How to optimize the implementation process and data management in game audio production

Master's thesis - Aalto University
Mikko Kolehmainen

Abstract

While video games are getting more complex, the amount of required audio content keeps increasing. Copious amounts of audio files are usually implemented manually and require consistent data management. This master's thesis is a study on the challenges typically encountered in large-scale game audio productions, where the often preferred linear workflows lead into new challenges, and whether these challenges could be solved with new tools such as custom extensions for middleware. Through detailed description of development and functionality of an example solution, the possible ways of using automation to optimize the audio implementation process and data management are shown. Finally, future development is discussed based on findings.

Tiivistelmä

Kun videopelit kehittyvät laajemmiksi ja monimutkaisemmiksi, tarvittavan äänisisällön tarve kasvaa. Valtavat määrät äänitiedostoja täytyy manuaalisesti integroida peliin, sekä datan hallinta vaatii tehokkaat työkalut. Työ on katsaus tyypillisiin videopeli-äänituotannon haasteisiin isoissa produktioissa, usein suosittujen lineaaristen työtapojen aiheuttamiin ongelmiin sekä niiden mahdollisiin ratkaisuihin mukautettujen sovellusten avulla. Työ esittelee esimerkin hyödyntää automaatiota datan hallinnan ja integroinnin optimoimiseksi käyden läpi esimerkkisovelluksen kehityksen ja toiminnallisuuden. Lopuksi ajatuksia jatkokehityksestä ja peliäänen tulevaisuudesta löytöjen pohjalta.

⚈ **Author**: Mikko Kolehmainen

⚈ **Title of thesis**: How to optimize implementation process and data management of game audio production

⚈ **Department**: Media ⚈ **Degree programme**: Sound in new media

⚈ **Year**: 2020 ⚈ **Number of pages**: 46 ⚈ **Language**: English

⚈ **Keywords**: game audio, implementation, data management, automation, Wwise, software development, workflow, pipeline

Contents

Introduction

This master's thesis is a study based on my observations on the development of game audio production workflows in the context of large scale video games. As video games are getting increasingly more complex, the previous ways of working are no longer optimal, and handling huge amounts of data creates new challenges. This study focuses on the conflicts of combining linear and non-linear workflows and attempts to assess how such conflicts could be solved. While new techniques to optimize digital work, such as automation, are still quite rarely used in the field of game audio, this thesis demonstrates possible ways to benefit from these new methods.

The thesis is created from the love of optimization and problem solving. The aim is not to give answers but rather to raise questions; questions which I've asked myself while working as an audio designer, technical audio designer and developer. These questions and the proposed solutions are largely based on my subjective observations. This study is mainly targeted to audio professionals with a good understanding of game audio production as a whole but also about its technical details, challenges and typical workflows.

While game audio productions are complex and full of problems to solve, this text focuses mostly on the technical side of content heavy productions such as the implementation and data management, but also covers some problems arising from problematic mindsets.

Thesis statement

While historically audio in games used to be real-time synthesis, closely attached to the game engine and coded by the programmers, the modern game audio has developed into a complex process of handling massive amounts of linear content created by a group of audio professionals. The increases in processing power and available memory have led to high expectations and ambitious goals to create highly realistic game worlds with countless sound effects and dynamic orchestral scores. While the quality and amount of required

audio content has drastically increased, the gap between the content creation and the implementation has gotten wider. Simultaneously, this has created a need for more advanced tools and middlewares to achieve the tight, interactive symbiosis of audio and game engine we used to have. Through observations, questions and an example project, this thesis is trying to answer the question: How to optimize the implementation process and data management of game audio in the context of large video game productions?

HISTORY & CONTEXT

In this chapter, the history and technical development of game audio will be briefly covered. Also middlewares will be discussed and their typical use cases and features explained.

Brief history of game audio

Early video games had no sound, and for a long time sound wasn't really even considered being an important part of games. During the time of early slot machines and arcade games, sounds were only seen functional, like Collins describes how sounds in early coin-operated arcade games were marketed if they were only to attract people during the time machines were not played (2008, p. 9). After a successful arcade game *Pong* (1972) made by Atari, presented how even a simple beep when a ball hits the paddle can add to the gaming experience, the idea of sound enhancing the gameplay started to become more mainstream. While sounds were first used only for the purpose of sound effects also musical ideas started to emerge, one of the earliest examples being *Space Invaders* (1978) made by Midway.

Implementing sounds for the early platforms was demanding due to very low processing power and small amount of available memory. Trying to fit even just the game itself into the small memory footprint proved to be a lot of work. Due to limitations, sounds were not high priority and often they would be created by the programmers instead of separate audio professionals. Even during the 1980s, when most of the arcade manufacturers had dedicated technology for producing sounds such as programmable sound generators and digital-to-analog converters, creating sounds was very technical. With limited memory, the solution was to program the sounds to be generated real-time with a variety of synthesis methods. Through the years lots of progress happened and new home consoles such as *Video Computer* System by Atari and *Intellivision* by Intelligent Television pushed development further. With their dedicated sound chips and ways to program sound effects and music, new systems with audio sampling methods and polyphonic voice support or

even ways to convert Musical Instrument Digital Interface (MIDI) notated music into code (Collins, 2008, pp. 9-24).

For a long time music in the games was mostly a collection of fairly short loops which would be played with certain logic. In the same manner the sound effects would be short and simple, effective yet memory efficient. Having a very small memory budget really pushed composers and audio designers to come up with creative solutions to create coherent soundtracks for the games. Naturally in many cases the only option to have sounds and music covering longer time of gameplay was to repeat the same sounds all over again.

Development from 8-bit and 16-bit to 32-bit, 64-bit and 128-bit systems granted more advanced ways to use sound. Technologies such as support for positioning a sound in three dimensional space and possibility to use real-time effects, moving from cartridges to CD-ROMs allowed game audio to become higher in quality and move from real-time synthesis to sample based solutions. One major upgrade was also when sound processors started to have their own memory such as in Sega Dreamcast. Through use of DVDs, Playstation 2 made a serious leap forward by supporting also surround sound formats. At this point it was clear, that sound effects and music were seen as an essential part of the video games and their potential was more cultivated. Sound was not purely functional anymore, but seen as fundamental component to support storytelling, contribute to world building and build sonic brand of the game (Collins, 2008, pp. 63-73).

Over the years games, and the audio teams working on them, have gotten larger. For example producing music for a game often involves much more people than before, earlier repetitive beeps created by one person can be now replaced by a full blown non-linear orchestral scores composed, arranged, produced, played, recorded, edited, mixed and implemented by highly specialized professionals. Before, audio production was limited by memory and technical aspects, now the limitation is often just imagination and time. And therefore money.

Non-linear nature of game audio

The description of the nature of game audio is easier through a comparison with something fairly similar, such as audio for film. One major difference between the game audio and the film audio is the linearity of the medium. A film is linear medium in the sense that it is consumed over time by watching it through some sort of screen or projection. Even though the definition of the time in the film isn't necessarily perceived very linearly, the film as a medium is still linear. At the same time, what makes video games non-linear is the factor of interaction, which isn't the case with the majority of the films. Since the interaction usually gives the player the freedom to decide what to do, where to go, how fast to approach in the game, it makes video games non-linear.

While the film audio has a mature and quite fixed workflows developed over decades, this isn't the case for game audio and productions can be quite chaotic. In game development the whole organization and planning of the production can differ considerably between companies. While the game development is complex organism bursting in different directions it also affects the audio work and need for flexible workflows and proper tools to support that. For both film audio and game audio, the tools for content creation work have been developing a lot, digital audio workstations (DAW) and audio plug-ins provide more and more possibilities to create extremely high quality audio design. While being powerful, these tools are designed for linear audio work, which in case of non-linear video games separates the process of content creation and implementation to two very different tasks.

On the high level, the game development work can be split into two main categories: code and content work. Code is the basis of running the game and its functionality, but also the means to handling multiple types of data, the content of all different disciplines. While before the audio in games was programmed and often generated according to the events of the game, making it part of coding work, now the creation of sounds is often seen as an external process. Since these tasks are separated and require a different set of skills, they're often assigned to different people. This creates different roles such as audio designers and technical audio designers. Audio designers are typically responsible for the content creation whereas technical audio designers focus on implementation and

developing tools and pipelines. Having multiple roles makes maintaining a good connection between the content, implementation and programming challenging.

Audio middlewares

Working with non-linear medium requires the right mindset, but also proper tools to support the workflow. While the amount of content, in this case audio files, has increased, new ways to manage and process the content has to be developed. When the creation of the content is happening as an external process, also its management has been separated from the game engine. This has led to the creation of audio middlewares.

Audio middleware is separate software integrated to a game engine. It provides a set of tools to help handling large amounts of audio files and to create connections with the game engine in order dynamically process the content. Advanced middleware with rich feature set grants access for audio designers to do more on their own without bothering technical audio designers or programmers with simple implementation tasks. Since audio middleware often includes at least basic audio processing tools and possibility to create audio events including complex logic how to play the audio back, it does not only allow audio designers to create more finished content, but it's also the essential tool set for creating truly interactive and dynamic audio design.

In order to play audio from the game, some sort of audio engine is needed and often developers will prefer to use an external middleware to separate the content creation work from the engine related work. On the other hand, in the case of custom game engine, the audio engine functionality can be built into the game engine allowing handling of audio events, memory management and conversion of the assets to be more optimized and more closely in connection with the visual assets, parameters of the game and features of the game engine. This kind of custom solutions usually also require some sort of audio editor framework with functionality, completely up to developers. Even a simple audio editor will naturally need maintaining and regular updates, not to mention new features.

Like Horowitz and Looney conclude in their book, using middleware doesn't only give audio designers more control over their work, but also more confidence that the audio will sound in the game exactly like they wanted it to. In addition to this programmers don't have to spend so much time and worry about the audio in the game. It's "an all-round win-win situation", authors are stating (2014, "Level 8").

While audio middleware certainly brings lots of functionality into the hands of audio designers and helps with implementation and management of audio assets, it also creates another gap between the middleware and the game engine. Integration of the middleware with the game engine is an integral part of the whole game audio pipeline, but in this thesis we will focus on the challenges when content creation is being too separated from implementation.

Choice of a middleware

At the time of writing, there are multiple audio middleware options on the market, with Firelight Technologies FMOD and Audiokinetic's Wwise being the most frequently mentioned ones.

Choice of middleware is mostly a matter of taste, scale of the project, budget, supported platforms and complexity but also flexibility of integration with the game engine. Most of the middlewares offer quite similar features for handling assets and the logic, but there are differences on available audio processing plugins and extensions, usability, how the implementation workflow works, which platforms are supported and what kind of feature set is provided for optimization per platform. A major factor to take into account is the availability of application programming interface (API), its flexibility and features. Scalable and powerful API allows audio programmers and technical audio designers to create custom tools and extensions for the middleware for instance to benefit the content creation work or develop advanced solutions for optimization of memory and event handling with the game engine.

I've ended up working a lot with Wwise which is audio middleware software developed by Audiokinetic. It's primarily focused on non-linear and interactive audio work such as game audio. I've chosen Wwise to be the reference for the needs of this thesis, with its terminology and basic features I'll cover the typical parameters of the implementation process. One of the Wwise's best features is its authoring API (Rodrigue, 2017, p. 1) which allows creating many kinds of extensions and integrating Wwise with other softwares easily.

Typical parameters in implementation

In context of game audio production, the implementation means the process of adding the created audio file, the asset, into a middleware or game engine and defining when it is played back and how. In this chapter we will cover the basic parameters in implementation at a very general level. Even though we will use the Wwise as an example middleware, the same basic functionality can be found in any commercially available middleware.

The playback related parameters presented in Wwise may be split into main categories such as general settings, conversion, effects, positioning, and advanced settings like settings for playback limiting.

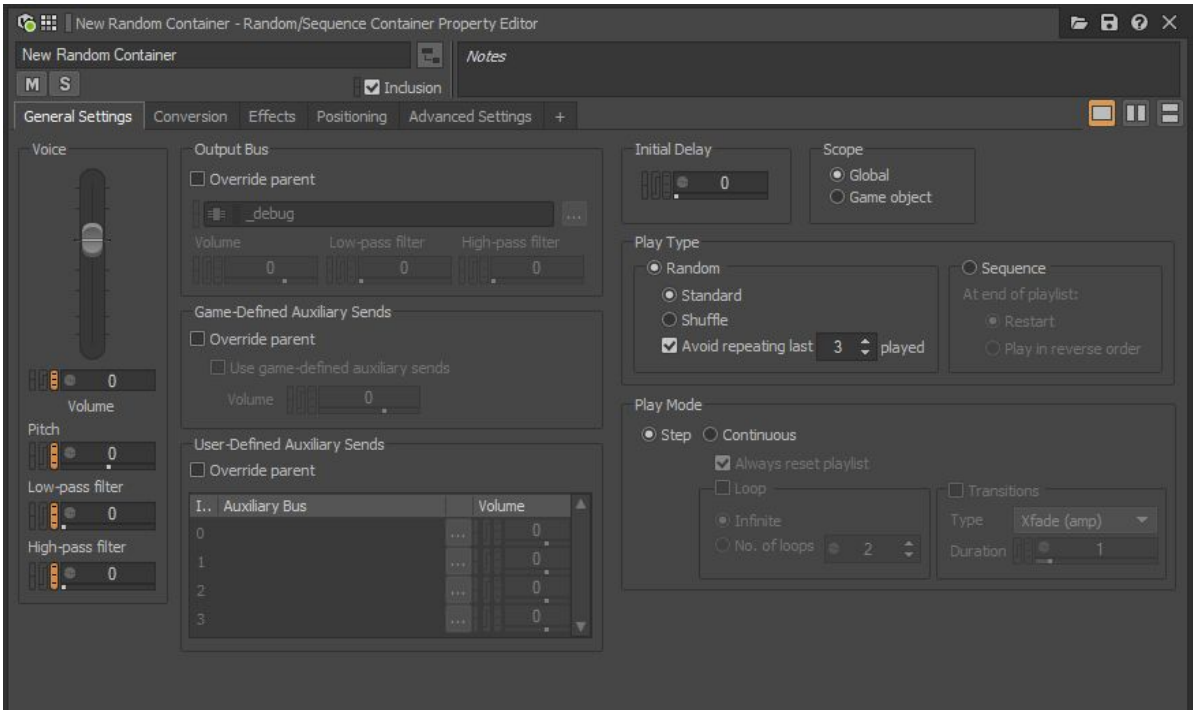

Figure 1. General settings view in Wwise. Screen captured by the author.

General settings are covering the conventional settings for audio assets usually found in any other audio related software, it includes parameters such as low-pass and high-pass filters, pitch and playback volume of an asset. Some of the functionality which comes with non-linear medium is to be able to define randomization to these parameters within a certain range every time the sound is going to be triggered. Depending on the type of a Wwise object, general settings may also include settings to define whether the asset is going to be looped and with what kind of logic.

Conversion settings allow the audio codec and audio quality to be defined per sound. This is essential functionality in order to optimize memory usage and central processing unit (CPU) usage of audio assets in the game.

Effects section allows a variety of audio effects to be applied to the asset. The effect parameters can be controlled in real time according to the information coming from the game or modulated for example with low frequency oscillators. Processing in real-time

isn't an only option, processing can be rendered to the asset which is good practice to save CPU usage.

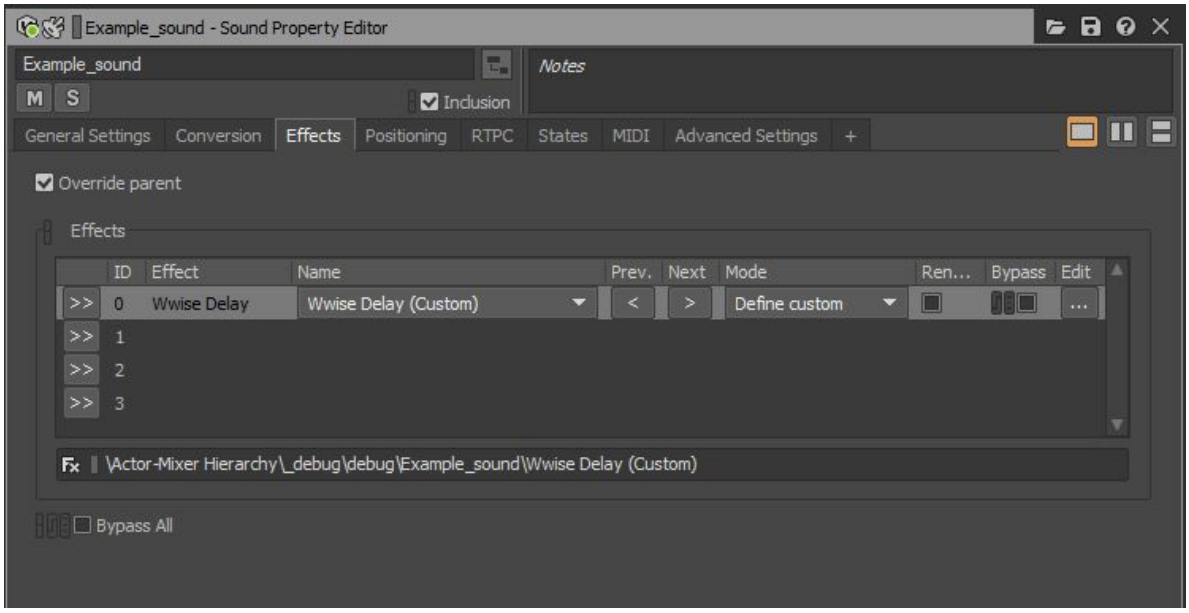

Figure 2. Effects settings in Wwise. Screen captured by the author.

Positioning settings are to define whether the asset is going to be three dimensional or two dimensional. When being three dimensional, the asset attached to the game object in the game world will be spatialized accordingly, so when being close to that object the sound can be heard coming from that object. Two dimensional sound would not be localized in the game world but playing back untreated as normal stereo, not affected by location of objects.

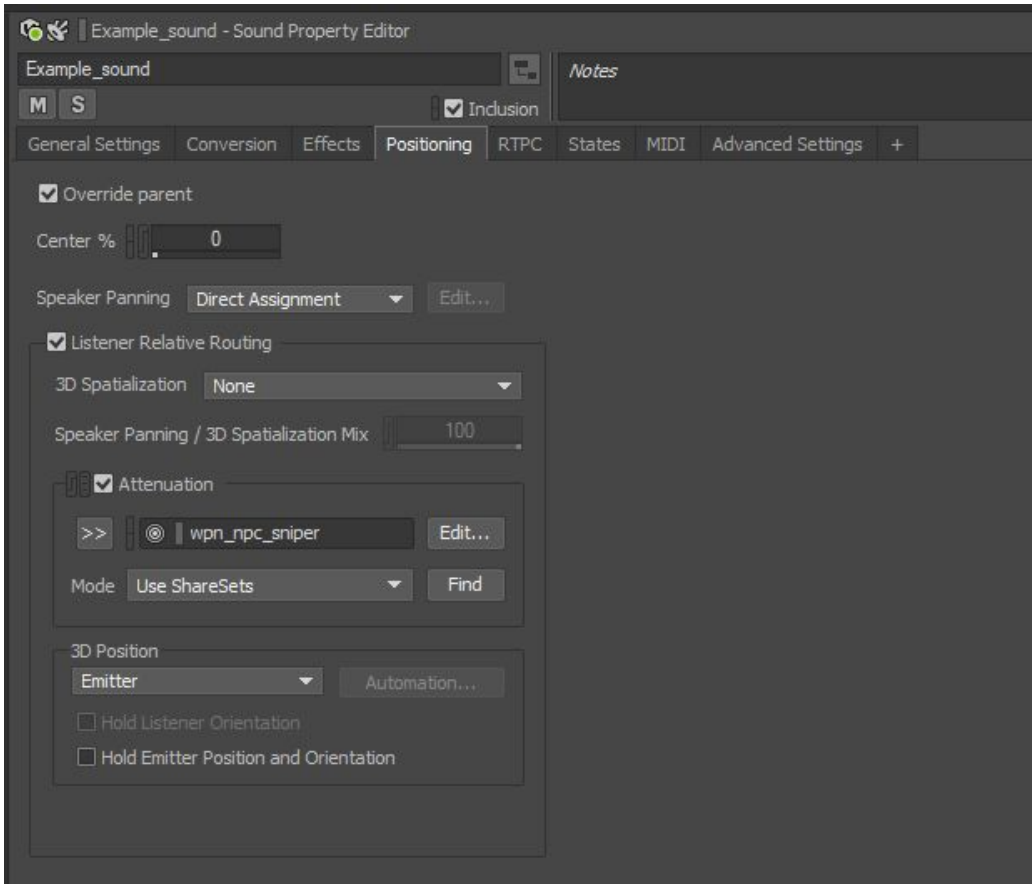

Figure 3. Position settings view in Wwise. Screen captured by the author.

Advanced settings including playback limiting is a very useful category of parameters which allows to define playback settings of the sound within the larger context. By defining priority and limiting settings it's possible to control the logic how assets will be handled in case too many of them are playing at the same time. Limiting can stop playing assets or put them "on hold" (virtualized, in Wwise terms) until the overall amount of assets is within the limits. This is a very important part of optimizing the audio performance of the game.

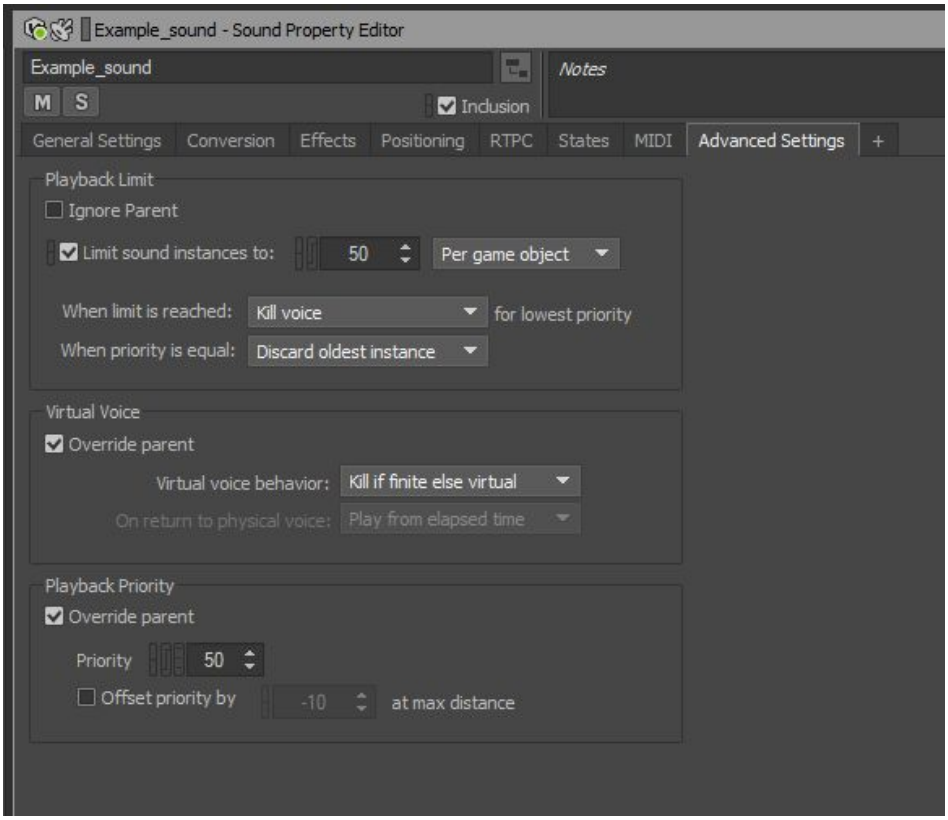

Figure 4. Advanced settings view in Wwise. Screen captured by the author.

In Wwise, you create events which can include different kinds of actions. For instance starting to play asset, stopping already playing asset, setting or resetting parameters inside Wwise or bypassing effects on some assets. These events can be very simple, for example only including one action to start the playback of the asset. On the other hand, one event can include multiple actions and include fairly complex logic structures such as starting playback of assets, changing effects parameters over time and after a while fading out the volume of the asset and stopping the playback. In the case of Wwise, these events are the link between the middleware and the game engine code. In short, Wwise creates "soundbanks" including the needed assets in specific format and the events per platform which are then included to the built game code. Then these events can be called from the game code or different Wwise parameters can be altered when needed.

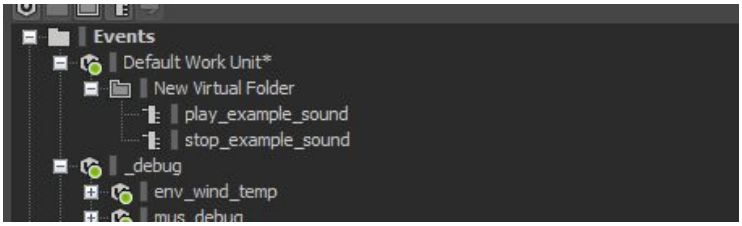

Figure 5. Event hierarchy structure in Wwise. Screen captured by the author.

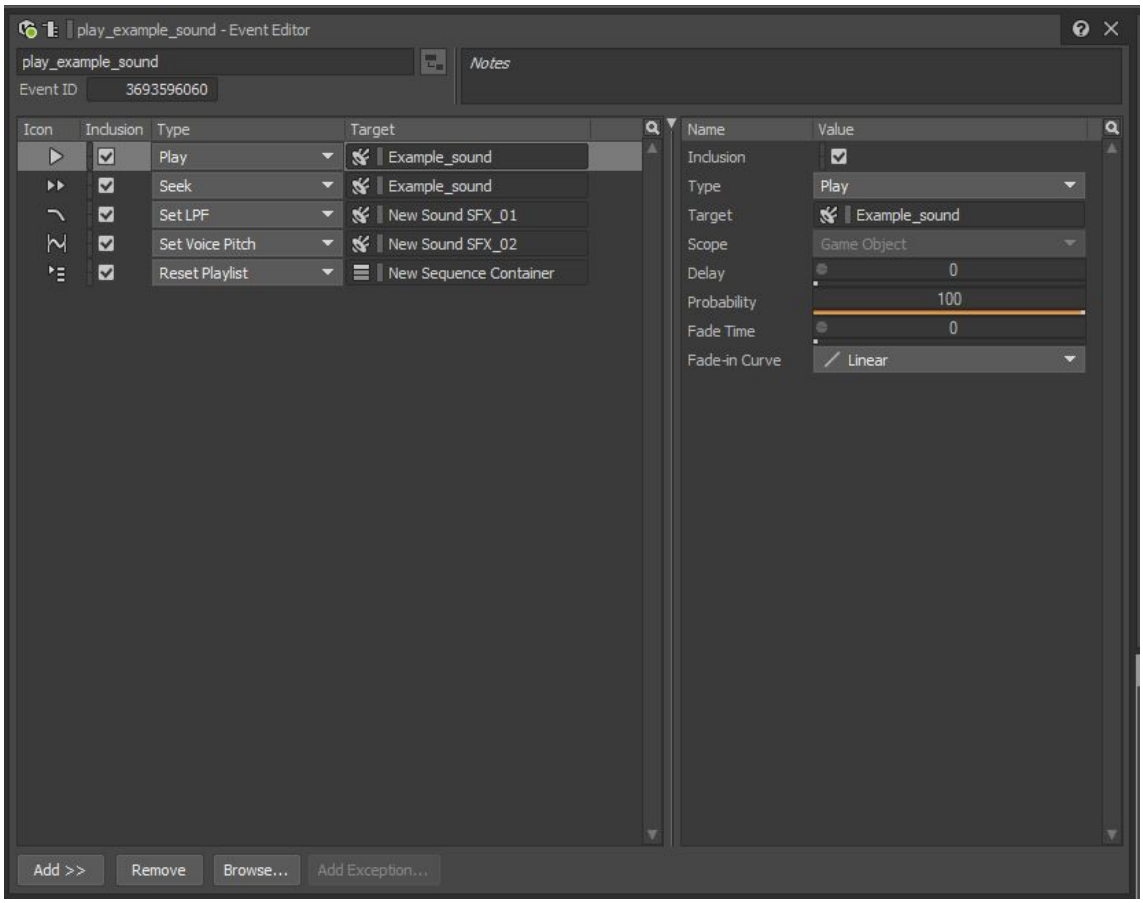

Figure 6. Event editor view in Wwise. Screen captured by the author.

This is one reason why using middleware can be very beneficial, only one event call from game code can trigger multiple actions at once. Programming all this functionality only within game engine code would be a much more complex task.

CONFLICTS & CHALLENGES

In this chapter I will point out my observations and describe how the influence of the film industry, linear tools and development of separate processes and roles are creating challenges in the game audio production.

Games are getting larger

Game audio work is hard. Modern games require large amounts of content to fill the game worlds, give a sense of space, represent vast variety of actions and different gameplay mechanics. As Marks (2012) is describing, there's an "interesting trend" of game audio productions getting so much more complex than before that it requires a full team of "composers, sound designers, voice-over directors, audio editors, engineers, and mixers, to name just a few" to be able to manage the projects. To keep the quality high enough, smaller independent developers need to team up with other developers to meet the goals. He sees this as positive news, since it creates possible new careers for the game audio field (chapter one).

The amount of work is one of the main challenges when it comes to game audio. Often the workload needs to be planned to fit into strict time limits while compromising the quality of the assets and dynamics of implementation. Since game audio design is a highly iterative process, losing the time meant for iterating and polishing has a huge impact on the end results. Optimally the given time for the production should allow the audio team to create results they can be proud of, but this isn't always the case. Scoping out, meaning not doing something which was already planned is often inevitable for some parts of the work. Often this is a matter of good planning and prioritization. Not all sounds in the game are equal. Some of the sounds might be played very often or they have a very important role in the game, while some are in the background or not even heard often. In well organized audio production priorities should be clear and time spent per asset somewhat correlating to their level of importance in the game. If overall workflow of the production and pipelines

per team aren't well designed it can make prioritization very difficult. This often leads to situations where too much time is used for creating and implementing certain assets, which don't have an important role, or aren't even used, in the finished game.

As memory budgets are increasing and the level of computing power raises, also expectations of the players for the quality and level of detail in games increases. Since there's less technical limits, it means more possibilities also for audio. It's not only about audio quality getting better and capability to play more sounds at once, but also being able to create highly dynamic and interactive audio design. When also taking into account how different ways to get information from the player are developing further, all this provides more and more possibilities. Naturally making use of these new technologies isn't necessarily easy and often requires designing new pipelines and tools which is time consuming and requires a lot of work. Unfortunately without good planning early enough it's easy to end up in a situation where huge amounts of content is already in the game and new technologies or implementation ideas are not possible since it would mean redoing a lot of work which isn't possible because of tight schedule.

While workloads can be difficult to manage, It's often also a side effect created by badly optimized workflows and nonideal tools. Game development is a non-linear process, which impacts the amount of work and especially how it is structured, but this isn't necessarily addressed when designing the audio production pipelines.

Role of the audio designer

Audio designer is usually the person responsible for the actual content creation work, this is an essential role of game audio design. In smaller companies, often only one person handles all the audio related tasks and will usually be called an audio designer. The role of audio designer is often very flexible and in many cases can include whatever is needed for the audio side of the game production.

In short the typical craft of audio design is the process of creating the content, a single audio file or collection of audio files which fits and serves some purpose in certain context. These files are then implemented so that their wanted playback logic and possible variations and/or post-processing is supporting the original vision of the audio designer.

The craft of audio design takes a lot of time to master and this naturally makes the masters proud of their work. From my experience this can also create an attitude where the actual work and methods are mystified and kept secret. This kind of attitude has one major side effect: it blocks more open and general discussion about the workflows and methods. Without shared knowledge and public discussion the progress of development tends to get slower and scattered. Since development isn't going to happen everywhere equally, open discussion and questioning of workflows is needed to inspire everyone and start more general development of workflows and pipelines.

When it comes to audio designers, we often tend to talk about artistic people whose workflow has a lot to do with emotions and being inspired, but not necessarily how to work efficiently. From my experience, technical requirements of the game audio designer role are often ignored despite the fact that the work is executed and connected to highly technical counterparts. By technical requirements I don't only mean to be able to use game engines, script in logical and readable manner and have the understanding of other aspects of the game, but also being able to see the possible ways to develop the work. Also if the technical part of the work is ignored, it usually means someone else needs to take care of those tasks. It’s not easy to convince people to change their workflow if the motivation doesn’t come naturally, even if the new workflow suggestion would be rationally much better than the current workflow. In my opinion, the game audio design work shouldn’t be only about creating the assets and implementing them but creating new more systematic ways how to use assets and allow workflows to go towards more dynamic and interactive audio. Why? Because it's doable, it's a unique feature of game audio and I think, in many ways it’s really needed.

Over the years many design tools such as different plugins and digital audio workstations (DAWs) have developed in a good phase. These tools are mainly designed for linear content creation work, exporting out linear audio files. Audio files are clips of fixed data, rarely able to serve multiple purposes in the game without processing them with the help of middleware. While modern games can be very large, only using linear audio files for content is one of the main reasons why workloads can become unmanageable.

While audio design as a craft has developed the implementation work hasn't been developing on a similar level. Middlewares are getting better and allow audio designers to implement their content with a lot of possibilities, but also create and modify content in real-time with synthesis methods such as granular synthesis. This allows using the same content in multiple places with dynamic modifications, meaning less work for the same amount of covered content in the game. Yet still middlewares are often used to only playback linear audio files. From my experience this is mostly because in many occasions implementation isn't considered being part of the design work. In the case of game audio this is extremely misleading. Implementation has a crucial role on how the assets are going to sound like in the end, without proper implementation it won't really matter how good the original asset was.

In order to game audio develop further, the concept of asset and therefore the role of the audio designer needs to be rethinked. Instead of seeing content creation and implementation as separate steps we should try to combine these two into more flexible and dynamic entities. Instead of audio designers only creating linear audio files more systemic thinking should be applied. While processing power has increased the possibilities of real-time creation and processing have become more versatile. Changing mindsets and attitudes of audio designers towards more non-linear workflows such as using procedural audio and systemic solutions to reuse assets would really help handling the workloads.

Tools and technical designers

Large scale game productions usually require a full audio team including audio designers, audio programmers and technical audio designers. Technical audio designers in cooperation with audio programmers are usually responsible for complex implementation tasks, but also design of different audio systems such as weather or propagation technology. While these systems serve the audio functionality in the games, also the working culture and audio pipeline, outside of the games, should be developed to satisfy the modern needs.

It is interesting to compare audio roles to the roles in game art development. Like Hayes explains, while game development has been getting more complex and team sizes have increased, it has led to a creation of a new role to narrow the gap between artists and programmers, a technical artist. As Hayes (2008) summarizes the role, “technical artist should be able to design and develop all art pipelines necessary for the game. In this sense, part of the technical artist's role is to be a pipeline and systems architect” (p. 1). Even though the role of technical artist can include different responsibilities and be slightly different between companies it’s still very specific, he creates custom solutions for unique needs and develops the digital content creation (DCC) pipeline further. This tells how the art pipelines are consciously developed to more efficient direction.

While the role of technical audio designer could be an audio version of the technical artist role, they rarely focus on developing DCC pipeline for audio. Like Collins (2008) states, the roles of people can be very flexible when compared to other industries than game development, when explaining the role of audio programmer she notes "in many cases they also play the role of sound designers" (p. 87). Since game audio often lacks clear roles, it also makes the development for more specialized roles and clearly defined pipelines difficult. Interestingly in many companies the amount of people working with audio is much smaller than the number of people working with visuals, which means huge workloads on a small amount of people. To be able to handle these workloads, clear and established audio pipelines is needed, but often the discourse is about crunching through the workload instead of how to avoid the crunch.

When thinking about the tools in the context of game development, there's one particular factor which makes it very different to many other fields; the tradition of creating custom tools for different levels of the production. Some companies may start by creating their own game engine and design the whole production pipeline around the workflow the game engine accepts. However, even without a custom game engine, there are still many levels in the production where custom solutions could be applied to.

Audio middleware is often seen as a solution to make the game audio work more streamlined. With middlewares my primary concern is their usability, feature set for asset processing and content optimization per supported platform, but also the possibilities for customization. While middlewares are full of features they are distinctly designed with specific workflows in mind. When trying to use them more creatively, without some level of customization their usability is often very bad. For instance, when creating logic systems in middlewares for more procedural audio generation, it's often doable but extremely painful to manage and use. Usability is a vital part of fast and efficient workflows, but also a big factor to contribute to the broader development of the tools. While audio middlewares aren't usually very user friendly, also being too close to linear, "timeline based" approach makes them quite uninspiring for truly non-linear audio work. Fortunately many middlewares allow using their API to create custom extensions and extra functionality for specific tasks and needs, and also to improve usability.

While middlewares can be useful, they aren't necessarily the best possible solution for every project. For small projects such as mobile games, benefits of feature heavy middleware can be too complex in relation to the amount of work and needed content. Among other things, middlewares make managing large amounts of content easier. But with modern large scale projects with enormous amounts of content even the most advanced middlewares struggle providing the tools for managing all the content. Essential data management features such as clear monitoring of all content or batch editing of assets are oftentimes nonexistent or badly lacking, especially when it comes to user experience design. Since middlewares are generic third party software, trying to handle every kind of project, they don't automatically serve more specific needs. While customization may be technically possible, not all developers can afford the time and money to do that. One

major problem on the market seems to be lack of competition which has reduced the number of available advanced middlewares.

Communication between disciplines

Game development is a complex process including lots of small iteration loops and testing. Poorly organized production can easily become tedious and frustrating for everyone, leading to employees redoing the same parts of their work multiple times. Naturally this happens if changes in game design are made too late in the production, without proper responsibility and consciousness about its effects on other parts of the production. The complexity of game development mostly comes from the fact, it's split into tiny non-linear disciplines which all require quite different workflow, different tools and most importantly very different time frame.

As Bridgett (2014) sums in his talk “the relationship between Sound, Art and Design is only as strong as the relationship between the Sound Designer, Artist and Designer" (05:18). This is an excellent point, the communication and dynamic between different disciplines of the production is crucial in order to make the production proceed smoothly. This is relevant with any type of production involving people from multiple fields trying to create something together, but also the underlying parameter when thinking about designing better pipelines for any of these fields. Often the first thing is the communication between people, but when we want to improve the pipelines we should also take into consideration how to improve the technical communication between these disciplines. Tools such as finding references of assets in the code base or tracking changes and their effects on all assets in version control should be multidisciplinary since they will help with many tasks. Bad communication between people and tools easily separates these disciplines from each other creating confusion and misunderstandings. Especially in larger game productions this can be costly in terms of time and money. When more gaps are spawned it often means more work and therefore extra people need to be hired to fill these gaps.

In case of any discipline, describing the work to someone else can be very challenging.

Game development includes highly specialized roles and understanding them can be hard without some level of education. This often creates a challenge; how to communicate about tasks to other people who are not as familiar with your tasks. More open and equal discussion and communication between disciplines is needed, first on a level of humans and then systems. Improving the processes to be more tied together will benefit everyone. Especially in case of game development, this is very much possible, since all disciplines are after all working on the same end product.

Outsourcing

Outsourcing some of the work is quite typical for big game projects, but not all tasks are easy to outsource. Especially when it comes to technical audio design, programming, implementation and development of new systems it is quite hard to outsource these very essential parts of the production and the long term development. In many cases however, the work of creating content, or some fairly small specific technical requirements, can fairly easily be outsourced.

Outsourcing is complicated. Workload can become huge; handling all the communication, making sure the outsourcing partner understands the context and vision of the project, defining what to outsource and when and when it should be delivered. With a lot of outsourcing in one company, this workload would be big enough to warrant hiring a dedicated person to manage it.

Outsourcing is always more expensive for the company than in-house work. Marks (2012) describes the problem when audio work needs to be outsourced despite having the in-house team; “the last thing they want to do is pay a contractor to do voice-overs or sound effects when they are already paying you to occupy space” (chapter three) . Even if the cost or the communication with outsourcing partner isn’t causing problems, there’s still the fact that the work isn’t finished after getting the outsourced assets. Someone needs to implement all these assets into the game and depending on the case, it can mean

there's more than 50% of the work still left. In that sense outsourcing isn't really solving the real problems of nonoptimal workflow.

METHODS & RESULTS

In this chapter, I will assess possible solutions for challenges such as ever growing workloads and usability of existing software. The project part of the thesis will be introduced; describing how to use automation to make the process of implementation easier and faster.

Procedural audio

As stated before, the amount of needed content in games is increasing. While content is urgently needed, it's good to question what is the most efficient way to create it? There's one interesting concept and approach for creating more dynamic audio assets with more control over their behavior: procedural audio. Procedural audio is a process where the sound is generated in real-time with code by using different synthesis methods, very much like it used to be in many old games. Farnell (2010) describes this approach in great detail and also concludes the benefits, "Procedural sound is a living sound effect that can run as computer code and be changed in real time according to unpredictable events. The advantage of this for video games is enormous, though it has equally exciting applications for animations and other modern media" (p.1).

In the beginning of his book, Farnell (2010) describes the idea of procedural audio and says: "It's about sound as a process rather than sound as data" (p.1). This is an excellent way of describing the principal difference in the thinking and approach, which these days is often missing from the content creation of game audio. While Farnell is talking about creating all sounds with synthesis, the principles of more systematic audio design could be applied also to sample based content and its implementation. How about creating a system instead of asset? Real-time parameters instead of scripted triggers? Behavior instead of random variations?

Despite the great level of control over the behavior and interactivity, the achieved audio quality of procedural solutions hasn't always been on par with a sample based solutions.

Perhaps the best way would be to come up with a good middle ground between completely synthesized and sample based solutions. As an example, in sample based approach, procedural principles can be still applied to logic how samples are handled. The goal should be to achieve fast and controlled, yet flexible process where the samples and/or logic itself is easily modified and scaled to possible changes in the game. When thinking about the process from the perspective where assets are like entities or prefabs being able to interact with each other and transform themselves according to any specific situation, the design process would be almost like supervising the independent iteration loops. Using a more systematic approach would allow more creative freedom and especially save time. Even if all content couldn't be created that way it would help with prioritizing the content and therefore production time.

Automating the implementation process

It's extremely fascinating how day-to-day workflows have been developed to a very high extent within the field of software development, but many of these principles and methods haven't reached practice of game audio or even many levels of game development. These fields are very different from each other, but they have one major aspect in common. All the work can be done only with a computer if decided so. The digitalization of audio work also allows new methods to emerge, one of them being automation.

Automating simple tasks is getting more popular in many fields of business and art, but haven't been used to a larger extent within the game audio. While content creation is a huge part of game audio work, it's also a complex process and therefore more difficult to try to automate. On the other hand, the implementation is often repetitive, fairly predictable process which with lots of content becomes quite error prone when done manually. Also managing all the data can become very tedious on large game productions.

My method and practical project part of this thesis is a middleware extension for automation, I designed and programmed during the fall of 2019. It was created to help with audio implementation and data management when working in modern, content-heavy

game productions. In the next section, I'll cover its functionality on high-level, how it's used, how it can improve the workflow and which problems it's supposed to solve.

Scriptable automation layer

While working on massive game production I recognized how easily humans executing repetitive tasks will lead to mistakes but also how convoluted the management of all content and data becomes. Since the used middleware; Wwise, didn't offer the functionality I wanted, I started designing a system to expand the functionality of it. I wanted to create a system to support an automated implementation process with better usability compared to manual implementation. I also wanted to add functionality to be able to monitor assets and their implementation settings and making it possible to modify groups of assets and their settings easier. Easier and more efficiently at any point of the production, without need to go through all those assets manually.

Implementation process of audio assets involves a lot of manual work. When importing the asset into the middleware, also many settings must be defined for it. At least how and when to play the asset and how it is handled in relation to other assets. The number of needed settings varies depending on the role and type of sound, but no matter what the asset, usually at least the basic settings need to be defined for every single asset. While an asset can require very specific settings, there are often similarities between the settings of assets. The implementation work often consists of small manual tasks with lots of repetition and even copying and pasting settings around the project. Identifying these similarities and patterns was an essential starting point when starting to design the system.

After researching typical implementation patterns, I came up with the idea to define presets of settings. Presets can include all needed settings for implementing one or multiple assets. These presets such as including playback limiting and conversion settings, could be created outside of the middleware itself allowing creating and editing them early on, even before the actual implementation work but also to easily share them between projects. In the long run this approach can develop into a creation of a company wide

database including vast number of implementation presets for all kinds of assets and game projects.

In principle the system allows to store needed implementation settings to presets and instead of applying settings manually to assets, letting the automation do it according to the information stored in presets. Since the usage of the system is very much like tagging a preset to asset, I decided to call the presets “tags”. Tags can be applied to any object or asset inside Wwise. After adding the tags, the automation script will be run and it will apply settings to assets and objects according to information of the tags.

In optimal case all assets in the project would be under the influence of at least one tag. With extensive projects this is very helpful for managing data, since tags makes it easier to track groups of assets faster but also compare and edit their settings. Tags and their included settings, are created outside the middleware. Keeping the information outside of middleware allows editing the tags also after applying them to the assets. If any settings inside a tag are changed, those changes are then applied everywhere inside the Wwise project to all assets using specific tag. This kind of functionality makes managing and optimizing large projects much easier and faster because of multiple reasons.

Firstly, it’s easier to understand the overall state of the project when the assets are categorized according to their implementation settings, for example comparing playback limit settings and prioritization between groups of assets is fast and easy. Secondly, updating settings for a huge number of assets is very time consuming when done manually. With the help of the system, it’s possible to make even big changes fast and update a large number of settings very efficiently, this helps to keep the project organized and optimized throughout the production. Thirdly, when the system is programmed well and it functions with reliable API, very little or none bugs are found. This leads to a solid system which minimizes the possibility of human made errors.

During programming and testing the tagging system I noticed how demanding it actually is to create intuitive, reliable software with good usability. Also finding a balance between the level of automation and customization was challenging. For the first prototype, I developed a version of the system allowing only using fixed tags. By fixed tags meaning that system

would take the information included in the tag, but wouldn't allow any modifications to settings outside the tag such as overriding parts of the settings. One thing worrying me was the number of created tags after using the system in the production, having as many tags as assets would not improve the usability enough. After using the system for some time, I noticed that the idea of a simple tagging system was moving more towards creating a scriptable automation layer with its own syntax. When thinking about how it's implemented, it can be seen almost like all objects in Wwise have their own editor, the notes section, which is used to access the system hidden under the hood.

For now, spreadsheets are used to store the information of a tag. Creating and editing the tags is also done through spreadsheets. All possible parameters which tags are able to control are defined inside a "Settings" column. In order to keep the connection between the spreadsheet and Wwise clear, parameters are named the same as they are in Wwise. Other columns in the spreadsheet are the tag names. As in fig 7, the "wpn" tag, shorthand for weapon, could be applied to any object inside Wwise. Settings for "wpn" tag are according to the parameter rows.

Setting	wpn
Inclusion	
Streaming enabled	yes
Prefetch length	2000
Conversion (Shareset)	pcm
Loudness normalization	
Make-up gain	
Center %	
Speaker panning	
Listener relative routing enabled	
3D spatialization	
Attenuation (Shareset)	chr\weapons\snipers\wpn_npc_sniper
3D position	
Hold listener orientation	
Hold emitter position and orientation	
Playback limit enabled	yes
Ignore parent playback limit	no
Limit sound instances to	50
Limitation scope	globally
When limit is reached	Kill voice
When priority is equal	discard newest
Virtual voice behavior	Kill voice
On return to physical voice	Resume
Playback priority	100
Use priority distance factor	no
Offset at max distance	
Play type	random - random
Avoid repeating last x played	children - 2

Figure 7. Spreadsheet with setting parameters and tag "wpn". Screen captured by the author.

If any field for parameter setting is left empty, when settings are applied to an object, that specific parameter will be ignored. This supports workflow of overlapping multiple tags and having a modular approach to control different parameters with different tags on the same object.

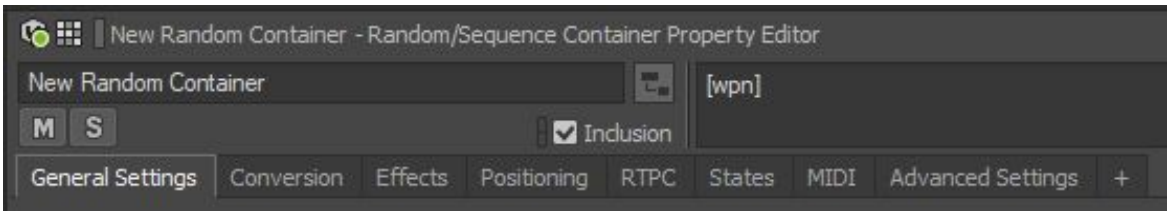

Figure 8. "Wpn" tag applied to the random container's notes section. Screen captured by the author.

Accessing the system inside Wwise works by writing the tags into the object's notes section as seen in fig 8. Notes section is available for any object type and is often clearly visible, located up on the right side corner of a "property editor" view of selected object. Since the notes section is meant for notes and useful for that purpose as well, I decided to create a syntax for tags which is easy to separate from normal text. To separate tags and normal notes, all tags are marked inside square brackets ("[]") and any text outside the brackets is considered being normal notes, as seen in fig 9. Using syntax which differs from normal written text also helps when doing searches through the project, the object with tags and without tags are easy to filter from each other because of brackets.

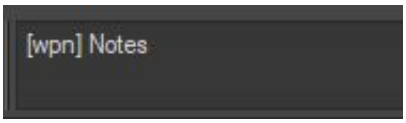

Figure 9. Tags and normal notes are separated from each other with the square brackets. Screen captured by the author.

After adding the tags into the objects, the user will run an update script of the system. The system automatically checks if the syntax of the tags is correct and if the applied tags actually exist in the spreadsheet. Everytime the update script is run, the information of the tags is fetched from the spreadsheet and applied according to the latest information.

Since a single object in Wwise can in practice have as many tags as possible, it allows creating flexible layers of configurations. For example, one tag providing settings for conversion and playback limiting settings while another defining whether asset is going to be a three or two dimensionally positioned. When having multiple tags on a one Wwise object the order of the tags inside the brackets matters. If two tags, on the same object, would have a different setting for the same parameter, the tag on the right side will always override the tag on the left, but this is only happening if they are trying to modify the same parameter. As an example in fig 10, the tag “foley” would override two other tags if their parameters overlap. This allows using tags for temporary changes, for example setting up mixing settings for upcoming conference demo of the game. After the demo is done, the temporary tag can be removed and the system applies the old mixing settings from the old tags which were there the whole time.

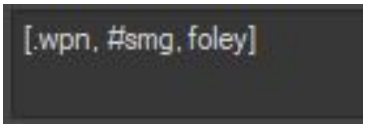

Figure 10. Multiple tags being applied to one Wwise object. Screen captured by the author.

Wwise uses a tree kind of structure, objects have a clear hierarchy of parents and children, where children will inherit the settings from their parent unless they’re overriding the settings of the parent. This kind of hierarchy is quite intuitive to use, so the tagging system has the same principle.

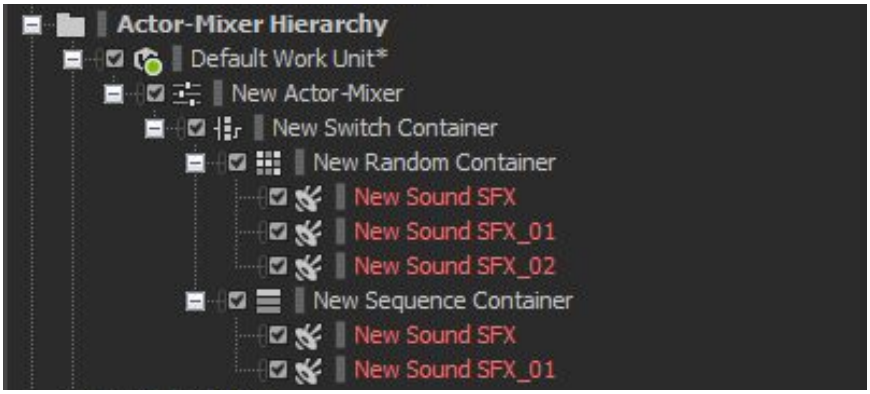

Figure 11. Capture of hierarchy structure of objects in Wwise. Screen captured by the author.

Having a parent with its children creates intuitive groups of objects, which helps to keep content well structured, but also allow implementing bigger entities in smaller portions. In many cases these groups of objects are implemented with exactly the same settings, but it's very time consuming to manually apply settings, or even tags, to every object. Since I wanted to make this process faster, but still have the idea of inheriting settings, I created a similar parent-children hierarchy for the tags. The hierarchy logic of tags automatically manages Wwise's own logic of inheriting settings.

Any tag can be used as a parent tag by marking "#" sign before the name of a tag. When the system notices a parent tag, it automatically adds a child version of the same tag with argument ".". The child tag will be added for every object under the parent, according to the Wwise's hierarchy structure. As seen in fig 12, object has child tag ".wpn", meaning it's inherited from "#wpn" parent tag from higher in the hierarchy. Child versions of the tags are never applied manually, the system handles them automatically, if the parent tag is removed the system removes its children tags accordingly. System also makes sure that a normal tag isn't under the influence of the parent version of itself and removes any unnecessary duplicate tags.

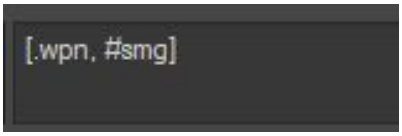

Figure 12. Child tag ".wpn" and parent tag "#smg" being applied to an object. Screen captured by the author.

Instead of only having fixed tags created outside of the middleware, I wanted to allow flexible way to modify the tags inside the middleware, without accessing the spreadsheet. Since the parent-children logic already introduced the way of modifying a tag, I started adding more functionality by allowing users to pass arguments for the tags.

In many cases audio designers might want to work on large entity of assets for a long time and implement its elements separately, but also experiment with different settings and/or multiple versions of the content. To have the possibility to use tags for some objects while keeping other objects, or some of its parts, untouched, I added an exception logic. To exclude an object, meaning to not apply any settings of any of its tags to it, the exception tag "exc" can be used. For example as in fig 13, having "[.smg, exc]" tags on an object would mean that any settings wouldn't be changed by the update script, because of existing exception tag. If the exception tag would be removed, the object would keep getting settings of the remaining tags.

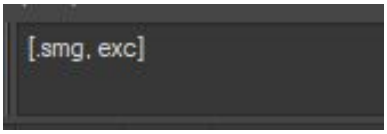

Figure 13. Example of exception tag "exc" being applied to an object. Screen captured by the author.

Assets can be very different to each other and therefore require diverse settings and logic structures. It's possible to create vast amounts of tags to cover all needs and particular setups, but having too many tags isn't very optimal. In many cases asset could have one tag providing most of the settings, but being able to locally change only a few small parameters

would be useful. As a solution, I allowed passing arguments for a tag, to override specific groups of it's parameters with a local setting of a specific parameter. This feature is meant especially for the cases when implementing an asset only with a small amount of custom settings. Overriding is a powerful method and shouldn't be used too often. Since locally overridden settings will be only on the object itself, these settings are not stored into the spreadsheet. When implementing any settings requiring lots of overriding, it's better practice to create a separate tag for specific custom settings. That way the information of the custom settings is stored in the spreadsheet and easy to track, but also to modify later if needed.

Since overriding the tags shouldn't be used too much, I wanted to create fairly limited options on what to be able to override. I decided to split the parameters in groups, in a similar manner how they are organized in Wwise, but splitting some of them to even smaller groups according to how often these parameters tend to be customized. To make overriding visually clear I created a dedicated tag per parameter group with added prefix "!". At the moment of writing the system supports overriding following groups of parameters: inclusion settings ("!inc"), streaming settings ("!st"), conversion settings ("!con"), positioning settings ("!pos"), playback limiting settings ("!pl") and play type settings ("!pt"). For example as seen in fig 14, to get settings from tags "wpn" and "smg", but to override conversion settings, "!con" tag would be added. After applying the "!con" tag, the object's conversion parameters can be set locally, without the system changing those parameters.

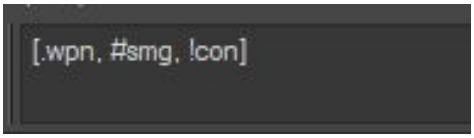

Figure 14. "!con" tag being applied to an object for overriding conversion settings. Screen captured by the author.

There can be cases where overriding parameters for multiple assets is needed, in that case it's possible to use overriding tags as parent tags by adding the prefix "#" as seen in fig 15.

This way the overriding is done on the object with a parent tag and for all its children objects.

Figure 15. Parent version of overriding tags "!con" and "!pl" being applied to an object. Screen captured by the author.

Summary of the syntax arguments:

- "#" - parent tag
- "." - child tag
- "exc" - exception
- "!inc" - overriding inclusion settings
- "!st" - overriding streaming settings
- "!con" - overriding conversion settings
- "!pos" - overriding positioning settings
- "!pl" - overriding playback limiting settings
- "!pt" - overriding play type settings

Having the system functioning on this level was already useful and making the implementation workflow much more efficient, faster and easier. The whole system started feeling more like a platform, literally almost like a new scripting layer, which could be expanded for many other tasks as well. I wanted to make the system more delicate and automated, while keeping it easy to use. I started looking into defining settings for different object types with different logic.

Wwise has a variety of different containers which have their own logic to play assets. Since different containers require different settings, use of containers also creates predictable implementation patterns. When thinking about the system so far, it was simply a code reading information from the spreadsheet and applying it to objects with certain logic. The system could do some much more, be more scriptable and automated.

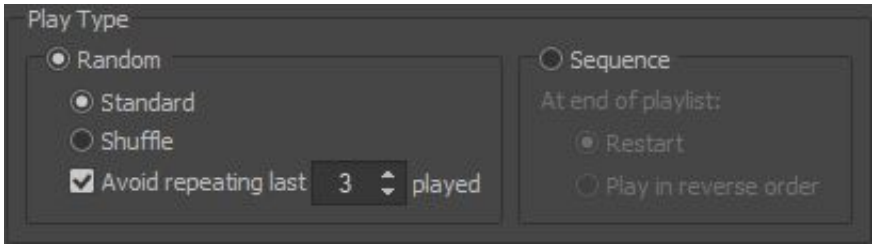

Figure 16. Play type settings of random container in Wwise. Screen captured by the author.

While the possible input values for many parameters in Wwise are very fixed, some parameters such as volume which can be almost any integer value. Random container's play type settings works as a great example where the new functionality for the system was needed. As seen in fig 16, a random container has a parameter "avoid repeating last x played". This parameter controls how many already played assets (x) inside a container, the randomization should avoid repeating when triggered again. In order to have even randomization of the assets, the number (x) should be two numbers smaller than the total number of assets inside the container. When applying integer value for a parameter, there wasn't reason why the value couldn't be for example a result of an equation, so I added support for it. Equations can be used for any parameter on an object supporting integer values. Especially in case of a random container, this is very useful. By using equation in the tag as seen in fig 17, the parameter value is dynamic and applies to every situation accordingly. The system will calculate the number of assets inside the container and apply the value for "avoid repeating last x played" parameter according to the equation. If the number of assets inside the container is changed the system updates randomization settings respectively.

Play type	random - random
Avoid repeating last x played	children - 2

Figure 17. Example of using equation to set a value for a parameter supporting integers. Screen captured by the author

Support for equations is only a start for more dynamic functionality of the system. While equations can be useful with given integers, they are more powerful with the possibility of getting the integer value of other parameters dynamically. For example currently supported keywords such as “children” (as seen in fig 17) and “priority” allow getting the number of children of object or the playback priority of object as an integer. In principle, any information being presented as integer could be gathered from anywhere in the Wwise project and to be used as part of the equation.

Simple example of using the system:

1. Create a random container.
2. Import assets into the container.
3. Apply wanted tags to the container, for example “#wpn”.
4. Run the update script.
5. The script will apply a child “.wpn” tag to every asset inside the container and update their settings according to “wpn” tag settings in the spreadsheet.
6. It’s also possible to override for instance the play type settings of the container by adding a “!pt” tag. The whole syntax would be [#wpn, !pt].
7. Now play type settings can be set locally for the container, without a worry of script changing them.
8. To modify settings of “wpn” tag, open a spreadsheet and change some settings.
9. Run the update script again and modified settings will be applied to all assets using the “wpn” tag.

Over months of developing the system, I let multiple audio designers use it as their everyday tool. This was crucial for testing and finding possible limitations, unexpected corner cases and bugs. I also gathered feedback throughout the process in order to see which features were the most useful, how usable the system was overall and how to develop it further.

The goal of the system was to make implementation faster and easier, and that way managing huge workloads easier and allow better prioritization of the work. I wanted to improve the usability of Wwise, so that even less technically oriented people could implement assets without getting too deep into the settings. I also added functionality which didn't exist such as getting good monitoring over great amounts of assets and powerful batch editing. Overall the biggest challenge was to minimize repetitive, manual work and eliminate human made errors.

I'm quite happy with the current stage of the tagging system, but naturally there's still plenty of things to improve such as usability and the management of tag information. Spreadsheets aren't the optimal way to store the information. Since it's a separate document outside of Wwise, audio designers need to have it open while implementing or they need to remember the settings of the tags they're going to use. Even though the system checks if applied tags actually exist, a better solution would be to have an overlay or auto complete functionality. With clear overlay when trying to apply tags, the system could clearly show and offer the available tags for the user. This would make the system easier to use especially with a large database of tags. At the current state of the system, the information is only read from spreadsheet but there's no support for sending information to spreadsheet from Wwise. Being able to select an object in Wwise and storing its current settings as a tag into the spreadsheet is a feature to be added in the future. That will be useful especially when building up a database of tags from different Wwise projects and already made implementation setups.

According to feedback some users find the tagging system easy to use while others don't, so it's difficult to make any conclusions based on the feedback. Overall usability of the

system is fairly good, and in any case makes the implementation work much faster. If this kind of functionality wouldn't be possible to include into middleware itself, the idea of writing tags into the notes section could be also replaced by separate software. The external application could work as a tag editor and tag database manager which then communicates with Wwise objects. The application could include useful features such as drop-down menus for parameter values, project specific databases of tags, database search features and extended functionality for monitoring like visual graphs comparing certain settings between the tags and groups of assets.

Generally speaking bad usability is tricky to deal with without changing the core functionality of the program, if the middleware isn't nice to use, trying to solve this by creating extensions isn't really tackling the main issues. In case of the tagging system, the approach was to create additional functionality which is also trying to hide the original, clumsy functionality. This works fairly well, but naturally in order to create a middleware with efficient workflows and with great usability, these should be the design goals of the core developer team. From my experience usability is often overlooked by the amount of features. While middlewares try to cover all possible purposes they end up slowing down the overall iteration time when sudden changes have to be made. Fast iteration times are crucial for game development, but also make the user experience much better. This is not only a matter of decluttering the user interface (UI), but also providing the tools and the power to the user when it's really needed. In case of Wwise, the UI has plenty of parameters and visual indicators visible most of the time, even if they would be completely irrelevant. While usually the work executed with middlewares is technically quite simple and straightforward, finding the right parameters can take lots of time and clicking only because of the design of the UI. Since we live in the time of digital tools, the UI design is extremely important in order to make working with the tools, and therefore the work enjoyable. As Myllys (2014) states when describing the development of DAWs; "current workstations offer a vast amount of processing power, but the inflexible user interface restricts the use of this power to the execution of conventional procedures" (p. 60). This is true with game audio middlewares as well, and the fact that extensions need to be made in order to use all that power is quite alarming.

While the tagging system isn't optimal when it comes to usability, it still has proved to be able to make the implementation process much faster and in many cases easier. After

discussing with audio designers and gathering the feedback, it became more clear than before that implementation usually follows certain patterns which can cover large amounts of content. Tags with right settings can automatically make sure all this content is correctly implemented even if some assets might change or get deleted during the development. While the system decreases the amount of technical hassle, it also makes the process much faster in many ways. For reference, to only enable “streaming” setting for 20 music segments in Wwise would require approximately 4 to 5 clicks and around 3 seconds per asset depending on the hierarchy, in total 60 seconds. With ready made tag, only selecting a parent container, applying the tag and running the update script, the whole task to implement all 20 assets would take around 5 seconds. Also making changes on settings of the tag and updating it to all 20 assets would take less than 2 seconds, instead of taking again 60 seconds. Naturally one major point here is that the amount of content per tag doesn’t notably affect the execution speed. While this might not sound like a huge difference, it is. Even small improvements in speed easily add up when talking about really large game productions requiring years of work, but in this case we’re talking about 10-20x improvement in speed depending on the situation.

Like mentioned earlier, maintaining custom tools requires a lot of time. While the tagging system can end up being quite a significant amount of code and already requires regular maintenance, it’s worth it when taking into account how much it speeds up the whole audio production. Also since the time spent on implementation is shorter it allows programmers and technical audio designers to use the time for developing new tools and features. Overall when the repetitive and time consuming manual tasks can be executed very fast it saves time and allows better prioritization of the workload. While audio designers spend less time implementing they can use more time to be creative and iterate on the content.

In the future the tagging system could be even more automated. Since the principle of using tags and automation functionality are two separate processes they can be developed separately. I’ve been prototyping predictive functionality which allows the system to be even easier to use yet faster, by automatically matching the tags for assets according to its filename. For example, if the name of the asset would be “npc_wpn_smg_foley_reload_01”, the name is essentially a string of keywords. Automation script can easily parse the name into keywords and try to match them with

already existing tags. Since the naming convention already describes the hierarchy structure, the system could easily apply tags to the asset, or its parent object, and match the tags within the hierarchy of surrounding tags and objects. Even moving the assets to their specific location according to their name could be a useful feature. With this kind of functionality the process of implementation can be made even more efficient, since also the step of applying the tags could be automated.

The system could also expand its functionality outside Wwise. In principle any information from the spreadsheet can function as a keyword and have any functionality in the code. This would allow all kinds of expanded functionality such as getting information and making changes in any object in Wwise or starting external processes like communicating with the game engine if implementation of some sound event has changed in the game.

I'm happy about the fact how flexible the system has become and how easy it is to add new features. It seems to be holding up well and has become a nice platform to build on. On the other hand, when developing the system to be more complex with additional predictive features, the usability and flexibility might be compromised.

While the system solves many of the challenges when it comes to implementation, that's only a half of the workload. Automated implementation process certainly helps when handling lots of content, but doesn't solve the actual problems created by very linear content creation. In many ways it's quite depressing that development of this kind of extension is required in order to be able to keep the audio work somewhat enjoyable. Clearly proper paradigm change from linear to non-linear audio work hasn't happened when it comes to tools, I really hope this will change in the future.

Conclusion

While working on the field of games, researching and writing this thesis I've encountered many articles, books and talks which tend to cover similar challenges I've been covering in the text. Some of these sources being more than 10 years old clearly indicates we are not talking about new challenges. Even though many of these problems will occur when the size of production is quite huge, it doesn't mean there wouldn't be problems in smaller productions.

I think game audio is technically still a very unexplored field when it comes to workflows and pipelines of large, modern games. While it seems games aren't going to get smaller, I don't think the workloads are going to be manageable in the long run. Often high-level organization of production and scoping is blamed and indeed they are important factors, but those actions are part of the production even if the process would be optimal and workload completely doable. I think the discussion and future development needs to go more towards systematic and non-linear direction especially when it comes to content creation. With optimized workflow it's easier to prioritize needed content within any scope but also allow audio designers to have more creative freedom and means to develop game audio as a whole.

I don't think anything develops further without questioning the current situation and recognizing the problems. The ongoing debate in the game industry regarding working overtime for long periods (crunching) indicates clearly about existing problems. Objectively I'm surprised how rarely the effects of nonoptimal tools are considered during these debates. I'm not saying better tools would solve these problems completely, but they could help and make the work more enjoyable. After finishing a project, it's quite different to remember struggling through the years than that at times you were able to have the time to create something beautiful, since the tools weren't slowing you down.

I think it's a matter of time and change of mindsets, if we can we reach better workflows with more suitable tools. While custom tools could be developed inside game companies that's not always reasonable, but certainly having only a few advanced but compromised, third party audio middleware options for the whole industry isn't optimal either. While

tools such as DAWs and middlewares are used by audio professionals every single day, they have an immense effect on how to work feels like. This comes down to ergonomics, good work-life balance and wellbeing and therefore to a reputation of the whole gaming industry. I truly hope these are factors which matter enough and will lead to a much broader discussion about the workflows and how to improve them. After all, it's about how to make great games and still have fun.

References

Karen Collins: **Game Sound : An Introduction to the History, Theory, and Practice of Video Game Music and Sound Design**, MIT Press, 2008.

Aaron Marks: **The Complete Guide to Game Audio**, 2nd Edition, Focal Press 2012.

Scott R. Looney, Steve Horowitz: **The Essential Guide to Game Audio**, Focal Press, 2014.

Rob Bridgett: **Better Sound Through Collaboration** https://vimeo.com/122122759, Game Developers Conference, 2014.

Bernard Rodrigue: https://blog.audiokinetic.com/introducing-the-wwise-authoring-api/, Audiokinetic blog, 2017.

Jason Hayes: https://www.gamasutra.com/view/feature/130074/the_codeart_divide_how_technical_.php, Gama Sutra, 2008.

Petri Myllys: **User Interface paradigms in Digital Audio Workstations : examining and modernising established models**, Taideyliopisto Sibelius-akatemia, 2014.

https://www.audiokinetic.com/products/wwise/, Audiokinetic Wwise website.

https://fmod.com/, Firelight Technologies FMOD website.