);
}
```

Figure 16. The Update function.

In the Update function, the player's position is tracked and stored using the playerPos variable. Then the GameObject the script is attached to is moved to the nearest point on the vector to the player at the speed determined by the speed variable. This is what allows the player to hear the SFX in a realistic manner when it is attached to a long object.

```
private Vector3 NearestPointToPlayer()
{
    //Creates a new vector between the player and the startPoint
    Vector3 vectorBetweenPlayerAndStartPos = playerPos - startPoint.position;

    //Calculates the dot product between two vectors
    float dotProduct = Vector3.Dot(vectorBetweenPlayerAndStartPos, vectorBetweenPoints);

    //Clamps the dot product
    dotProduct = Mathf.Clamp(dotProduct, 0f, vectorBetweenPointsMagnitude);

    //Returns closest point to the player on the vector between startPoint and endPoint
    return startPoint.position + vectorBetweenPoints * dotProduct;
}
```

Figure 17. The NearestPointToPlayer function.

The NearestPointToPlayer function contains the calculations needed to determine the closest point on the vector to the player.

First, a new vector is created between the player's position and the position of the startPoint by subtracting the position of the startPoint from the player's position.

Then, the dot product of the vector between the startPoint and the endpoint, and the vector between the player and the startPoint is calculated.

Next, the dot product is clamped between 0 and the vectorBetweenPointsMagnitude variable so that the GameObject can only travel between the two points.

Lastly, the closest point to the player on the vector between startPoint and endpoint is returned by adding the position of the startPoint to the vector between the startPoint and the endPoint and then multiplying it by the dot product calculated above.

8 Music

8.1 Logic markers

In FMOD Studio, it is possible to add logic markers that control how the event's audio tracks playback. Adding logic markers can be especially useful for events containing music since, in games, the music is often looping or is affected by what the player does in-game. For example, a music cue that plays during combat might increase in intensity when there are multiple enemies but lessen in intensity as the enemies drop in number. Or, sometimes a music cue must loop until the player opens a door after which the music can progress into the next section. All this can be accomplished using logic markers.

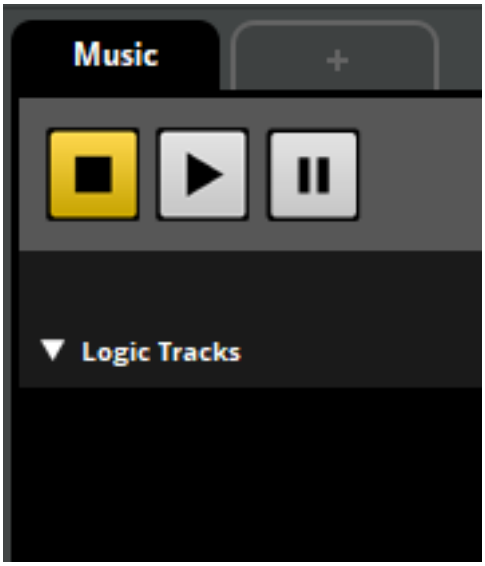

Figure 18. Music event.

Logic markers must be added to the event's logic tracks and it is possible to tell if the logic tracks for the selected event are available for use by checking the direction of the arrow icon located to the left of the words "Logic Tracks" under the playback controls. If the arrow is pointing to the right, the logic tracks will be hidden and inaccessible. If the arrow is pointing down, then by clicking on the arrow the logic tracks will appear under the timeline as shown in the above figure.

By right-clicking on the logic track, a list of options for adding logic markers will appear:

- Add Destination Marker

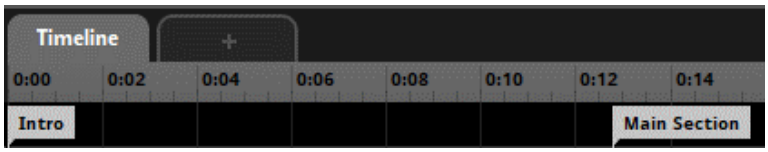

Figure 19. Destination Markers.

Destination markers are used to mark certain points in the timeline and are used in conjunction with transitions and transition regions. By double-clicking on the marker, it can be renamed.

- Add Tempo Marker

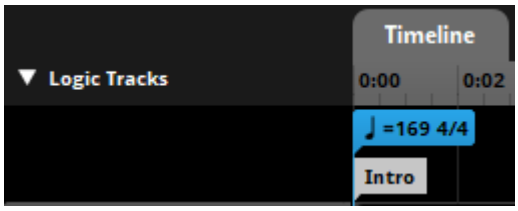

Figure 20. Tempo Marker.

A tempo marker is used to mark the tempo and time signature of the piece of music. Multiple tempo markers may also be used to mark tempo or time signature changes.

This should be the first piece of logic added to a music event as it makes it possible to change the timeline to display beats instead of time and much of the other logic requires a tempo marker to work properly. It also allows for beatmatching if needed.

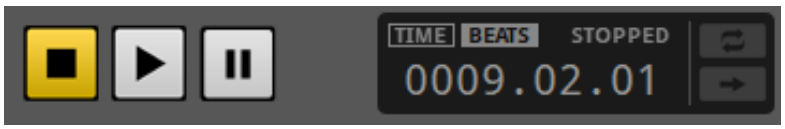

Figure 21 Time/Beats display.

Once the tempo marker has been added, click on the word “Beats” to the right of the playback controls to display beats on the timeline, which makes markers and transitions easier to place according to how the track has been composed.

- Add Loop Region

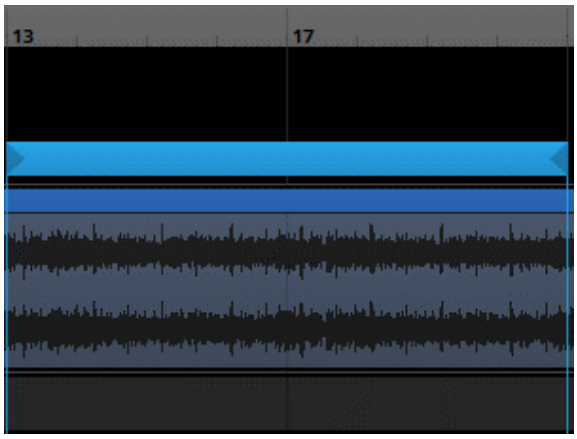

Figure 22. A Loop Region.

A loop region is used to loop a section of music indefinitely. By dragging on the outermost edges of the loop region, the start and end of the loop region can be adjusted. Click on the loop region to show the conditions governing the loop at the bottom-left of the UI.

Figure 23. Conditions.

If no trigger conditions are present, the loop will be infinite. In the figure above, the loop will only be active when the value of the parameter used for the trigger condition is between 0.75 and 1.00. Also, by clicking on the dice icon and adjusting the probability, the chance of the loop triggering at all can be lowered.

- Add Sustain Point

Figure 24. Sustain point.

A sustain point is used to pause playback temporarily until the sustain point is released through code and playback can continue. Asynchronous clips will not be affected and will keep playing.

```
//Releasing a sustain point
eventInstance.triggerCue();
```

Figure 25. The code for releasing a sustain point.

- Add Transition

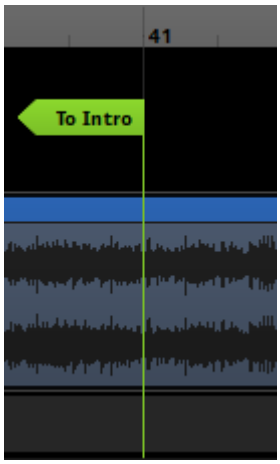

Figure 25. Transition.

Transitions are used in conjunction with destination markers to skip from the marker and restart playback from whichever destination marker is indicated. Conditions and probability may be applied in much the same way as for loop regions.

- Add Transition Region

Figure 26. Transition region.

A transition region works exactly like a transition but the transition to the destination marker can happen at any point within the region if the conditions are met.

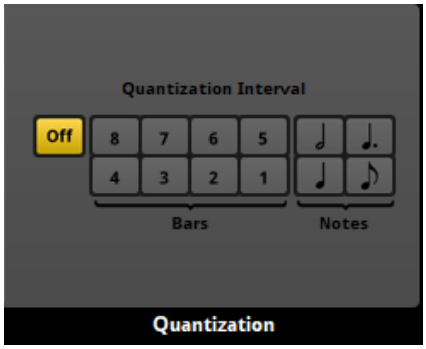

Figure 27. Quantization settings.

Next to the conditions, there are separate settings for quantization. Quantization in music terms is the process of moving data that is out of time to a spot that makes sense considering the underlying rhythm of the song. In this case, when a transition is called for, FMOD will check the playback position inside the transition region and if it is not in time it will wait according to the quantization interval settings shown in the above figure before triggering the transition.

The setting can either be bar-based or beat-based and both are calculated from the most recent tempo marker.

8.2 Adaptive music

For games, it is often desirable to have music that adapts to the game world and the player's actions. This way the music can reflect the interactive nature of video games. What follows is an example of adaptive music in the context of a boss fight that is typical of an action-adventure game.

8.2.1 Creating the FMOD event

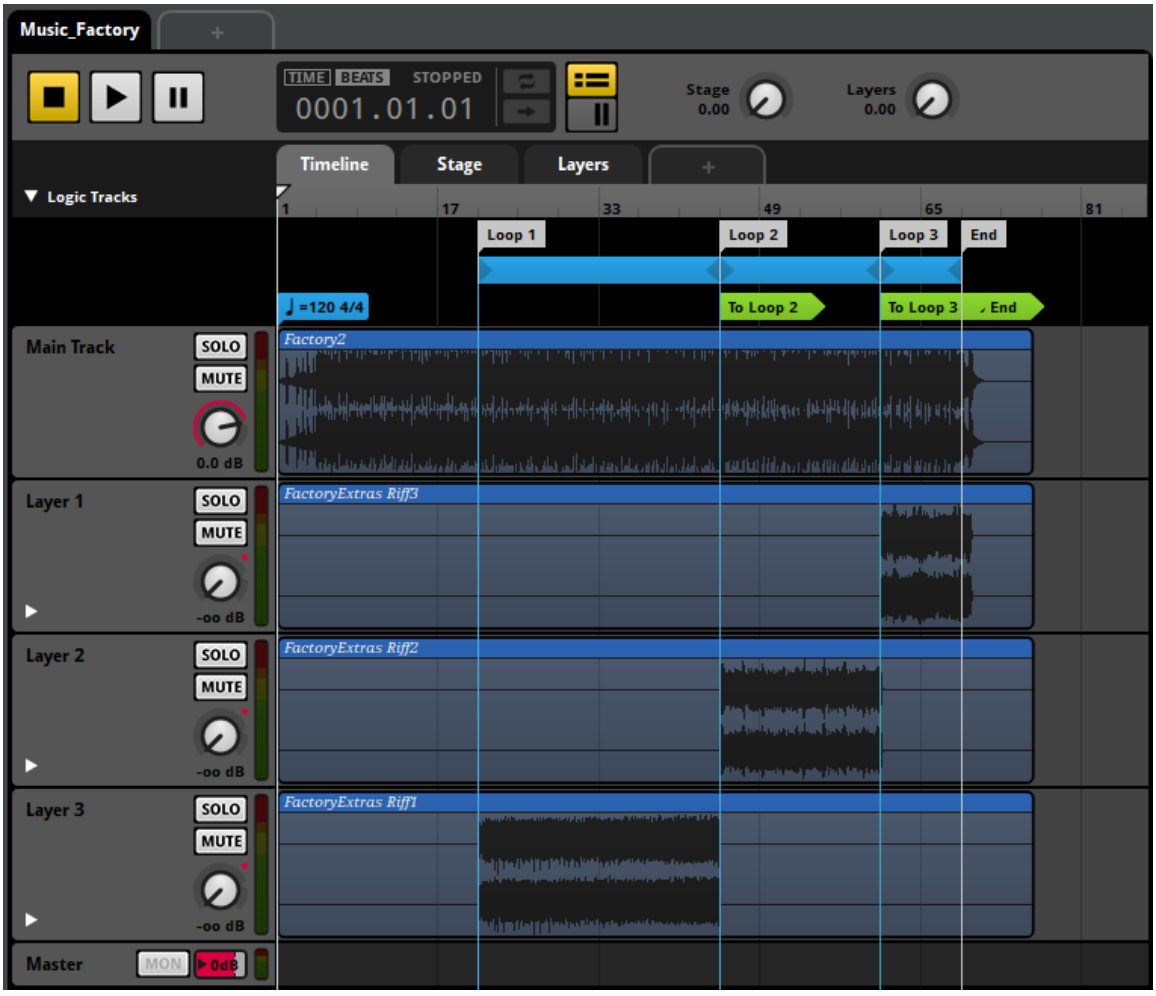

Figure 28. An adaptive music event within FMOD Studio.

First, a tempo marker is added at the start of the event and the timecode is switched to “Beats” mode. The event contains four audio tracks with the first one being the main track that serves as the base of the event.

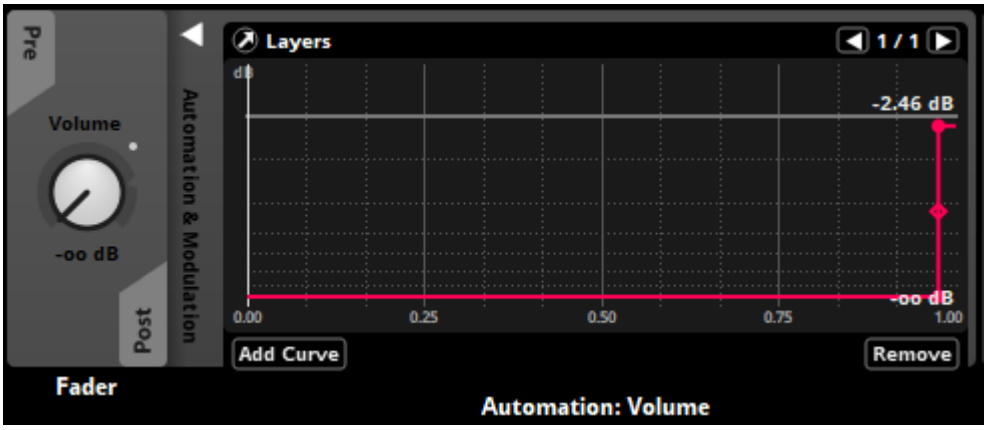

Figure 29. Volume automation for the layered tracks.

The three tracks after it are layered on top of the main track by increasing the volume from zero decibels to a pre-defined level when the parameter named “Layers” is set to a value of 1 as shown in the figure above.

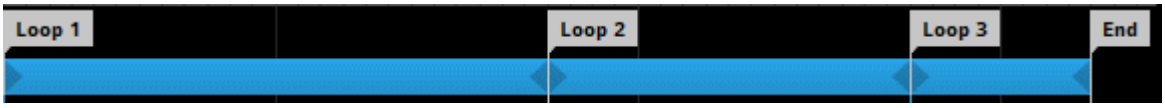

Figure 30. Destination markers and loop regions.

Destination markers are added to the logic tracks to mark the beginnings of where the looped sections of music are and so that they can be used in conjunction with transition markers. In this case, the destination markers are placed at the beginnings of bars 21, 45 and 61. Three loop regions have also been added spanning from destination marker to destination marker.

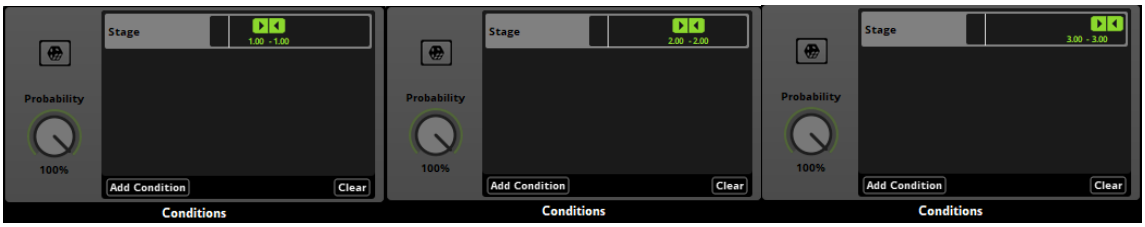

Figure 31. Transition markers.

Next, three transition markers are added to the logic tracks at bars 45, 61 and 69. The first marker is set to transition to the second loop if the “Stage” parameter is set at 1, the second marker checks for the value of 2 before transitioning to the third loop and the last marker checks for the value of 3 in which case the end of the piece of music will play out after the loop has ended.

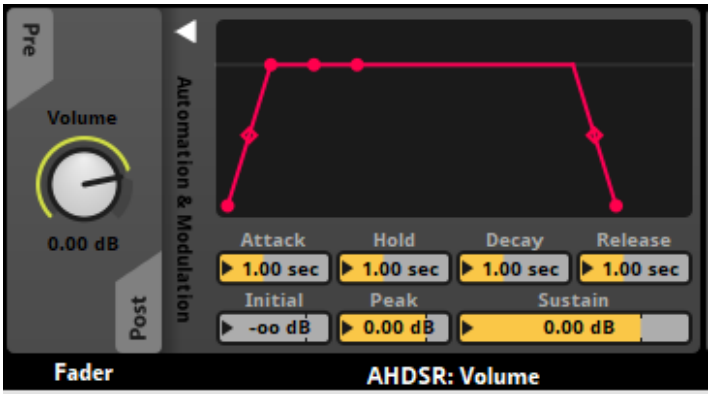

Figure 32. AHDSR envelope.

Lastly, an AHDSR envelope is added to the master volume of the event so that the audio fades in and out smoothly when the event starts or ends.

8.2.2 Scripting

```
public int bossfightStage = 0;
public int bossfightIntensity = 0;

FMOD.Studio.EventInstance eventInstance;
```

Figure 33. The variables.

For this example, two public integer variables have been created with a default value of 0. These will be used to determine which stage of the boss fight is active and the intensity of combat, which could be determined by the number of enemies around the player or by the player's health points for example. In practice, these variables would be set in another script dealing with the mechanics of the boss fight and its gameplay.

```
void Start()
{
    //Creating the instance and starting it
    eventInstance = FMODUnity.RuntimeManager.CreateInstance("event:/Music_Factory");
    eventInstance.start();
}
```

Figure 34. The Start function.

In the Start function, an instanced event is created and started. A oneshot event would not be suitable for making changes to parameters.

```
void Update()
{
    SetStageParameter();
    SetLayersParameter();
}
```

Figure 35. The Update function.

The Update function is running the functions explained below.

```
void SetStageParameter()
{
    if (bossfightStage == 1)
    {
        eventInstance.setParameterValue("Stage", 1);
    }
    else if (bossfightStage == 2)
    {
        eventInstance.setParameterValue("Stage", 2);
    }
    else if (bossfightStage == 3)
    {
        eventInstance.setParameterValue("Stage", 3);
    }
}
```

Figure 36. The SetStageParameter function.

The SetStageParameter function uses if-statements to check the stage of the boss fight currently active and sets the “Stage” parameter within the FMOD event accordingly.

```
void SetLayersParameter()
{
    if(bossfightIntensity == 1)
    {
        eventInstance.setParameterValue("Layers", 1);
    }
    else
    {
        eventInstance.setParameterValue("Layers", 0);
    }
}
```

Figure 37. The SetLayersParameter function.

The SetLayersParameter function uses an if-statement to check the intensity of combat and sets the “Layers” parameter within the FMOD event accordingly.

9 Footstep system

9.1 What is this?

This chapter covers creating a system for footstep sound effects using Unity, C# programming as well as FMOD Studio.

9.2 Preparation in Unity

Navigate to “Standard Assets – Prototyping – Prefabs” and find the prefab called PlatformPrototype08x01x08. Drag two copies of this prefab into the scene and place them near the player character at a suitable depth for the character to step onto them, a Y position of 0.12 should be sufficient. Rename the objects in the hierarchy so that they read GrassPlatform and SandPlatform.

Next, navigate to “Standard Assets – Environment – TerrainAssets – SurfaceTextures” and drag the GrassHillAlbedo texture onto the GrassPlatform object in the scene and do the same with the SandAlbedo texture and the SandPlatform object. This step is done so that the platforms can be visually differentiated when testing and is not part of the footstep system.

Click on the “Create” button under the Project tab and create two PhysicMaterials called Sand and Grass respectively. To keep things tidy, move the PhysicMaterials to the folder under Standard Assets called PhysicsMaterials.

Click on the GrassPlatform object in the hierarchy and find the Material input within the Box Collider component in the inspector. Set the material to the corresponding the PhysicMaterial. Now do the same for the SandPlatform object.

9.3 Creating the FMOD event

In FMOD Studio, open up the project and make sure the Events tab is selected in the top-left corner of FMOD Studio. Click “New Event” in the bottom-left corner of FMOD Studio and create a new 2D event named Footsteps. Assign it to the master bank by

right-clicking on the event and clicking on “Assign to Bank – Master Bank”. For this example, four different sets of SFX will be used for the footsteps so prepare some SFX to be used for this purpose. Create four audio tracks within the event named: Wood, Water, Dirt & Sand. Add a Multi Instrument to each track and then drag in some SFX for each footstep type. Add a bit of volume and pitch modulation to each multi instrument for more variation in sound.

Create a parameter named Surface with the default settings. Add volume automation for each track and use the Surface parameter to make sure that at points 1 and 2, only one track has the volume turned up at a time.

9.4 Scripting

Select the ThirdPersonController object in the hierarchy. Click on “Add Component” in the inspector and scroll down to “New Script”. Create a new script with the name of FootstepScript.

```
[FMODUnity.EventRef]
public string eventPath;

public GameObject leftFoot;
public GameObject rightFoot;
```

Figure 28. The variables.

These are the global variables needed for this script. They are public variables so that it’s possible to assign the variables using the inspector window in Unity as shown in the figure below.

Figure 29. Variables as shown in the inspector.

The eventPath string is the path to the FMOD event and it needs the [FMODUnity.EventRef] namespace to function. The leftFoot and rightFoot GameObject variables

are references to the EthanLeftFoot and EthanRightFoot GameObjects that are child objects of the ThirdPersonController object. These will be needed for the PlaySFX function.

Click on the magnifying glass icon next to the Event Path variable in the inspector and assign the Footsteps event created earlier.

Now find the EthanLeftFoot and EthanRightFoot GameObjects using the Project window and drag them into the Left Foot and Right Foot variable slots in the inspector also.

```
int CheckSurface(RaycastHit hit)
{
    /*
     * 0 - NULL
     * 1 - Sand
     * 2 - Grass
    */

    string materialName = hit.collider.material.name;
    string fixedMaterialName = materialName.Remove(materialName.Length - 11, 11);

    if (fixedMaterialName == "Sand")
    {
        return 1;
    }
    else if (fixedMaterialName == "Grass")
    {
        return 2;
    }
    else
    {
        return 0;
    }
}
```

Figure 30. The CheckSurface function.

In this function, a RaycastHit variable is used to check the name of the PhysicMaterial that was assigned to the box colliders of the two surfaces earlier. The name is stored using the materialName string but Unity adds some extra characters to the string so the fixedMaterialName string is the shortened version of materialName leaving only the actual name of the material intact. Next, the name is checked and a different integer is returned based on what the material is. If no material is found, an integer of 0 is returned.

```
void SetParameter(FMOD.Studio.EventInstance e, string name, float value)
{
    FMOD.Studio.ParameterInstance parameter;

    e.getParameter(name, out parameter);
    if (!parameter.isValid())
    {
        Debug.Log("The parameter named: " + name + " does not exist");
        return;
    }
    parameter.setValue(value);
}
```

Figure 31. The SetParameter function.

This is what sets the parameter in the Footsteps event. An FMOD EventInstance, as well as a string and a float, will be passed as arguments when this function is called. First, an FMOD.Studio.ParameterInstance variable is created to store the parameter so that the value of it can be set later on. Then the parameter is retrieved using the reference to the EventInstance and the name of the parameter, and stored in the aforementioned variable.

Next, the parameter is checked for validity and if the check returns a value of false, a debug message is printed stating so and a return statement is used to stop executing the function. If the parameter is valid, the last line sets the value of the parameter to the float variable that was passed as an argument.

```
public void PlaySFX()
{
    RaycastHit hit;

    if (Physics.Raycast(leftFoot.gameObject.transform.position, Vector3.down, out hit, 10.0f))
    {
        if(hit.collider.material.name != "")
        {
            int materialIndex = CheckSurface(hit);

            if (eventPath != null && materialIndex != 0)
            {
                FMOD.Studio.EventInstance e = FMODUnity.RuntimeManager.CreateInstance(eventPath);

                SetParameter(e, "Surface", materialIndex);

                e.start();
                e.release();
            }
        }
    }
    else if (Physics.Raycast(rightFoot.gameObject.transform.position, Vector3.down, out hit, 10.0f))
    {
        if (hit.collider.material.name != "")
        {
            int materialIndex = CheckSurface(hit);

            if (eventPath != null && materialIndex != 0)
            {
                FMOD.Studio.EventInstance e = FMODUnity.RuntimeManager.CreateInstance(eventPath);

                SetParameter(e, "Surface", materialIndex);

                e.start();
                e.release();
            }
        }
    }
}
```

Figure 32. The PlaySFX function.

This is the main function of the script that will be called when a movement animation reaches our desired frame. First, a RayCastHit variable is created to store the results of our raycasts. Within the if-statement, as well as the else if-statement, a raycast is sent down towards the ground with the origin being the leftFoot and rightFoot variables defined in the inspector. If the raycast hits a collider, the if-statement is found to be true. Within the statements, a check is made to see if the raycast hit a surface that has a named material and if so, the CheckSurface function is used to check the surface type of the collider that was hit and the result is stored in the materialIndex integer. Afterwards, another if-statement is used where the eventPath and the materialIndex variables are checked, and if they are true, an EventInstance is created using the path to the desired FMOD event. After that, the SetParameter function is used to select the correct audio track within the event and then the event is played and immediately released from memory afterwards.

9.5 Character setup

Because the character has been imported, the animations that need editing will be set as read-only. To solve this, navigate to “Standard Assets – Characters – ThirdPersonCharacter”. Under this folder are the Animation and Animator folders.

First, open up the Animation folder and click on the arrow to the left of HumanoidRun. This will show the subobjects where the animation which is also named HumanoidRun can be found. Duplicate the HumanoidRun animation by selecting it and then pressing Ctrl+D. Then in the Animator directory, double-click on the AnimatorController called ThirdPersonAnimatorController. The Animator tab should pop open. Double-click on the state called “Grounded” and then click on the “Blend Tree” object. In the inspector, all the animations used by the character should be available to be seen. Replace the HumanoidRun animation by dragging the duplicated HumanoidRun animation in its place.

Make sure the ThirdPersonController object is selected in the hierarchy and then open up the Animation view by clicking on “Window – Animation”. Open up the list of animations to see that the HumanoidRun animation is no longer read-only. Select it from the list and click on the Preview button. Make sure the character can be seen in either the Scene or Game window and then use the timeline in the Animation window to find the animation frames where the characters feet hit the ground. Right-click under the timeline and click “Add Animation Event” at these spots in the timeline. Click on each animation event and select the PlaySFX function created earlier using the inspector. Repeat this procedure for all animations that include footsteps such as HumanoidWalk.

10 Conclusions

FMOD Studio allows a sound designer or a composer to more easily create complex audio events to be used in a project and it also saves the programmers from having to spend time coding audio systems that are provided directly through FMOD Studio and other similar middleware such as Audiokinetic's WWise. Therefore, the workload of the programmers will be drastically reduced allowing them to work on gameplay features for example. Producers will also benefit since fewer cuts to features will have to be made due to the amount of time saved by using FMOD Studio.

Two topics that were left out of this thesis, due to limiting the scope of the thesis, were beatmatching and dialogue systems. Beatmatching is a way of retrieving which beat in an event is playing and then using that information to trigger something using code. Dialogue systems are complex to create and include using line codes in conjunction with a programmer instrument and a feature of FMOD Studio call an audio table.

Hopefully, this guide will be of use to anyone in the game development community needing to learn how to create some of the more commonly needed audio systems for games and it might also help them avoid some of the pitfalls and problems that can be encountered while working with FMOD and FMOD Studio.

List of references

Bigworldtech.com. n.d. BigWorld Technology – BigWorld Partners. Accessed 03.12.2018. https://web.archive.org/web/20100918044026/http://bigworldtech.com/company/partners.php

Connect.creativelabs.com. n.d. Creative Labs: Connect. Accessed 30.11.2018. https://web.archive.org/web/20100504065155/http://connect.creativelabs.com/developer/Gaming/Forms/AllItems.aspx

Crytek.com. n.d. Crytek GmbH: Specifications. Accessed 03.12.2018. https://web.archive.org/web/20091217135020/http://www.crytek.com/technology/cryengine-2/specifications

Firelight Technologies. n.d. FMOD Games. Accessed 03.12.2018. https://www.fmod.com/games

Firelight Technologies. n.d. FMOD Licensing. Accessed 12.11.2018. https://www.fmod.com/licensing

havok.com. n.d. Products – Vision Game Engine | 3rd Party Integrations. Accessed 03.12.2018. https://www.webcitation.org/671o6F5xS?url=http://www.havok.com/products/vision-engine

HeroEngine wiki. n.d. HeroEngine 1.47.0 Enhancements. Accessed 03.12.2018. https://web.archive.org/web/20110411083558/http://hewiki.heroengine.com/wiki/HeroEngine_1_Release_40#1.47.0_Enhancements

scaleform. n.d. Integration. Accessed 03.12.2018. https://web.archive.org/web/20100420052611/http://www.scaleform.com/products/integration

Unreal Technology. n.d. Unreal Technology. Accessed 30.11.2018. https://web.archive.org/web/20100529013331/http://www.unrealtechnology.com/partner-program.php

Valve Corporation. n.d. Implementing FMOD. Accessed 03.12.2018. https://developer.valvesoftware.com/wiki/Implementing_FMOD

Wikipedia. n.d. FMOD Studio. Accessed 12.11.2018. https://en.wikipedia.org/wiki/FMOD