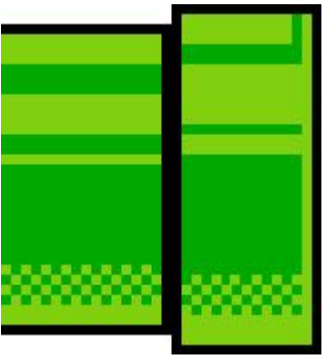


Game Audio Pipelines

A research of how we organize ourselves

Bachelor's project in Media Sonic Communication,

Sonic College - UCSYD Haderslev

19. december 2019

by Jeppe Emil Lindskov & Mads Vesterager Riddersholm

Contact: jeppelind@icloud.com & contact@madsriddersholm.com

Project Supervisor: Niels Bøttcher

Characters: 84.000

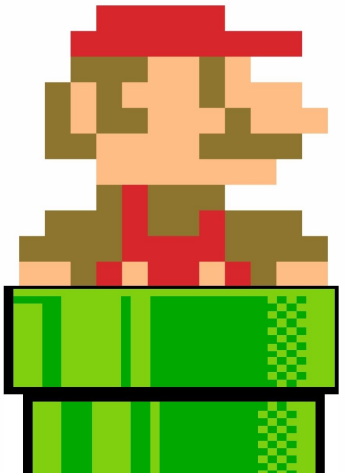


Table of Contents

1.0 Introduction…… p. 2
1.1 Motivation & Problem Statement…….. p. 2
1.2 Methodology……..p. 3
2.0 Game Audio Pipelines…….. p. 5
2.1 Definition…… p. 5
2.2 Historical Context…… p. 7
2.3 Pre-Production……..p. 10
3.0 Tools & Software…… p. 12
3.1 DAW…….. p. 12
3.2 Audio Middleware……p. 15
3.3 Implementation…… p. 19
3.4 Tools……...p. 21
4.0 Organization…… p. 25
4.1 Transparency…… p. 25
4.2 Naming Conventions & Categorization……p. 26
4.3 Generalist vs. Specialist……..p. 29
4.4 Collaboration……p. 31
5.0 Key Takeaways……..……p. 33
6.0 Pipeline Development……...p. 36
6.1 Organizing…… p. 36
6.2 Choice of DAW…… p. 37
6.2.1 Reaper Export/Import Pipeline……p. 38
6.3 Choice of Middleware…….. p. 40
6.4 Tools and Implementation…….. p. 41
6.4.1 WwiseType System…….. p. 42
6.4.2 Custom Third-Person Listener……...p. 44
6.4.3 Wwise Material Switch…… p. 45
6.4.4 Wwise Ambience Follower…….. p. 46
6.4.5 Wwise Sound Painter……...p. 46
6.4.6 Wwise Animation…….. p. 48
7. Closing Thoughts……p. 49
8. Conclusion…… p. 51
Abstract……...p. 53
Acknowledgements……p. 53
Bibliography……...p. 54
Product Appendix……p. 55
Appendix…… p. 57

1.0 Introduction

1.1 Motivation & Problem Statement

A common saying in the audio industry is that game audio is 30% sound design and 70% implementation and communication. But why do we as sound designers spend so much time on implementation and communication, instead of designing sounds and what can be done to reduce the time spend on implementation and communication and make more room for working creatively.

The motivation behind this thesis is that, while a lot of available information can be found on how to make sound design for video games, not much can be found on the subject of creating efficient workflow and pipelines. If the communicative and technical aspect of working in game audio makes up 70% of the time spend, then it should be at least as important, as the time spend on making sound design. But it's rarely documented and often something you will have to get taught, when you start working at a game developer. Our own experience is that, if you are not thinking about your workflow and pipeline, it will cause problems during production, even if you work as the sole sound designer at an indie company.

This study aims to provide the reader with an understanding of *How to create a game audio pipeline, that streamlines and optimizes workflow and game audio implementation?* To do this we will be addressing the following subquestions:

- What considerations are made in the development of a game audio pipeline?
- What importance does it have for the development of a video game, when audio is a part of the pre-production.
- What does AAA companies have in common in their game audio pipelines and what are they doing differently?
- Is it possible to develop reusable software for general workflow improvement in game audio?

1.2 Methodology

The thesis will be split into two different sections.

The first section will be focused on the different stages and practices of a pipeline. It will start out with an description of what a game audio pipeline is, followed by a presentation to the history and evolution of game audio. Then we will look at the different key points in a game audio pipeline, including the practice of organization. The purpose is to lay the foundation and outline the various principles and elements that goes into creating a game audio pipeline. At the end of the section we will summarize the key takeaways.
The data provided in this section will be based upon literature and qualitative interviews with mainly audio directors from AAA studios. The reason for us to make use of audio directors is, that we want our data to be backed up by experience and knowledge on the subject. Most of these people have 10-20 years of experience developing pipelines and working within the video game industry, making them experts in the creation of game audio pipelines.

In the second section we will use the knowledge and conclusions from the precious section to exemplify how we would plan a game audio pipeline. We will be using the Unity template project *The Explorer: 3D Game Kit*[1] as a prototype case. Additionally we will be using Reaper, Wwise and Unity as our DAW, middleware and game engine solutions. All audio assets in the finished product will be produced by us.
The case will be treated as a prototype in pre-production and our approach will be according to this, the challenges the project might give us and how we will manage our pipeline to accommodate it.
We will produce and exemplify various pipeline tools, for export/import improvement and implementation. The examination and descriptions of these tools will be focused on why we created them and how they work, with remarks on how these tools fit into the decision making of planning our pipeline. Video demonstrations of each tools will be provided. The tools will mainly be written and authored by ourselves, but some

[1] https://learn.unity.com/project/3d-game-kit (17-12-2019)

will be built upon pre-existent content. If a specific part of a tool isn't made by us, it will be mentioned clearly. Credit is given where it is due.
This section will end with a reflection of our closing thoughts on the subject.

Game audio pipelines is a big and broad determined subject. To cover all aspects isn't possible within the boundaries of this thesis. We have made a number of demarcations to keep the thesis as focused as possible.

In this project we will not go into details with pipelines of other departments and the collaboration with other departments. We will not cover design documents, production management, documentation and source control. Note, that these are all very important subjects to discuss when talking about game audio pipelines.

We will not be composing music for our case study, because it isn't in the focus of the subject.

We will not make use of surveys or user test. The information in this thesis will only be based upon the expertise of our interviewees, literature and our own experience and knowledge.

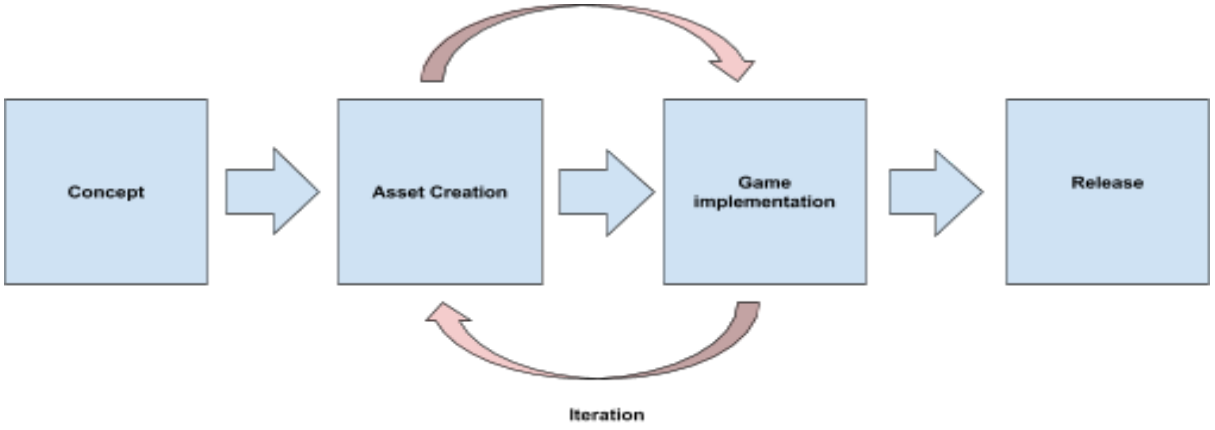
2.0 Game Audio Pipelines

2.1 Definition

In order to give a better understanding of a software development pipeline we first need to understand the term itself. In its most basic form a pipeline is basically a chain of processes that are arranged, so that the output of each element is fed straight into the input of the next. Much alike a real physical pipeline transportation.[2]

Especially in a cross disciplinary field like game development, pipelines are used extensively. Almost every discipline in game development have some sort of pipeline in order to get content into the game.

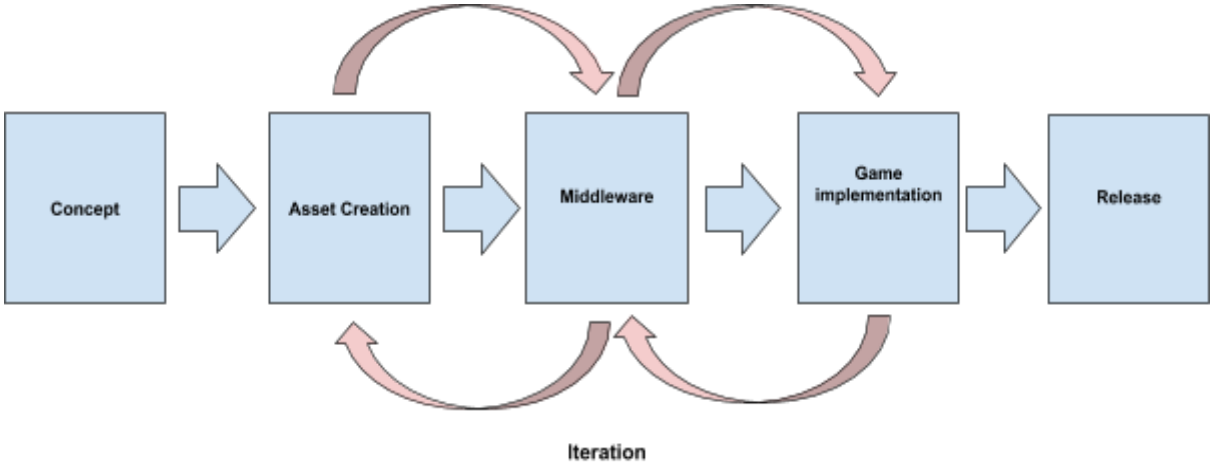
So what does a game audio pipeline look like? The most basic form of game audio pipeline starts with a concept and stops when that concept can be heard by the player in game. It could look like this:



In the concept phase you might work on a broader level of design, establish a tone and mood. From there you go into your preferred Digital Audio Workstation (DAW) and create audio assets. The assets are then exported to a library and imported into the game engine where they are integrated into the game.

[2] https://en.wikipedia.org/wiki/Pipeline_(software) (09-12-2019)

Another very common pipeline includes one additional step, which is the audio middleware. The middleware is a tool that sits in between the asset creation and the game engine implementation.



The audio middleware is a type of software tool that is integrated into the game engine and is giving the sound designer common audio functionalities out of the box. The concept itself will be explored in a chapter later in this thesis.

From the outside these models seem very simple, but in truth there is a lot more to it. When you compare this kind of pipeline to the movie production pipeline, one major difference is apparent. In movies, the design process ends when the audio has been exported from the DAW. Meaning that the whole implementation part is taken out of the equation.

When designing game audio, you might design assets to gameplay footage, but you will have to integrate those assets into the game. Not only to hear them in the right context, but also to make sure they are played back the way they were intended. This means that instead of having a creative iteration process that only happens in the DAW, you have an iteration process that happens over several steps in the pipeline, as shown in the pictures above. A more detailed example of a game audio pipeline can found in Appendix 7 p. 58.

This added layer of abstraction from the creative aspect of working straight in your DAW makes it very important, that each step between the various pipeline elements are as frictionless as possible. The fewer questions that needs to be answered the better the pipeline.[3]

> "*For me, it is important that my team is able to spend their time working on the sound and iterating on that sound, so it is fundamental that the tools and asset creation / execution pipeline is as frictionless as possible. (...)If my team spend all their time fighting with the tools to get their initial sound to play, this subtracts greatly from the amount of enthusiasm, time and creativity that remains for work on the sound itself.*"[4]

So in order to provide your sound designer with the best creative environment, you have to spend time on how you manage your pipeline. From asset creation, to implementation, to iteration, to finished product. The way this is done, is wildly different from studio to studio. In the following chapters we will go more into these different steps and describe how different studios does it and how we can use this knowledge to develop our own pipeline.

2.2 Historical Context

The development of games is in constant change. Hardware is getting better, bigger and faster and the software is following this development. The possibilities for the game developers is expanding and demands from the audience too. Seen from the perspective of a sound designer, this means that we need to be able to adapt, learn and follow the trend of new tools, new techniques and at the same time manage the projects we are working on. In other words, pipelines and the role as sound designer is in constant change.

[3] Appendix 3, p. 57, Interview_DICE_01.mp3 (2:00)
[4] Appendix 2, p. 57, Interview with Rob Bridgett p. 1

When we asked Lydia Andrew, the game audio director at Ubisoft Quebec City Studio, about the difference in game audio, from when she started working in the industry 18 years ago compared to today, she described it like this.

> *"The pipelines have changed completely, since I joined games. When I first joined games you couldn't really integrate your audio, a programmer had to do it. Even if you could integrate your audio, you couldn't change any parameters, a programmer had to do it. And the parameters you could change were very limited, so maybe pitch, maybe volume. Was it a loop, was it a one-shot. That was pretty much it. So the tools and pipelines have changed massively in terms of what you can actually do."*[5]

In the early days of game development, most game developers would build their own sound engines, inside of their own game engines. Building your own tools was an expensive and time consuming job resulting in fluctuating quality, depending on if the programmer/s responsible for creating these tools had experience working with audio and DSP[6]. Some companies would have dedicated audio programmers, who's job would be to create audio engines and other audio tools. Everything from mixers, filters, randomization and other DSP would have to be written from scratch. Having to write it all from scratch meant, that time was taken away from being creative and that every company would have unique game audio pipelines tailored for the tools and audio engines they created.[7]

The sound designers job would be to create the assets and deliver them to a programmer, together with a description of their behaviour. The audio programmer would then implement them. The systems and tools wasn't necessarily capable of real-time processing and the memory allocated for sound was limited. Effects like reverb would be baked into the assets and typically there wouldn't be room for much variations.[8]

[5] Appendix 4, p. 57, Interview_Lydia_15.mp3 (00:50 - 01.28)
[6] *Digital Signal Processing*
[7] Appendix 5, p. 57, Interview_Guy_Somberg_15.mp3 (01:47- 02:30)
[8] Somberg, Guy, 2017, p. 165

As the processing and hardware got better, so did the need for better tools.

> *"(...) a need to speed up production time and save valuable programming resources, while simultaneously increasing immersion and quality. So, to cater to both demands, a game-specialized audio toolset, production pipeline, and workflow needed to be developed."*[9]

And the audio middleware was born.

First and foremost, audio middleware has changed the role of the sound designer. Suddenly the sound designer is capable of controlling everything from recording to implementation. The tools that the middleware is offering, are available from day one and not something that needs to be developed in the course of production. Having this control is giving the sound designer the freedom and creativity to create with a different set of limitations, push these limitations and the boundaries of what game audio is, even further.
When it comes to the audio programmer, the power and tools that the middleware is offering means, that the audio programmers no longer have to build any of the fundamental functions and can focus their time on creating the features that will make the game sound unique and solve hard problems that they didn't have time to do before. The easy integration of most middleware even mean that you don't necessarily need a dedicated audio programmer.[10]

[9] Somberg, Guy, 2017, p. 165
[10] Appendix 5, p. 57, Interview_Guy_Somberg_16.mp3

2.3 Pre-Production

The production of a video game starts with pre-production. This is the initial phase, where the team will come together and scope the project. It is also in this phase that the production pipeline should be planned out.
Typically a video game production will consist of three major stages:[11]

- Pre-production. The planning and concept phase. This is done to prove out pipelines and also team dynamics and metrics. The pre-production ends with a demo or prototype of the game.
- Production. In this phase the learnings from the pre-production is used to build the game and make it playable. The production phase ends with post-production
- Post-Production. This phase consist of several stages. Pre-alpha stage is when all features are completed. Alpha stage is when the game is playable, but not polished. Beta stage is the end of the polish phase. Gold stage is the launch of the game.

In most media productions, sound is usually thought of as a post-production process. There is a tendency at some game developers to do this and at the same time start defining the audio in pre-production, without including a sound designer. This is doomed to create problems down the road. It goes with other departments as well. Every person on the pre-production are there to understand and reinforce the vision and to talk about and share their experiences, for the sake of the game.

> *"They should have experiences of what goes wrong in pre-production, where you waste your time, what you're going down, how you brainstorm ideas, how you work out if ideas are good enough".*[12]

That being said, when having sound designers join the pre-production, they aren't only there to talk audio. Being an audio specialist, working with video games makes

[11] Appendix 2, p. 57, Interview with Rob Bridgett, p.4
[12] Appendix 3, p. 57, Interview_DICE_13.mp3 (02:30 - 2:40)

you an expert in making games. Not only are you developing games, but you are passionate about games. *"You opinion about anything in the feature is valid"*.[13]

Creating a pipeline is a planning process and deciding on every aspect as early as possible is going to save time and make the process of creating a video game more transparent for everybody involved.

> *"(...)having the audio pipeline thought through, laid out, and proven at the beginning of a project provides higher planning stability and a more reliable schedule without last-minute surprises. It also gives audio design and audio technology the time to build a trustworthy collaboration(…)"*[14]

In the following sections we will look at the different steps in this process and therefore also the decision making of making a pipeline.

[13] Appendix 4, p. 57, Interview_Lydia_13.mp3 (5.45 - 06:15)
[14] Somberg, Guy, 2017, p. 166

3.0 Tools & Software

3.1 DAW

One thing that hasn't changed much over the years, is the use of DAW for creating assets. Although, when the subject comes to DAWs, sound designers have a tendency to become religious. The amount of DAWs and plugins available is extensive. They all have the same functionalities while offering unique or different ways of working.

Throughout our interviews with various audio directors and leads, when the subject landed on DAWs, there seemed to be two different company approaches. A structured approach and a free approach

Structured approach

The companies using the structured approach, have a policy about having every sound designer working in the same DAW, often with the same suite of plugins. The reason for doing this, is to create transparency. Having the same setup on every workstation means that as a sound designer you are not limited to only work on one workstation. It also means that people can share sessions and take over the work of other sound designers, without having problems opening them because of missing plugins, samples, etc. Some companies work in very modular ways, where sharing sessions and accessing old projects is part of the everyday work. Not having streamlined this process, would result in a lot of problems.

Another reason is economy and maintenance. The one responsible for the licenses, has less to manage, when everybody is working in the same software, and you don't have to tailor a new workstation every time a new sound designer joins the company.

Free approach

The idea with the free approach is to let sound designers work in the tools that they are most comfortable working in. The point is to help creativity, by allowing people to

work with the tools they have their muscle memory in. By forcing people to work in a specific DAW with specific plugins, you run the risk of losing those edge skills that people have.[15]

Ben Minto from DICE explains another reason for them to be using the free approach. At DICE they don't really do finished sound design inside of their DAW. Differently from most other companies, they are not using audio middleware, but they are using the audio engine inside of Frostbite[16]. The way that the audio engine works is very different from the pipeline of most popular middleware. Most of their sound design happens inside of the engine. Their point of reference is the engine and not the exported assets. Where the assets come from then doesn't matter as much.[17]

Mixing the approaches

The cons of these two disciplines usually comes down to the advantage of the other. This doesn't mean that you can't make exceptions and create hybrids of the approaches.

Ubisoft Quebec City Studio is an example of this. They used to have everybody work in Nuendo, but gradually this have changed into a mix between Nuendo and Reaper. Reaper is integrated into their voice pipeline, because of the scripting and modification possibilities that Reaper offers. Slowly Reaper has spread to some of the sound designers, who like to work in Reaper because of these possibilities. But they are still practicing a structured approach. In the case of plugins and libraries. If sound designers want individual plugins or libraries, it needs to be very specific for their work and it needs to be valuated and tested, before they buy copies for all workstations.[18]

[15] Appendix 3, p. 57, Interview_DICE_02.mp3 (01:50 - 02:30)
[16] https://www.ea.com/frostbite/engine (17-12-2019)
[17] Appendix 3, p. 57, Interview_DICE_05.mp3
[18] Appendix 4, p. 57, Interview_Lydia_04.mp3 (01:56 - 02:45)

Another example is at IO Interactive. The audio lead Jonas Breum believes in giving the individual sound designer freedom in the way they choose to work.

> *"Because we want people to work fluently and work expressively, more than efficiently. So it's import that people can go to their weapon of choice and bring their own kind of working. That's the kind of balance that we are trying to hit."*[19]

But he also sees the advantage on having everybody working in the same DAW. Which is why they are now transitioning towards a more structured approach. Jonas is seeing the benefits from working with Reaper, because of the built-in functions and possibilities it creates, in terms of scripting and pipeline improvements, to be bigger than the benefits of working in individual DAWs. At the same time, with the amount of new people starting in the office, in many cases Reaper will already be their primary DAW. So it makes sense to kind of follow the new tendencies with the people who are going to do the bulk of the hands on work.[20]

Reaper and Nuendo

Through our interviews with various audio leads and directors. Reaper and Nuendo seemed to be the most popular DAWs. The reason for this popularity is foremost their in-built export options. When working in game audio a lot of time goes into exporting and importing assets. This time could be considered a waste, especially if it involves a lot of manual work. Most popular DAWs doesn't support the option of batch processing and having to export many files then becomes a tiresome and time-consuming task. Both Nuendo and Reaper offers the sound designer multiple ways of batch processing. Furthermore, both Wwise and FMOD offers additional tools, that can communicate with Nuendo and Reaper to reduce this even further.

In the case of Reaper there is another reason for its popularity. Reapers extensive customization opportunities makes it an ideal DAW for pipeline and workflow improvements. It is possible to customize key bindings, interface layout and build

[19] Appendix 6, p. 57, Interview_Jonas_Breum_Jensen_02.mp3 (01:15 - 01:55)
[20] Appendix 6, p. 57, Interview_Jonas_Breum_Jensen_04.mp3

custom scripts, which makes it possible to integrate Reaper into the pipeline of other software. Reaper supports all plugin formats and furthermore the cost of licensing is low compared to most popular DAW's.

But the development of DAW's and other audio software is in constant change. New developers, new versions and the demand for new functionalities from sound designers is making the market in a constant change. Right now Reaper and Nuendo seems to be popular, in a couple of years this might change and game developers might consider adapting.

3.2 Audio Middleware

An audio middleware is a third party software that is connected to a game engine and provides common audio functionalities. As the name specifies it sits in between something. In this case it is building bridge between the audio assets and the game engine. The sound designer will typically be working inside the GUI[21] of the middleware. The middleware is giving the sound designer a big assortment of baseline functionalities, tools and other options for the playback and behaviour of audio assets. Inside the engine, the programmer can access the API[22] of the middleware, making it possible to integrate and specify the behaviour of audio playback.

An advantage with the transition to audio middleware, that wasn't mentioned in the Historical Context chapter is the cross-platform flexibilities. Since the middleware isn't hard coded into the engine, it makes it easy to reuse and adapt systems and code, into new projects, engines and platforms. The cross-platform possibilities together with audio middleware being used in the vast majority of studios,[23] is making sound designers working with middleware solutions very versatile within all companies using the same middleware. Companies might have different ways of

[21] *Graphical User Interface*
[22] *Application Programming Interface*
[23] Appendix 5, p. 57, Interview_Guy_Somberg_04.mp3 (00:50 - 01:00)

structuring their middleware project. But the fundamentals will be the same. In this way, audio middleware provides a shared language between sound designers. Creating flexibility in the industry and encourage the sharing of knowledge.

Wwise & FMOD

The two most popular audio middleware are *Audiokinetics* Wwise and *Firelight Technologies* FMOD. They have in common, that they both offer a lot of the same fundamental tools and functionalities, but they also differ greatly in terms of interface, API, pipeline and execution.

FMOD

FMOD is one of the older audio middleware. It has been re-written and re-designed a couple of times. The current version is FMOD Studio 2.0. FMOD is an easy accessible middleware for people transitioning into game audio. The reason for this is the layout of the GUI. The layout is visually resembling the layout of a typical DAW. The event system looks similar to the typical arrangement view, with tracks, timeline and your effects placed in a tap below the arrangement. The same goes for the mixer. Furthermore there is a heavy use on buttons, knobs and tabs, which makes it user-friendly for sound designers.
FMOD used to be the biggest player back at FMOD Designer. It didn't have the design described above and in general sound designers started to get tired of the old design. During this time Wwise appeared and really talked the language of the sound designer.[24]

Wwise

Wwise is currently the most popular of the two middleware solutions. The GUI layout isn't as user-friendly as the one in FMOD, which makes it a little harder to learn. But the possibilities and build in functions exceeds those of FMOD, making it a powerful tool in the hands of the sound designer.
Wwise comes with the Wwise Launcher, which is a great app for managing projects. In the launcher the sound designer is able to upgrade to new versions and make

[24] Appendix 5, p. 57, Interview_Guy_Somberg_04.mp3 (03:10 - 04:05) (06:10 - 06:35)

project integrations into Unity and Unreal Engine, more or less with the click of a button. This is very friction free and easy to use.

The event system in Wwise is different from FMOD. The event system of FMOD can be compared to the Mixer-Hierarchy of Wwise. This is where you set up your assets, containers and build your systems. Events in Wwise, on the other hand, is used to run specific actions, like play/stop, set states, RTPC, switches, change DSP values, etc. By customizing the actions of events, is empowering the sound designer. Because of this event behaviour, in many cases the programmer can get away with only having to post events in the engine, without having to specify the behaviour in the engine.

One big difference between Wwise and FMOD is the different degrees of control from the sound designer and the programmer.
The tools and functions available out of the box in Wwise exceeds the ones in FMOD. This combined with the behaviour of events, mentioned above, makes Wwise way more appealing to a lot of companies, because the need of an audio programmer isn't as vital.

The API of FMOD is superior, compared to the API of Wwise. The API of FMOD is split into a high level API and a low level API. With the high level build on top of the low level. The high level allows the programmer to play events, set RTPCs and so forth, similar to the API of Wwise. Having access to the low level, allows the programmer to manipulate the DSP-graph directly. This makes it possible to build very unique systems and tools inside the engine. This control is empowering the programmer.[25] The API of Wwise on the other hand doesn't allow programmers the same degree of control and changing something is more likely to start a chain reaction of problems, instead of offering the tools of creation. In that way you can say that in Wwise the sound designers is the one in control and in FMOD it is the programmer that is in control.

[25] Appendix 5, p. 57, Interview_Guy_Somberg_04.mp3 (04:00 - 04:50)

Alternative Middleware Solutions

Although Wwise and FMOD is the industry standards, a lot of other competitors exist or are making their way into the market. These companies are bringing new pipelines, new tools and are inspiring new ways to think about game audio. It is worth mentioning a couple of these.

Fabric from Tazman-Audio is an audio tool for Unity, that works like an extension or package integrated directly into the Unity Project. It is probably more an audio tool/engine than a middleware. But in 2020 Tazman-Audio is releasing Fabric 3.0 which is going to be a stand-alone audio middleware for Unity and Unreal Engine.[26] ADX2 from Criware is another middleware, which is used in a lot of Asian titles.[27] Tsugi-Studio which is producing different asset- and pipeline tools for game audio, probably best known for their GameSynth Tool, is also offering a middleware solution with their GameSynth Engine that is capable of generating procedural audio.[28] Elias Software is a audio middleware mainly focused on adaptive music, but in 2020 they will release a new middleware that also focus on sound design.[29]

Even though middleware is becoming industry standard, some companies still work with built-in audio engines. A good example of this is Frostbite, which is the game engine of DICE. Frostbite provides unique functionalities that is hard to replicate in a middleware.[30]

[26] https://www.tazman-audio.co.uk/copy-of-fabric (06-12-2019)
[27] Appendix 5, p. 57, Interview_Guy_04.mp3 (06:45 - 07:10)
[28] http://tsugi-studio.com/web/en/products-gamesynth-runtime.html (06-12-2019)
[29] https://www.eliassoftware.com/ (06-12-2019)
[30] Appendix 3, p. 57, Interview_DICE_04.mp3

3.3 Implementation

The simple definition of implementation is the process of moving an idea from concept to reality. In this thesis we use the word to describe how you put your sound into the game. An example could be how you choose to implement a footstep sound system. You could choose to do it through animation, it could be done with physics or maybe by using the footfall curve of a character. The point is, that there are many different ways of implementing sound and how each game studio implements can be very different depending on the game engine, the middleware and personal preferences. So it is hard to determine the best way of implementation.

Sound as a gameplay feature

A traditional approach to implementing audio, is to think of it as something you implement "on top" of a system, meaning that sounds are often tied to objects or motion and then driven by this object or motions behaviour. In other terms, the sound is driven by the gameplay.

Another approach is allowing the sound design to control gameplay, motion or objects. Because of the way sound is implemented into Frostbite, the engine used at DICE, it allows the sound designer to make visual and gameplay content on the basis of sound. Frostbite and DICE is giving a lot of power to the sound designer through its modular audio/scripting system and build-in audio engine. This is giving the sound designer control of many aspects of the game. Like being able to control the visual intensity of an in game bonfire or the amount of camera shake.[31]

This way of thinking audio as interconnected with the gameplay of the game can also be a great help to other departments. In Game Audio Tales of a Technical Sound Designer Volume 1 by Damian Kastbauer, he explains how they build a system that allowed fire particles to be controlled by the RMS of the wind .wav files. Not only did they create a more realistic environment, but they also managed to fix a problem that the people working on particles wasn't able to solve.

[31] Appendix 3, p. 57, Interview_DICE_04.mp3 (0:44 - 00:55)

> "*We were really excited at this point, but even I was concerned about how this would go over with the artists(…)Surprisingly enough, they loved it. (…)the particle guys had been struggling big-time with issues like short loops and repetition; and their only real tool to combat this was to add more particles or systems to a given effect to make it appear moving and somewhat random–which of course hurt performance(…)We weren't a threat–we were problem solvers.*"[32]

This fear or misinterpretation of audio being something that is added to the game, instead of seeing it as something that can actually drive the gameplay, solve problems and give something unique to the game, is not helping game studios. Having the open mind for experimenting and interconnecting disciplines can create great results when building implementation systems and might end up helping instead of being a threat.

[32] Kastbauer, Damian, 2016, p. 197-198

3.4 Tools

Another very important technical aspect of creating a good pipeline is creating tools for the sound designer to use. While Implementation into the game engine also depends on a lot of different tools, tools does not have to belong in an engine. Many tools are also developed for other certain parts of the game audio pipeline, and comes in different sizes and shapes. In a way, a tool can be seen as the thing in our pipeline that makes it run smoothly from concept to implementation as stated by Guy Somberg:

> "*The goal (of a game audio pipeline red.) is to facilitate the flow, from ideas in the sound designers head. All the way through to get content into the game*".[33]

Like implementation, the amount of tools needed are different, depending on the need of the studio. Tools however can be shared across multiple projects, especially if the same middleware and game engine is being used. But it is important to constantly review and evaluate on what tools are needed and how to improve on existing tools. Ubisoft Quebec City Studio is using a review system of their pipeline tools, that they call *Start, Stop, Continue*.

> *"It is what we should start doing. What we should try to stop doing and what we should continue to do and build on, and then we used those approaches throughout the project and in the post-mortem meeting."*[34]

The idea is to always be aware of what works and what doesn't. If the need for a new tool arises, it gets added to the technical backlog and prioritized depending on how urgent it is.[35] This way a company can be aware of what problems lies ahead and be able to react in time.

Through our interviews we have found some common guidelines that a good tool should adhere to:

[33] Appendix 5, p. 57, Interview_Guy_Somberg_01.mp3 (00:40 - 00:50)
[34] Appendix 4, p. 57, Interview_Lydia_Andrew_03.mp3 (04:55 - 05:15)
[35] Ibid (04:55 - 05:15)

1) **The tool should try and ease one or several pain points in the pipeline.**

This could be reducing the complexity of a task or reducing the number of clicks needed to perform an action. The first task when designing a tool is to sit down and ask yourself, what are you trying to archive. If the creation of the tool requires help from a programmer, it is essential to focus on the problem you are trying to solve, not necessarily the solution you have in mind.

> "*I encourage sound designers to bring me problems, not solutions. What that does, it encourages them to go back to the original problem that they are having when they approach me. And then what i can do is listen to the problem and try to find a better solution than they might have come up with ordinarily."*[36]

2) **The tool should be able to speak the language of the user and help the user fulfil their creative potential.**

It is important that the tools created are easily understood by the people using them. For example making sure the UI/UX is understandable and that features are named accordingly to what they do. Otherwise it can cause frustration by the end user[37].

> *"Good word working tools should feel like an extension of the crafts person's hands, and here in audio it is the same, but more an extension of the crafts person's imagination*[38]*".*

3) **The tool should be expandable and be able to be constantly iterated on.**

The tool should likewise be granular and easy to update and maintain, often times they will get developed early in the production so it is important that they are flexible so that they can evolve.

> "*The only solution to this unpredictable situation is granularity—that is, the goal to build a flexible set of tools that are constantly able to update, evolve, and adapt to the needs of various projects and in their different stages of production.*"[39]

[36] Appendix 5, p. 57, Interview_Guy_Somberg_02.mp3 (02:52 - 03:17)
[37] Ibid (01:00 - 02:00)
[38] Appendix 2, p. 57, Interview with Rob Bridgett, p.1
[39] Guy, Somberg, 2017, p. 165

Storyteller

A good example of a tool, that adhere to above guideline are Ubisoft's Storyteller. Storyteller is a tool developed at Ubisoft for their Assassin's Creed Franchise. It was specifically created for their game Assassin's Creed Odyssey. Unlike the other games in the franchise, the game is made as an open world RPG and features a vast branching narrative. This lead to a massive increase in the amount of dialog that had to be produced and managed.

Ubisoft was facing multiple problems with the choice of narrative branching.

- They couldn't necessarily have the voice actors in the same studio, at the same time. This meant that they needed a tool, that would allow the voice actors to play up against each other.
- Each line of dialogue could have multiple answers, resulting in tons on lines. So it should be easy to navigate.
- Not only sound designers were going to work with and implement voice over, but also quest designers, game designers and other people dealing with gameplay. So the tool should be accessible and speak a common language.

To combat this, Ubisoft made Storyteller. The tool works similarly to *"choose your own adventure"* books. All dialogue lines are accessible and you can go back and forth to check the other branching options. This makes it easy to navigate and understand the manuscript, instead of having to manually search through tons of pages. The voice lines are store in a database where they get an ID and are named appropriately. This is pushed all the way through the process of production.[40]

On top of this, Storyteller is hooked up to Reaper. From Storyteller you can get direct access to a Reaper project for each specific line. In these Reaper projects, the dialogue is recorded and saved within a region. The region is making sure the recording gets the correct name and ID. This way the recordings are accessible for everybody involved in the pipeline, and slowly over the various recording sessions,

[40] Appendix 4, p. 57, Interview_Lydia_Andrew_07.mp3 (01:02 - 02:15)

references are getting filled out. This means that the voice actors are able to play up against actual recorded dialogue, making the dialogue more coherent.[41]

Ubisoft has taken a problem, and found a solution that not only reduces the amount of management needed to be done by the team, but also inspires and helps the voice actors and enables a clear creative vision for the voice directors, sound designers and everybody else involved.

[41] Based on Lydia Andrews presentation at the "Leading With Sound" Conference at Den Danske Filmskole (06-10-2019)

4.0 Organization

The size of video game companies can span from everything between one person working on one project up to hundreds of people working on multiple projects. The time span also goes from short periods up to many years of development. Many companies see big changes in their staffs. Employees leaves, new joins in, companies expand, companies shrink. Planning and organizing is core to keeping the projects on track all the way through production.
When developing your pipeline it is important to plan out how you want to organize your project from day one. Most of the previous chapters have already been touching on the subject of organization. Will you be using a structured or free approach to the use of DAW? What audio middleware will you use or maybe you will not use a middleware? What tools will you have to create, to streamline the pipeline? These are all decisions you will have to take, when designing a pipeline and in this perspective organizing your pipeline.
In this chapter we will look at organization from the point of how to keep consistency throughout production. We will look at how you make your pipeline transparent, the importance of establishing a naming convention and use categorization, and we will look at how you organize you audio team and the importance of cross-disciplinary communication.

4.1 Transparency

The word has been used a couple of places throughout the project, so first of all it is important to understand what we mean with transparency. In the first chapter we talk about what a game audio pipeline is. It is mentioned that working in between the different parts of the pipeline needs to be frictionless. An import part of making the pipeline frictionless, is to make the pipeline transparent. To do this, you need to organize and come to agreement with both the audio team and everybody involved in the process of the game audio pipeline, how the pipeline is going to be executed.

This involves figuring out the steps explained through this project, create a consistent naming convention (which we will talk about shortly), what format should different types of assets be exported as, what is the hierarchy structure in the middleware and what tools are available and how should they be used.
Gustav Rathsman from DICE uses the term "*One True Source",* which is a great way to describe this need for coherence and transparency throughout a project: *"That is a classic problem, where people have different sources of truth and then you end up with something that doesn't align in the end."*[42]
The keyword here is communication. Communication is absolutely essential to keeping everyone in the loop on what is happening.[43] Having these steps laid out as early as possible, makes it much easier to adapt changes and when something is not working (and something won't work), a transparent project is going to make the problem solving much easier.

4.2 Naming Conventions & Categorization

"Naming conventions are one of the cornerstones of an organized project, and whatever your discipline, a solid, sensible naming convention will save your skin as you hit full-on production."[44]

When we asked our interviewees about naming conventions, it became clear that organizing your assets is very important and a system needs to be established as early as possible.

The naming convention needs to be carried all the way through the game audio pipeline. From the moment the creation of assets starts and all the way to the end of implementation. It is therefore recommended to keep the same naming convention all through this process. This goes for the assets, the folder structures of your library,

[42] Appendix 3, p. 57, Interview_DICE_06.mp3 (2:45)
[43] Bridgett, Rob, 2019, p. 64
[44] Ibid, p. 84

the hierarchy structure in your middleware, the events in you middleware and the implementation in your engine.
But it doesn't have to stop there. Many game audio departments try to maintain a naming convention in the studio and not only for their current project.
DICE is using a folder structure for their assets that goes all the way back *Battlefield: Bad Company* (2008). By doing this they have a type of folder structure that everybody are familiar with, and they don't have to re-learn a new structure every time they start working on a new project. On top of that, their naming convention for assets is following this folder structure. The reason for doing this, is because it's stupid simple, and when it's stupid simple, it is less likely that anybody is going to make mistakes.[45] This is a great example of transparency.

But how should a naming convention be structured? Our interviewees have given us a lot of great examples of naming conventions, but there is no magical solution for this and often it can vary, depending on the type of game you are working on.
But the two of the most import rules when establishing a naming convention, is to:

1. Make it very clear what the asset or file is and what is it connected to.
2. Have it appear in sorted order, for easy search and browsing.

Stephen Hodde provided us with this template for the naming convention he is using:

Category_Subcategory_Name_Layer_VariationNumber[46]

A example of this naming convention in practice, could be:

Weapons_Pistol_Glock_Impact_010

Using this naming convention makes it easy for everybody to understand the purpose of the file, and when sorted in order it will be easy to find.
A lot of the naming conventions we were provided with had similar structure, but the categorization and additional categories is often where games will differ.

[45] Appendix 3, p. 57, Interview_DICE_08.mp3 (3:30)
[46] Appendix 1, p. 57, Interview with Stephen Hodde p. 2

This is an example from DICE:

SW02_Planets_Takodana_Shared_Ambience_SoundArea_UnderForestCanopyLightBirds_01[47]

In this case they start by defining the title origin of the asset. In this case Star Wars Battlefront 2. Followed by specifying the planet. In this case there is multiple map levels on the planet *Takodana*, and this sound is *Shared* between these levels. This it then followed by a categorization similar to the one of Stephen Hodde.

Often companies will use abbreviations to minimize the length of characters. Examples of this would be *CHA* for Character or *AMB* for Ambience. It can be more manageable with shorter names. But, when using abbreviations it is very import to agree on the abbreviations used. Having some sound designers name their sound effects *SFX* while others name them *FX*, can end up creating a lot of problems. Agreeing on the *One-True-Source* is important when creating a naming convention.

Other typical additions to the naming convention is remarks on dynamics and velocity, typically described with *"small, medium, large"* or with the dynamic marking used in classical music *"pp, p, mp, mf, f, ff"*, and the use of numeric to distinguish variations. Rob Bridgett advises the use of additional numeric to prepare for iterations and in the case with dialogue lines, prepare for changes in the manuscript. Rather than use 01, or even 001 with leading zeros only, then allow an extra decimal place in all numerical file naming:[48]

*Amb_Day_Bird_Dove_pp_001**0*** or *Amb_Day_Bird_Dove_pp_001_**0***

"Precise audio terminology is the key to improving communication and achieving a more granular and detailed level of information, regardless of the target platform or development software used. Be it Event, Soundbank, Sound Definition, Parameter, RTPC, Preloading, Trigger, or Area Shape, reinforcing the precise usage of these terms will help to identify, debug, and resolve potential issues much faster."[49]

[47] Example provided by Ben Minto, see Appendix 3, p. 57, Interview_DICE_08.mp3 (4:30)
[48] Bridgett, Rob, 2019, p. 87
[49] Somberg, Guy, 2017, p. 170

4.3 Generalist vs. Specialist

In the game industry there exist a career hierarchy much similar to that of the software industry. In the game audio department this is also the case. Depending on the size of the sound department these roles can vary a lot. But typically from the top down you would have audio directors, audio leads, senior sound designers, sound designers, junior sound designers, audio interns. Furthermore the team would have technical sound designers, composers, audio programmers, audio QA and so on.

When it comes to hiring new sound designers for the team and distributing the work, there seems to be two different approaches. Either you sound department consist of specialists or it consist of generalists.

Specialist

When a game audio department works with specialist, every person working on the audio team will either be responsible for a specific task or category of sound. These can be in broad terms, down to more narrow subjects. Typically this will be areas like dialogue, foley, cinematics, music and implementation. The more narrow subjects could be environment, characters, weapons, UI, vehicles and so forth.
The idea with this approach is to create the best possible sound design in the individual areas by having people work with subjects that they are experts in. It is also a way of delegating the work, to make it clear who is responsible for what

This way of thinking in specialists probably derives from the classical film production, which consist of a lot of specialist working with their individual part of the audio. The creative problem with this, both in terms of film and video game production, is that the specialist can have a hard time understanding or adapting their work to the wider contextual elements of the production.

> *"if someone is just doing Foley recording to picture, but never hears or is able to get a sense of their work in context of the rest of the soundtrack, like the music cues, the SFX cues, ambience and so on, then they are going to do much more than is necessary and spend 100% of their time and effort and*

> *storytelling in the creation of the Foley track. The majority of this work is then going to be removed during the mixing phase of the project. This is an example of a very inefficient working method whereby someone responsible for each element of the soundtrack, attempts to tell 100% of the story with the one area they are responsible for."*[50]

It is therefore important to regularly make gameplay sessions, meetings and other ways of involving and keeping everybody on track with the creative direction of the project. This goes without saying no matter what type of approach is used, but the remark is important.

A disadvantage of the specialist approach is the risk of becoming to specialised. This is both on an individual level and on a company level.
Companies working with many specialists are putting them self at risk. If a specialist gets sick, then the knowledge they hold isn't accessible. Even worse if a specialist leaves the company then the knowledge might be lost.
A good example of a specialist could be a composer. Composers usually gets hired as specialists, not only because they are specialised in composing music for games, but often also because of the style of music they are specialised in. The need of this specialisation will in most cases be freelance hiring in short periods of production.[51]

Generalist

Rob Bridgett is a big advocate for the generalist approach. He describes the generalist approach like this: *"that any one person in the team does not monopolize knowledge, expertise or techniques that the other does not possess."*[52]
The generalist contra the specialist, is a person who is able to do anything on the game audio team. It is a balance between being creative, technical and collaborative. Each element ideally at 33.3% of the person's ability and focus.

[50] Bridgett, Rob, 2019, p. 36
[51] Ibid, p. 38 - 39
[52] Ibid, p. 36

> *"These three things will all feed into each other - a discussion (collaborative) can spark an idea (creative) which then will be prototyped (technical), discussed (collaborative) and refined (creative) and iterated upon (technical) - and so on."*[53]

When having the generalist approach, the sound designer should be able to go into any particular map or element of game and start working, without the need of consulting a specialist beforehand.

What approach to use is also a question of scale. The smaller the team, the more areas does the sound designer have to cover. This makes the generalist scalable. The generalist can function as the only sound designer during a whole project, but can also be part of a bigger team of multiple sound designers. The specialist on the other hand is only applicable in bigger teams or as a short term freelancer on smaller projects.

4.4 Collaboration

Even though it was stated that the thesis won't go into details when it comes to collaboration with other departments, it is still important to emphasize on the need of collaboration between departments, when talking about organizing a project. Creating a video game is a group effort of many different disciplines coming together. The pipeline of each department doesn't exist in isolation and treating it as so will defeat the goal in the pursuit of transparency. It is therefore important to plan meetings, stand-up, communicate and use shared documentation throughout production, to make sure that everybody involved stays informed on the direction, both within the audio team and across disciplines.

[53] Bridgett, Rob, 2019, p. 38

"Regardless of whether you are a small group of independent developers or part of a big production team, there is no excuse for not having a structure! At the end of the day, game development is a team effort where very specialized and passionate people come together to create a product. We need to use all the tools that are available to us to make production as simple and efficient as is possible. We need to build a solid production framework where we can maximize the time we spend on the creative part of the job."[54]

[54] Somberg, Guy, 2017, p.175

5.0 Key Takeaways

In the first section of this thesis the focus has been on the different stages and practices of a game audio pipeline. The purpose of this has been to create the foundation and outline the principles and elements that goes into creating an game audio pipeline, by examine each element and the different ways of solving them.

Working in game audio is a balance between working creatively, technically and collaborative. You need to establish a good pipeline that is as frictionless as possible. The fewer questions the sound design needs to ask the better. So in order to provide your sound designer with the best creative environment, you have to spend time on how you manage every step in your pipeline from asset creation, to implementation, to iteration, to finished product.

By looking at the evolution of game audio, we understand how game development is in constant change. Pipelines have changed completely over the last decades with the advancement in hardware and the introduction of better tools, like the audio middleware. As the scope of games are changing so is the need for optimization and streamlining.

The pipeline starts with a concept and it stops when that concept can be heard by the player in game. How this process is managed is different from company to company. But creating a pipeline is a planning process and deciding on every aspect as early as possible is going to save time and make the process of creating a video game more transparent for everybody involved.

When it comes to working with DAW's we see two approaches. A structured approach where all sound designers on the team are working in the same DAWs with the same plugins. This approach is focused on modularity and transparency.
A free approach where sound designers work with their preferred DAWs, and plugins. This approach is focused on helping creativity and productivity, by having people work with the tools they have muscle memory in.

But we see a tendency that people are preferring DAWs that are modular and offers pipeline functionalities like batch processing and middleware integration.

The advancement of audio middleware has had a big impact on the possibilities for game audio. It has provided sound designers with a shared language. Day-one implementation of fundamental features and cross-platform flexibilities, means that sound designers and audio programmers can spend their time coming up with better and more creatively and technical solutions. Most middleware offers the same fundamentals, but their pipelines differ greatly. The decision on which one to use should be made according to the composition of the audio team. But even though middleware has become industry standard, game engines like Frostbite do offer unique pipeline and implementation possibilities, that shouldn't be underestimated.

The way game studios implement sound into their games is very different from studio to studio, project to project, and no system is alike. But having an open and creative mind while developing these system, is helping audio get better connection with the other disciplines in a game development environment.

Tools are an important technical part of a pipeline, and can be shared between different projects in the studio if needed. When developing a new tool, it is important to try and adhere to these three basic guidelines:

1. The tool should try and ease one, or several pain points in the pipeline.
2. The tool should be able to speak the language of the user and help the user fulfil their creative potential.
3. The tool should be expandable and be able to be constantly iterated on.

When developing your pipeline it is important to plan out how you want to organize your project from day one. All of the above mentioned subjects and the options presented offers different solutions, but they are all decisions that needs to be made when organizing a pipeline.

Striving for transparency is a subject that resonates throughout the thesis. Transparency is achieved through communication, planning and organization. By making sure that everybody stays consistent and follows the *One True Source,* that was decided upon in the pre-production, mistakes and questions are less likely to occur and easier to treat.

Having a naming convention and sticking to it, all the way through production is very important. We have shown some examples of how a naming convention could look like and how it could be used. There is no golden solution, but a good naming convention needs to follow these guideline:

1. Make it very clear what the asset or file is and what is it connected to.
2. Have it appear in sorted order, for easy search and browsing.

We have presented two different ways of approaching the people working on the audio team. The Generalist approach and the Specialist approach. The decision on which to go for, depends on the needs of the company.

Game development is a team effort spread over many different disciplines. This makes communication a vital part of the daily work and a key to success. This both goes within the audio department and cross-discipline. Establish ways of communicating to keep everybody informed about the state of production, is an essential part of planning a pipeline. There are many ways to this and will typically be in form of meetings, stand-up and shared documentation.

"I do think that the concepts of pipelines are very interesting, because it's not just what tool I'm using, what game engine I'm using. Am I using Wwise. Am I using FMOD. Am I using Frostbite. It's not that. Pipelines really for me are about that system, that systemic or holistic approach of how are we organizing ourselves. Physically, creatively, technically, in order to optimize our work, to optimize our player experience. Because ultimately that's what gets into the game, what the player sees, which is what you are aiming for."[55]

[55] Appendix 4, p. 61, Interview_Lydia_Andrew_15.mp3 (18:55 - 19:45)

6.0 Pipeline Development

Now that we have explored the concepts of a game audio pipeline, it is time to take what we have learned and apply it to our case study.
The game we have chosen as our case study, is the 3D Game Kit from Unity. The game is a third person adventure game, that features some minor puzzle and combat elements. The game takes place on a foreign jungle looking planet, where you have to explore the ruins of an ancient civilization. The game is made in Unity, so that will be our game engine. We will use Reaper as our DAW and Wwise as our Middleware. The reason for this will be elaborated.

To try and make the case as realistic as possible, we have chosen to view the game itself as a vertical slice of a bigger production. The company is in pre-production and our goal as sound designers is to make a robust game audio pipeline, but also a pipeline that is easily expandable to accommodate the extra features that might be available in the full game. A playthrough of the finished product can be found in Product Appendix 10 p. 56.

6.1 Organizing

The first thing we are going to decide on is how we organize the project. At some point the game is going to scale and more sound designers might join the team. Therefore it is important that our project is transparent from day one. We are also going to work closely with other departments, so naturally the company will already be planning meetings, documentation and deciding on a product pipeline that also includes audio. In the case of the audio department we have decided to use *Jira* for backlogging and keeping track of tasks.

Naming convention and categorization
Another important aspect to decide upon before we start producing is how we want to organize the project when it comes to categories, hierarchies and naming conventions.

For naming convention we have chosen to go with a modified version of Stephen Hoddes naming convention, with the option for adding a Dynamics category and a remark that a file can have multiple Subcategories.

Category_[Subcategories]_Name_Layer_[Dynamics]_VariationNumber

The reason for this style of naming convention is because it is flexible and can easily be expanded as the game grows in scope.
We have decided to not use numeric for iteration, this is because our Reaper to Wwise export/import tool, that will be explained later, doesn't allow additional symbols after the variation number. But this will happen if we were to include voice over in the project.

This naming convention is used all through our pipeline. This means that our Originals folder, Wwise hierarchy and our Wwise event folder will follow the same structure. This is with the purpose of making it transparent and easy to navigate.

6.2 Choice of Daw

We have decided to use the structured approach, when it comes to the use of DAW. Ideally we would also use the same suite of plugins, but because of financial limitations we will be using the plugins that each of us has available.
The DAW that we will be using is Reaper. This is foremost because we both are quite used to working in Reaper, but also because of the deep automation and scripting possibilities within the engine. We want to reduce the time spend on exporting assets and importing them into Wwise. This is because it is a very time consuming part of the pipeline, that we know we are able to improve significantly on, with the help of existing tools. Below is a description of the tools that we will be using to streamline this process.

6.2.1 Reaper Export/Import Pipeline

We have chosen to focus on two points of improvements for Reaper. A shortcut to easily make a endless loop from a media item and a collection of tools that makes naming and exporting assets from Reaper into Wwise easy.

Seamless Loop Shortcut

The first tool is quite simple and just makes a loop from a media item by cutting the file in half at nearest zero point crossing and moving the left half across to the other side and make a crossfade. This tool is a collection of scripts built into Reaper and made into an action. It is a quick improvement of a task that will be done many times in the course of production.
See Product Appendix 1 p. 55 for a video demonstration of the tool.

Reaper to Wwise system

This system is a bit more complex and is made up by a set of tools made by others. It consists of three independent scripts or systems and utilises the region render functionality to export and it makes naming multiple assets very fast. Unlike the standard way, this method is also able to easily export sounds that consists of several layers.

The first step in the process it to create some tracks and have one of them be a folder for the others. This track must be renamed the following way: *Regions 100 NameExample*. The naming itself won't matter but are used by one of the scripts later in the process to determine what regions to create. You then go on and create the sound how you normally would. You can layer sounds and do as you normally would.

The next process is to create text items on the folder tracks, so that we later can use these to make render regions. This is done by an action that consists of several different scripts put together. You highlight all the media items you want to name and call the action. This prompts a dialog that is used to name the text items files. They will all get a number added as an Affix e.g. *"Item01, Item02, etc."* These scripts are

part of a packages you can download off the internet and is not written by us they can be found here.[56] A big thanks to *X-Raym* and *spk77* for making these available for free.

After this, another script made by *HeDa* then takes these text items and creates regions around them. Now all there is left is to open the Regions Render Matrix, add the tracks you want to export and then open then Renderer, select the appropriate setting and add the regions to the Render Queue. For the last part, you open the WAAPI Transfer Plugin from *Karl Davis* and import you sound into Wwise. Unfortunately this tool is only available on windows due to the different file structure that Mac OS uses.

This workflow enables the user to fast and efficiently name, manage and import several audio assets into Wwise, while making sure that there are minimal mistakes in the naming convention. One thing that could be made better in terms of workflow is, that you still must go into the Wwise project and select the containers you want to import into. It would be great if you could do that directly in Reaper.
See Product Appendix 2 p. 55 for a video demonstration of the tool.

ReaOpen system

Our pipeline between Wwise and Reaper is also able to go the other way around thanks to the ReaOpen app made by *Nikola Lukic* from Audiokinetic. It works by tagging all the exported files with metadata of the project location on the hard drive. Wwise then reads these and puts them in the notes of the sound. When you want to open the project your sample was created in, it reads the note and opens the .rpp[57] file at the appropriate place. This is quite useful if you quickly want to make changes to your sound inside Reaper. However, you need to share your projects with colleagues if you want to open each others sessions, which won't necessarily be completely beneficial for us, since we might be using different plugins.

[56] https://www.extremraym.com/en/my-reaper-scripts/ (18-12-2019)
[57] *Reaper Project File*

With the implementation of these systems in our pipeline, we have considerably reduced the export/import time, even when it comes to iteration, while also reducing the risk of making naming mistakes.
See Product Appendix 3 p. 55 for a video demonstration of the tool.

6.3 Choice of Middleware

As our middleware we have chosen to use Wwise. This is primarily because of the control that Wwise is giving the sound designer, when it comes to build-in features. Especially in terms of working on a 3D game. Wwise offers many tools, such as the ability to use obstruction, occlusion and set multiple positions of a sound. This is compared to FMOD that also has a lot of these functionalities, but it would require a programmer to get all features implemented. Based on our mutual interest for programming and system creation, we believe that we will be able to utilise Wwise, the pre-existing Wwise tools in Unity and additionally build the tools and systems inside Unity without the immediate need of a dedicated audio programmer.

One could argue that Unitys own audio system would be viable. Partly because of how you're able to export audio assets directly into the game engine with no need for soundbanks. But the amount of time we would need to spend on designing and implementing even simple features like pitch variation, does simply not hold up against the amount of features we get out of the box, when using a middleware solution. And with the export/import workflow between Reaper and Wwise that we have developed, we already have a fast iteration process.

As mentioned earlier, the structure of our Wwise project, when it comes to the hierarchy and event folder, is following our naming convention. Additionally we have decided that the bottom layer is made with folders and the second layer is Work

Units.[58] What this means is that it reduces the chances of us overwriting each other's work and it saves time and focus, by reducing the chance of us having to wait and get distracted, when a Work Unit is occupied by another sound designer.
Pictures of how this looks like can be viewed in Appendix 8 p. 59.

6.4 Tools and Implementation

In terms of audio implementation, most of the logic such as the behaviour of triggers, monsters and pickups, has already been set up by the programmers. The game itself is designed to be very modular, so in terms of implementing sound on the different objects it was a matter of creating a new system to replace the old and additionally look at what tools we would have to create, to make the implementation as transparent and creative as possible.
We started by brainstorming different ideas that we either saw an immediate need for, read/heard about or in general was missing in our usual pipelines.
When mapping out the tools, we wanted to follow the guidelines presented earlier in the thesis.

1. The tool should try and ease one, or several pain points in the pipeline.
2. The tool should be able to speak the language of the user and help the user fulfil their creative potential.
3. The tool should be expandable and be able to be constantly iterated on.

But we also wanted these tools to be reusable. Being able to reuse a tool for different projects is valuable for a company, because it eliminates the need of recreating everything from scratch, every time a new project is started and the staff doesn't have to relearn new tools as well.
In the following pages we will present the different tools we created, with a focus on why we made them, and how they work.

[58] When working with version control and Wwise, Work Units functions as the base file, which is overwritten when changes has been made. Therefore you can't have two people work in the same Work Unit.

6.4.1 WwiseType System

Why?

Implementing events, RTPC, states and switches into the rest of the code of the game can be done in many ways. With the introduction of *WwiseTypes*[59] this implementation has become easier, because the need of using strings or custom ID systems has become unnecessary. We wanted to further develop on this system and minimize the amount of information and communication needed between the sound designer and the programmer, and the amount of code the programmer needs to write every time he/she is implementing *WwiseTypes*. When setting up an event through code, the programmer needs to at least specify what emitter the sound should play from. If the event should override the event action or have a value assigned, then the programmer needs to write even more additional code. All this information should be communicated from the sound designer to the programmer and if something changes, some of the code may need to be revisited.

How it works

The idea with this system is, that the only information the sound designer needs to give the programmer is what type (event, RTPC, state, switch, bank, auxbus) should be used and when it should be played. The only code the programmer then needs to write is a declaration and a call of the declaration, as shown in Appendix 9.1 p. 60.

All the code determined behaviour of the *WwiseType* is nested inside the serialized *WwiseEvent* class. This is deriving from the parent *WwiseSystem* class. Similar child classes for RTPC, state, switch, bank and auxbus is also available. The behaviour of the derived class is then exposed in the inspector (as shown in Appendix 9.2 p. 60), giving the sound designer the control to setup and change the behaviour without having to communicate with the programmer.

When it comes to the three guidelines of what makes a good tool.
First of all it is fixing a common problem/question, which is the question of how

[59] https://www.audiokinetic.com/library/2017.1.9_6501/?source=Unity&id=unity__wwise__types.html (10-12-2019)

should audio be implemented into the game. It is providing a simple solution, which is reducing the amount of time it takes to set up *WwiseType* behaviour. It reduces the amount of detailed communication needed between the sound designer and the programmer. It reduces the amount of maintenance, by giving the responsibility of sound maintenance to the sound designer, who is the one who ideally should be responsible.

Secondly it is speaking the language of the sound designer by shifting the setup from the code and out in the inspector. Typically the programmer would try to make normal integration look good in the inspector for the sake of transparency. But this isn't needed, because the setup has already been decided within the system. This is saving a lot of the programming time.
Further it comes with highlight tooltips that explain the behaviour of the different functions, as shown in Appendix 9.3 p. 65.

Thirdly the tool is expandable. It is already being used in multiple of our other custom tools, *WwiseAnimation*, *WwiseAmbienceFollower* and *WwiseTrigger*, which makes it an integrated part of our implementation pipeline. The tool is still in an early development phase and could be expanded to include all WwiseTypes in one single class, the UI could be improved with custom drawers and it could include an in built callback system, making it very useful for music implementation.

But this isn't a tool that is going to remove the need of writing *WwiseType* behaviour ever again. Currently it isn't applicable in our *WwisePainter* and *WwiseMaterialSwitch* tools, because of the specific ways they are accessing and using *WwiseType*.

The idea for this system comes from the event action system developed at *ThroughLine Games*. The big difference between the system of *ThroughLine Games* and the system we built, is that we implement the *WwiseType System* directly into the code of the component we want the behaviour to come from. The system used at *ThroughLine Games* is sitting on its own component and is then referenced by other components. Using our system reduces the number of components and game

objects in the project, making it more manageable and more transparent when it comes to eliminating the confusion of the pipelines execution.
See Product Appendix 4 p. 55 for a video demonstration of the tool.

6.4.2 Custom Third-Person Listener

Why?

This custom Third Person Listener we have implemented is one that Guy Somberg mentioned in his book and in our interview with him. Since our game case is a third person listener game, we thought we would give this method a try, and it turns out this implementation made a dramatic difference, compared to simply placing the listener on the camera. Since we need to get the players position and that the way you would expose this would be different from every game, it can't really be considered a tool but rather an implementation.

How it works

Having the listener attached to the listener or player creates some artefacts in the way you would expect to hear the 3D sounds in the game world. Having the listener on the camera causes the sounds to be oriented correctly but throws off the distance attenuation. Doing the opposite and placing the listener on the players head would create the opposite problem and make the orientation wrong, but make the attenuation correct.

Instead, what you want to do is to place the listener at the position of the player but use the camera as the method of rotation. In Wwise this can be done by simply calling `Aksoundengine.SetPosition()`. This method takes in a transform and a Vector3.forward and a Vector3.up.[60] All you must do is set the listeners transform to be equal to the player position and then set the orientation to be equal to that of the camera. See Product Appendix 5 p. 55 for a video demonstration of the tool.

[60] https://docs.unity3d.com/ScriptReference/Vector3.html (18-12-2019)

6.4.3 Wwise Material Switch

Why?

The main idea with the Wwise material switch is to give the sound designer an easy way of organizing materials and different Wwise switches. It is designed to be as agnostic as possible meaning that it does not decide how you want to "get" the material from the unity game object. This opens the possible of reusing the tool for not just footsteps, but also for different applications like Weapon impacts where you would need a material to determine what sound to play. .

How it works

On the outside the setup is made as simple as possible and consist of a scriptableobject where you assign materials to an array and a corresponding switch. The idea is that you just needs to assign a switch and the materials you want the switch to trigger on. As shown in Appendix 10.1 p. 61.

On the implementation side you create a *WwiseMaterialSwitch* variable and call the `Initialize()` function when the game object is enabled. This function can be seen in Appendix 10.2 p. 62. Wherever you do your material check, call the `CheckMaterial()` function. This function takes in a material and a game object. It will try to change the switch, to the matching material. If it can't find the material, it will keep the material switch set as default. This function can be seen in Appendix 10.3 p. 62.

The reason for having to provide a game object is to give the option of specifying what game object you want to change the switch on. Again, the idea is not to retrieve the material itself, because the way it is done can vary from implementation to implementation. The way it is done for a footstep system, may not be the same for a weapon impact system. See Product Appendix 6 p. 55 for a video demonstration of the tool.

6.4.4 Wwise Ambience Follower

Why?

The idea behind the ambience follower is to have a way of creating "zones" you can move in. This could for example be a forest. Here you typically want the sound to be 3D the further away you are, but 2D when you are "inside" the forest. This tool makes it easy to define areas, but also makes sure that they fade correctly and reduces time the sound designer has to spend on making overlapping ambience zones fade correctly

How does it work?

The component works by spawning an emitter inside a collider. The emitter will try to follow the player, but confines itself to only move inside the collider. In Wwise you define a RTPC that changes the sound from 2D to 3D based on the distance to the player, to get the feeling of "entering" the area. It can also be used with only distance but no panning, to make 2D ambiences fade in and out over distance. It is possible to set the update interval, to optimize the performance, since there is no need for updating the position every frame. Inspector values can be seen in Appendix 11 p. 63. The max distance is also a performance optimizer. It makes sure that the event isn't playing and the position being updated, when the player is further away than the max distance. See Product Appendix 7 p. 56 for a video demonstration of the tool.

6.4.5 Wwise Sound Painter

Why?

The Wwise sound painter is made as a tool to help sound designers quickly marking areas that should make up a volumetric sound area, such as water areas. Without this tool, you would manually have to place game objects by hand, where you want the area to be, often leading to a very tiresome and slow process. This tool significantly reduces the time spend doing this by enabling you to place the emitters where you want them, simply by clicking in the scene view. Making it simple and very intuitive, even for people not used to place game objects in a 3D environment.

How does it work?

At it most basic level this uses the Wwise provided `AkSoundEngine.SetMultiplePositions()` function to set multiple positions for one emitter, enabling the sound designer to make entire area that share the same instance of a sound, improving performance a great deal.

The tool itself is split up into a standard scripted component and 3 editor scripts. The component added to the game object lets you select what sound you want to play and manages the multiposition game objects. As shown in Appendix 12 p. 63. Selecting the game object on which this script resides in the unity hierarchy, gives a "Stamp" when hovering above your scene viewport. Left clicking when this stamp is active, creates a point in the scene. This is done by doing a raycast down from the scene camera and then spawning a game object with the *AkAmbientLargeModePositioner* component on itself, allowing Wwise to correctly add it to the *AkPositionArray* it uses to compute the different positions.

The layer you can "paint" on can be specified by the *Layer Mask*. And the max slope determines the about of degrees you can paint on, meaning if walls and structures should be taken into consideration. The tool is also able to show you the attenuation curves of the points, enabling the sound designer to see the extent of the sound, and could in the future be extended into a prefab painter, allowing you to paint more than just multi position emitters. See Product Appendix 8 p. 56 for a video demonstration of the tool.

6.4.6 Wwise Animation

Why?

In many games it is normal to have audio being played at specific points in the timeline of animations. In Unity this is done using animation events. This is events that can be placed on the animations timeline. These events activates a specified function, that can take in one variable, when the animation is being played. Creating a new function for each event is ineffective and can easily become chaotic when the numbers of animations and events grows. We wanted a flexible system that can adapt to an increase in animation events and it should be reusable for every model.

How does it work?

The *WwiseAnimation* script is a modification of a system from *ThroughLine Games*. By creating a list of *WwiseEvents* using the *WwiseType System* described, the sound designer can specify the number of events that is going to be needed directly in the inspector. The sound designer then chooses the Wwise event that each *WwiseEvent* in the list should play. An OnValidate function then automatically renames each *WwiseEvent* in the list, to the name of the event chosen.
Mentioned before, the animation event in the animation timeline takes a function and a variable. The function is called *WwiseAnimEvent* and the variable we are using, is a string with the name of the *WwiseEvent*, that we renamed in the OnValidate function. The function is shown in Appendix 13 p. 64.

When *WwiseAnimEvent* is called from the animation timeline, an enumerable *FirstOrDefault* method runs through the list of *WwiseEvents* until it finds the one that matches the string value of our event.

A problem with this pipeline is that the function and the string name needs to be manually written in the animation event. To improve on this we have added the option of copying the Wwise event name to clipboard. This eliminates the chance of mistyping the event name and it slightly improves the time spend on setting up animation events. See Product Appendix 9 p. 56 for a video demonstration of the tool.

7. Closing Thoughts

Taking a step back and evaluating what we would need, and how we were going to organize the audio pipeline, helped us immensely during the creative process. Deciding on *One True Source* in terms of a clear naming convention, a transparent pipeline execution, deciding and building the tools we knew we would need, has made the process from idea to execution faster and more transparent.
By using the Reaper to Wwise tools, we were able to drastically decrease the time spend when exporting assets from Reaper, compared to using a DAW without batch processing or similar tools. The Unity tools we created made the implementation quick and easy to do. It offered us ways of speeding up the process, like painting multiposition emitters and quickly set up simple things like animation events. While also improving the listener experience, like the custom third-person listener.
In the end, these ways of working is letting us get much more creative, because of the shorter iteration and implementation time and the lesser time spend on patching bad management.

In our case study we haven't been part of production from day one, but probably pretty late in the pre-production. If we were part of the production from day one, we would probably have done a lot differently. Coming into a project where many things have been decided upon in advance, will make the process harder.
An example could be the fact that all animations in our case study, were imported as read-only, which makes the implementation of animation events more time consuming. Being able to plan the best practice with the animation department before they started working on the assets or we started working on our tools would be ideal. This goes with many things and it only emphasizes the fact that audio should be part of production as early as possible.

As mentioned earlier many of the tools we have designed are designed with the purpose of being reusable in other games. Most of our interviewees mentioned that they reuse tools between projects. As a freelancer, you might not be able to reuse tools from one project to another. Many studios are using different game engines

and middleware, and the way they are structuring their code may also be very different. But with the growing popularity of general game engines like Unity and Unreal Engine. We could potentially see a future where freelancers more easilier can bring in their way of working and the tools they have created, onto new projects.

8. Conclusion

Through the thesis we have presented some of the key-points of consideration, when developing a pipeline and we have given multiple ways of approaching these. But it is important to emphasize the fact that first of all, there is no right way of doing this. It is rather a question of making the decision on how to do it. Secondly, there is much more going into the development of a pipeline, then what we have been able to present. Because of the scope limitations of this thesis we haven't been able to make room for working with moodboards, documentation and cross disciplinary pipeline development. But to get a full understanding of how to develop a pipeline, these subjects should be taken into consideration.

When developing a pipeline, that is going to stay organized, transparent and stick to the idea of *One True Source,* having audio be part of pre-production is vital. It is impossible to prepare for every situation, but having outlined and tested the pipeline, is going to provide higher planning stability. That being said, being part of pre-production is also a question of acknowledging everybody working in the game industry as being experts in video games. The knowledge they bring to the production, is more than just their field of expertise.

We see a lot of commonalities and differences in the pipeline execution of our interviewees. But we do also see a tendency that studios are starting to use the same middleware solutions and uniform their use of DAW's. In the case of indie companies, this also goes with game engines. With more studios starting this progression towards the same tools, we could see some kind of global pipeline streamlining, especially when it comes to tech.

It is possible to develop reusable tools. But not only is it possible, it is also something that should be striven for. When developing a pipeline and the respective tools, it is important to treasure it as something that is going to last beyond the current project. The game audio pipelines and tools at many of our interviewees respective

companies, have been developed over the course of many years and projects, and they still find ways to improve upon them.

The hardest part of working with a determined pipeline, is keeping it consistent if things should change, which inevitably will happen. There is no universal answer to a perfect pipeline, and the pipeline we have built is by no means tested and true and a lot of things could still be improved upon it. A pipeline is developed through experience, trial and error. That is also why it is important to be aware of how you are working, what steps you need to go through from A to B and then try to find ways of improving it. Time spend working against the pipeline, is time you could have spend on being creative. In the end, that is what the player experiences.

Abstract

This thesis aims to provide the reader with an understanding of the importance of designing a game audio pipeline and how to create a pipeline, that streamlines and optimizes workflow and game audio implementation. To do this, an examination of the various elements that goes into creating a pipeline and what considerations to take into account, is made. This information is then exemplified in a case study of a third person adventure game. The game is treated as a prototype in pre-production and aims to showcase how a pipeline could be designed. This includes various pipeline and implementation tools, designed with the case study in mind.

The study concludes that there is no universal answer to building a pipeline, but that it is important to be aware of how you are working, what steps you need to go through and then find ways to improve upon it. Building a pipeline is a question of organizing, outlining and testing, with the purpose of providing higher planning stability. It is impossible to prepare for every situation, which is why the importance of having game audio be part of the pre-production phase of game development is being argued. Pre-production is the initial phase of production and deciding upon a pipeline this early and keeping it consistent through production is vital. It will make problem solving easier and create more room for creativity.

The thesis is based primarily upon interviews with various game audio directors and game audio professionals. The reason for this is the value of expertise on the subject these individuals have been able to provide us and the limited pre-existent material available on the subject.

Acknowledgments

We would like to express our sincerest gratitude to Ben Minto, Gustav Rathsman, Guy Somberg, Jonas Breum Jensen, Lydia Andrew, Rob Bridgett and Stephen Hodde. Without your contribution this thesis wouldn't have been possible.

Bibliography

Bridgett, Rob, 2019: *100 unusual, novel and surprising way to become a better sound designer in video games,* Rob Bridgett

Kastbauer, Damian. 2016: *Game Audio Tales of a Technical Sound Designer Volume 1*, Damian Kastbauer

Somberg, Guy, 2017: *Game Audio Programming: Principle and Practices*, London: CRC Press, Taylor & Francis Group.

Product Appendix

Product Appendix 1: Seamless Loop Shortcut

Video example of tool

https://www.youtube.com/watch?v=Ys7NKG2gPbY

Product Appendix 2: Reaper to Wwise system

Video example of tool

https://www.youtube.com/watch?v=cGZRgcZvnLk

Product Appendix 3: ReaOpen system

Video example of tool

https://www.youtube.com/watch?v=jLkoypSyK5Y

Product Appendix 4: WwiseType System

Video example of tool

https://www.youtube.com/watch?v=Pd4eYPv1G4M

Product Appendix 5: Custom Third-Person Listener

Video example of tool

https://www.youtube.com/watch?v=myuOvg6ibag

Product Appendix 6: Wwise Material Switch

Video example of tool

https://www.youtube.com/watch?v=i7bxsXQr21g

Product Appendix 7: Wwise Ambience Follower

Video example of tool

https://www.youtube.com/watch?v=7e6MXlNjiHI

Product Appendix 8: Wwise Sound Painter

Video example of tool

https://www.youtube.com/watch?v=65MO97XnWb0

Product Appendix 9: Wwise Animation

Video example of tool

https://www.youtube.com/watch?v=9AEcd0Qu1b4

Product Appendix 10: Playthrough of product

Video playthrough of case study

https://www.youtube.com/watch?v=-xCvy3IjzGk

Appendix

Appendix 1: Interview with Stephen Hodde

Stephen Hodde, Audio Director at PolyArc

This file isn't publicly available.

Appendix 2: Interview with Rob Bridgett

Rob Bridget, Audio Director at EIDOS Montreal

This file isn't publicly available.

Appendix 3: Interview with DICE

Ben Minto and Gustav Rathsman, Audio Director and Sound Designer at DICE.

This file isn't publicly available.

Appendix 4: Interview with Lydia Andrew

Lydia Andrews, Audio Director at Ubisoft Quebec City Studio.

This file isn't publicly available.

Appendix 5: Interview with Guy Somberg

Guy Somberg, Lead & Audio Programmer at Echtra.

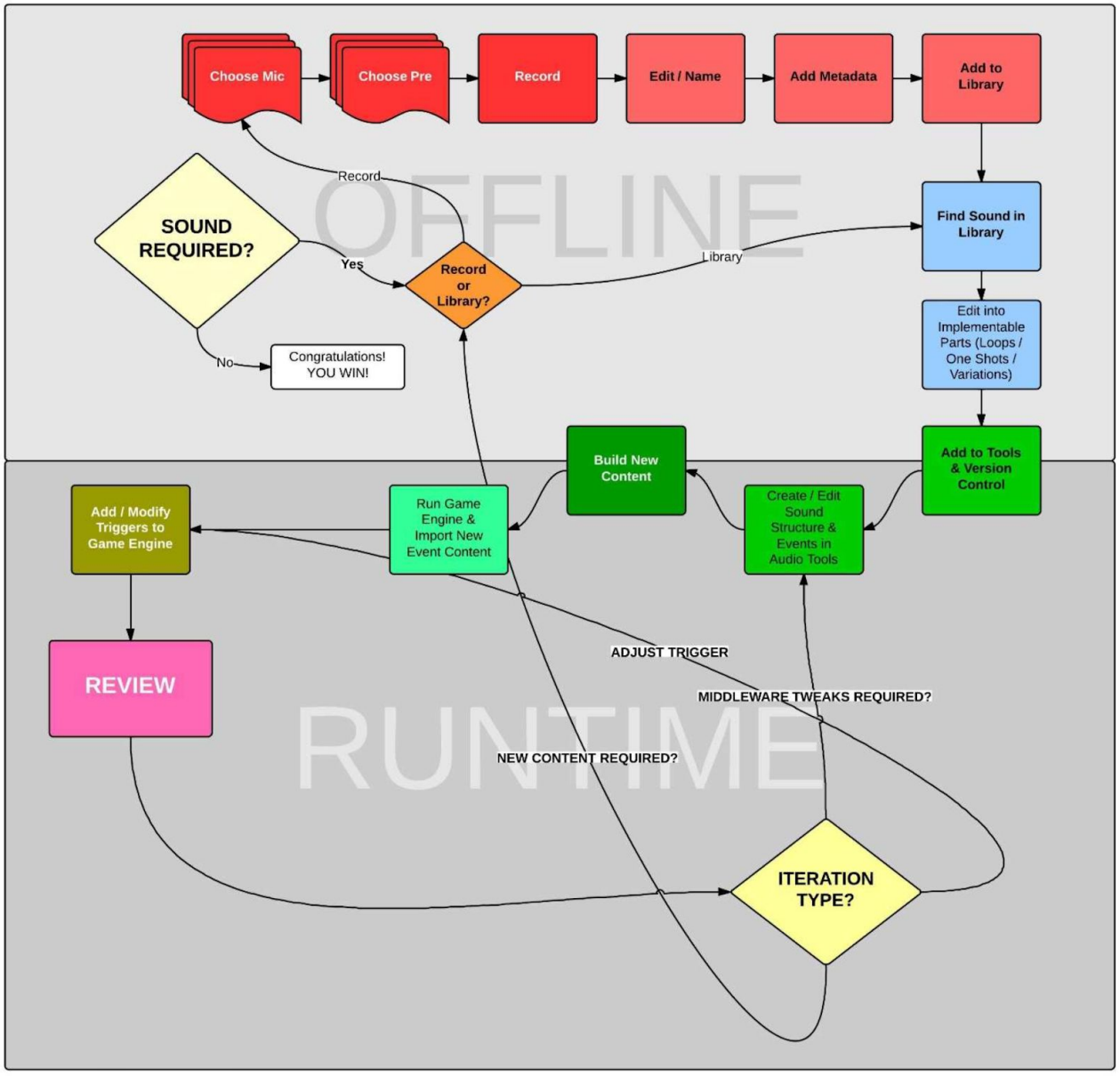
This file isn't publicly available.

Appendix 6: Interview with Jonas Breum Jensen

Jonas Breum Jensen, Lead and Principal Sound Designer at IO Interactive.

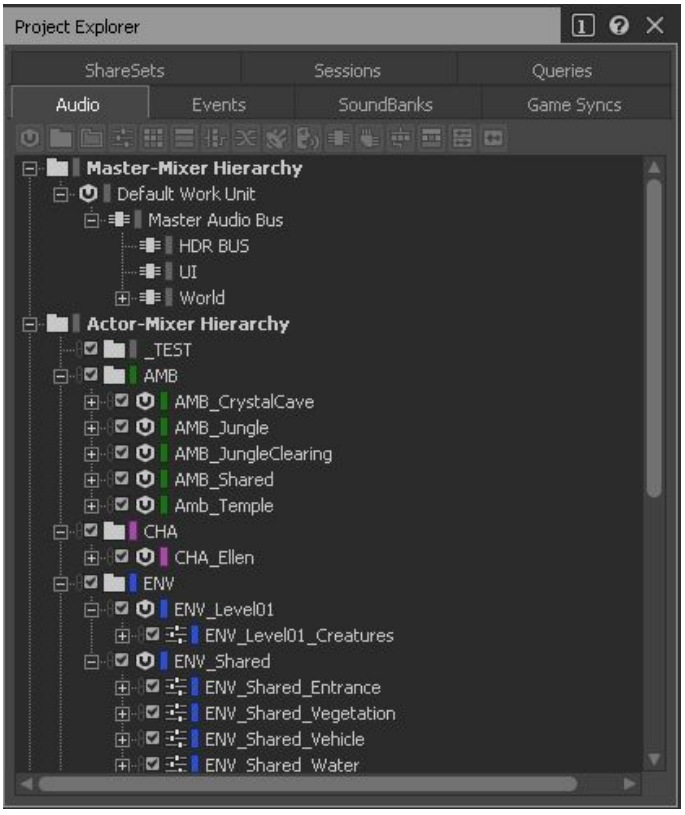
This file isn't publicly available.

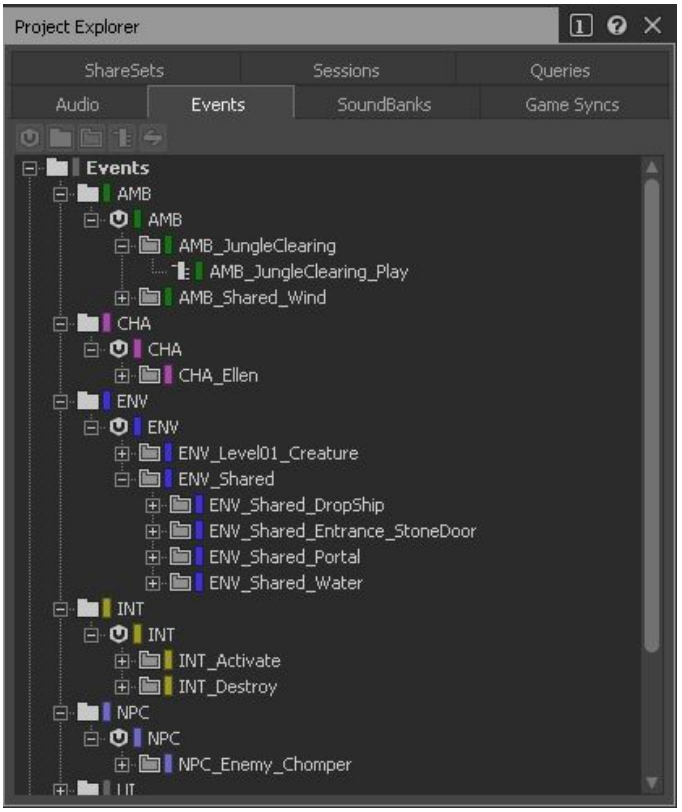
Appendix 7: Illustrated pipeline process



From Bridgett, Rob, 2019: *100 unusual, novel and surprising way to become a better sound designer in video games,* p. 59

Appendix 8: Example of Wwise structure



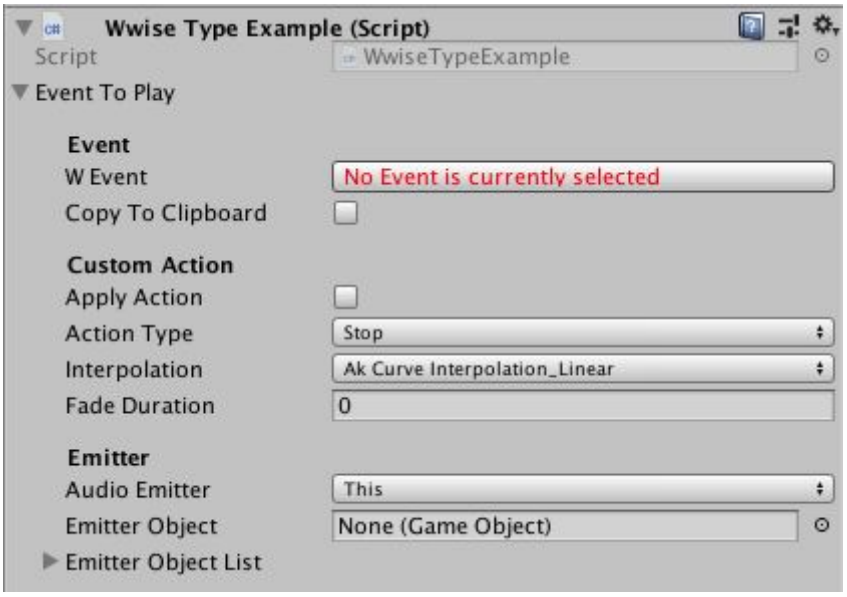


Appendix 9.1: WwiseType System

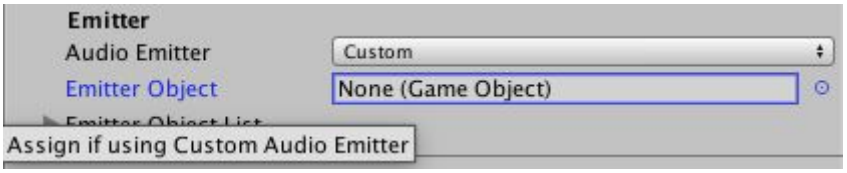
```
public class WwiseTypeExample : MonoBehaviour
{
    public WwiseEvent eventToPlay;

    void MyMethod()
    {
        eventToPlay.Post(this.gameObject);
    }
}
```

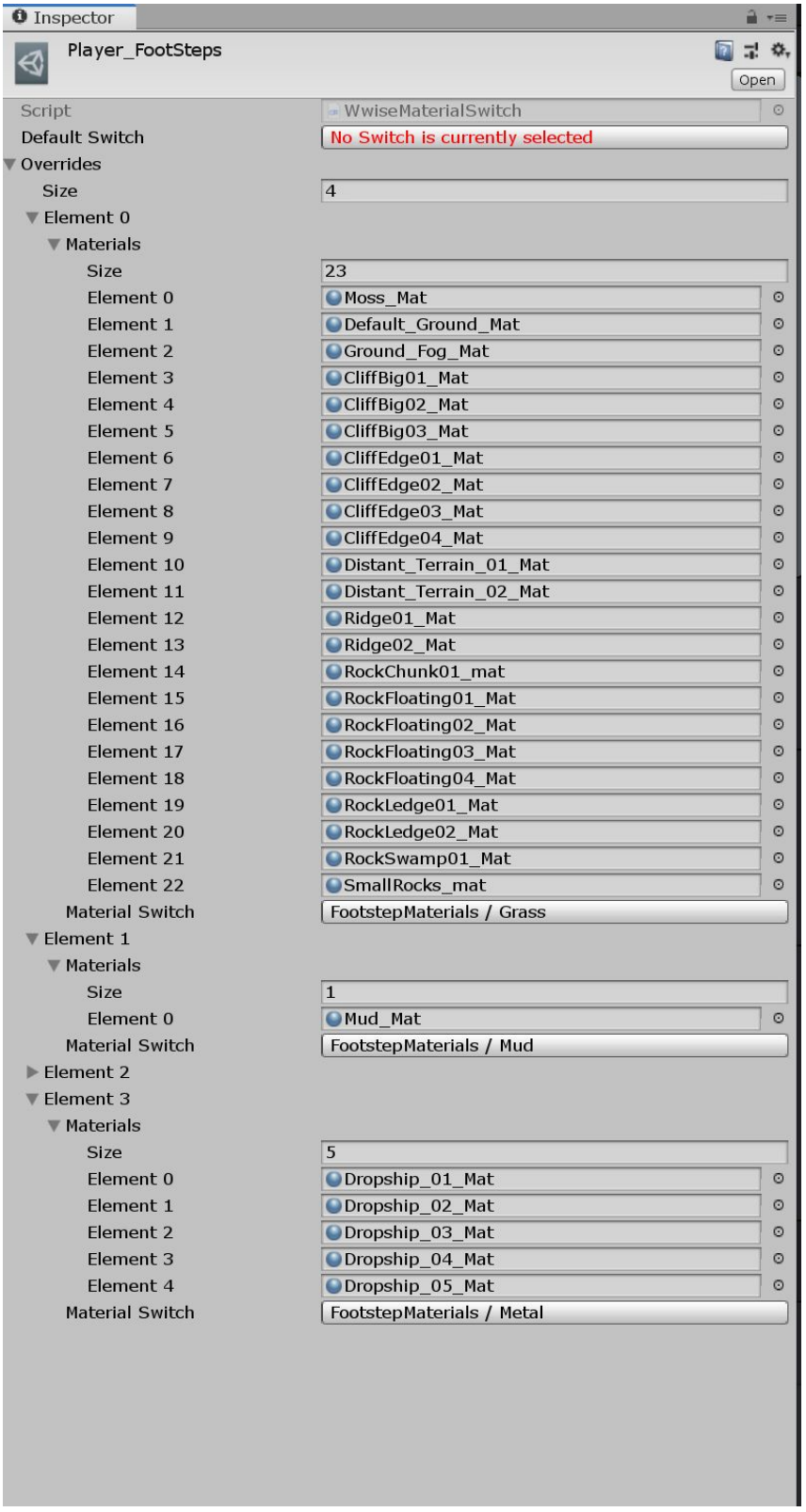
Appendix 9.2: WwiseType System



Appendix 9.3: WwiseType System



Appendix 10.1: Wwise Material Switch



Appendix 10.2: Wwise Material Switch

```
1 usage  Jeppe Lindskov
public void Initialize()
{
    for (int i = 0; i < overrides.Length; i++)
    {
        foreach (var material in overrides[i].materials)
            m_Lookup[material] = overrides[i].materialSwitch;
    }
}
```

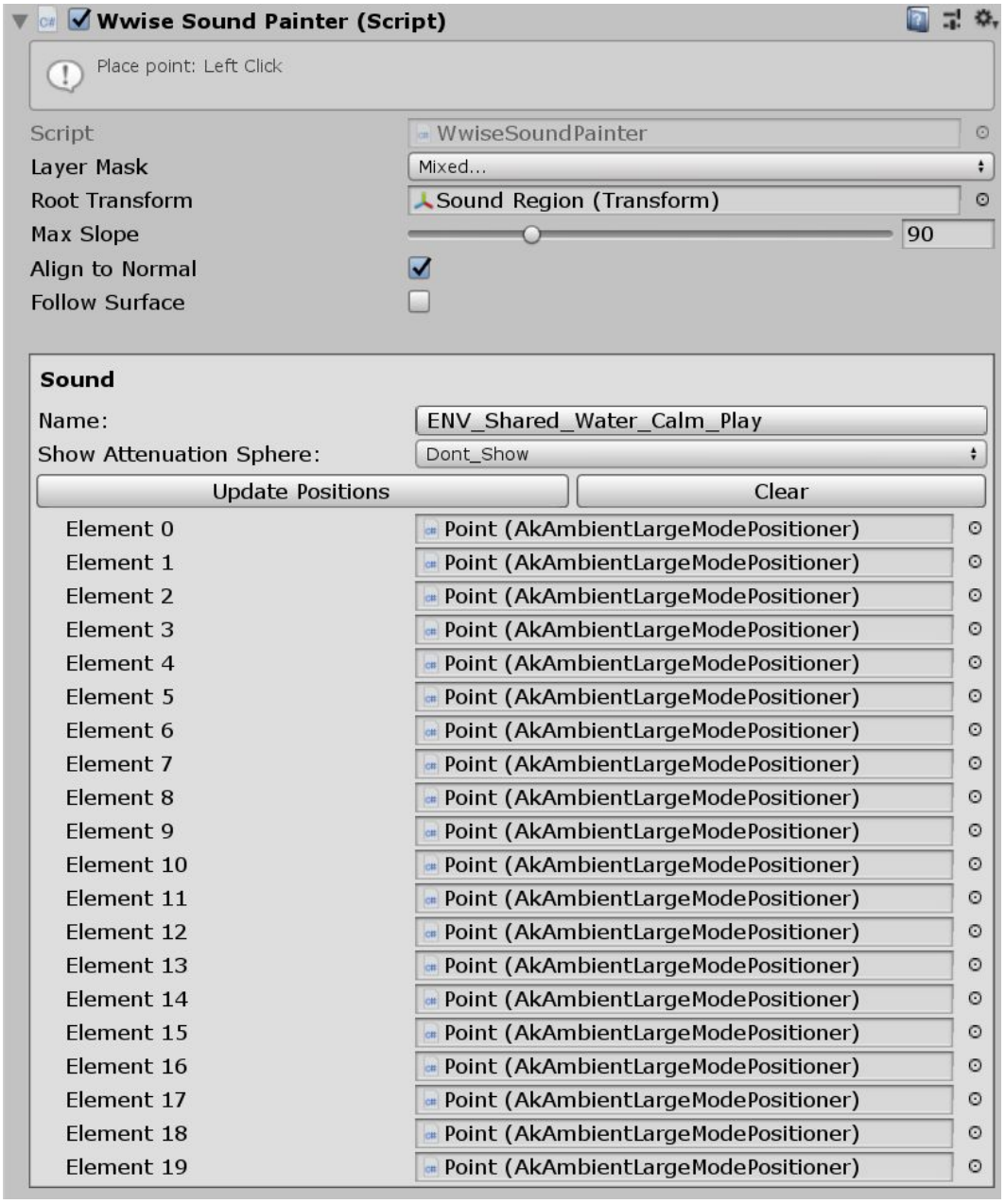
Appendix 10.3: Wwise Material Switch

```
Frequently called  3 usages  Jeppe Lindskov
public void CheckMaterial(Material surfaceMaterial, GameObject gameObject)
{
    if (!surfaceMaterial)
    {
        return;
    }
    if (!m_Lookup.ContainsKey(surfaceMaterial))
    {
        return;
    }

    if (m_Lookup.TryGetValue(surfaceMaterial, out switchToSet ))
    {
        if (oldSwitch != switchToSet)
        {
            if (switchToSet != null)
            {
                switchToSet.SetValue(gameObject);
                Debug.Log(switchToSet.Name);
            }
            else
            {
                switchToSet = defaultSwitch;
                switchToSet.SetValue(gameObject);
            }
            oldSwitch = switchToSet;
        }
    }
}
```

Appendix 11: Wwise Ambience Follower

Appendix 12: Wwise Sound Painter



Appendix 13: Wwise Animation

```
public void WwiseAnimEvent(string eventName)
{
    if (disableEvents) return;

    WwiseEvent wwiseEvent = wEvents.FirstOrDefault(x => x.wEvent.Name == eventName);
    if (wwiseEvent != null)
    {
        wwiseEvent.Post(this.gameObject);
    }
}
```