Adaptive 3D Game Audio with Unity Engine

Giovanni Librizzi

California Polytechnic State University – San Luis Obispo
Senior Project
Professor April Grow
June 2024

Introduction

Audio is a key component in games that's often overlooked. It has the power to give atmosphere and tone to a game that could otherwise feel totally different based on its visual appearance. With that power, it's also an important way to give players feedback for their actions and the events occurring in the game. It can increase or decrease tension that's happening in gameplay, and also give the player different emotions such as fear, determination, or happiness. I've always been interested in the effect music has on games, and one interesting way that might increase the impact music has on the medium is by having the audio be adaptive. It seems to have the power to increase immersion not only by having sound effects have realistic effects based on the environment, but by having the music change dynamically with the actions and events occurring during gameplay, leading to the soundtrack feeling more like a cinematic score for a movie. I want to look into seeing the effects adaptive audio has on the emotions players feel when playing them, and if it can cause the game to leave a greater impact on the player.

Background

In order to achieve this level of adaptive audio, I chose to use the Unity game engine and the FMOD audio middleware that can directly interface with Unity. Unity is an engine that allows you to create 3D games by creating game objects and adding built-in components that give them functionality, or by attaching custom scripts written in the C# programming language. This programming environment makes it much easier to work on a 3D game, as it's able to manage the difficult aspects of 3D games such as collision detection and physics, allowing the programmer to focus more on implementation.

FMOD studio is a sound engine that can directly allow you to have a lot more control over the sound and music in your game than what game engines will typically provide. I decided to use FMOD for this project, as this flexibility allows you to manage multiple audio tracks at the same time, create adaptive audio events using its built-in event editor, and also change audio effects such as reverb or a low-pass filter in real-time through code. It also has built-in support for Unity integration, which makes it a lot more simple to work with. FMOD allows you to send parameters from your engine of choice, which you can then configure how those parameters affect your music event. For instance, you can set up a parameter such that when it's at a certain value, a certain track can raise in volume while another track can mute, or you can have it change an audio effect a track has in real time such as a reverb or an equalizer.

Related Work

One of the game series I was directly inspired by for this idea was Pikmin. Most notably, the first two games on the Nintendo GameCube had an adaptive music system that would dynamically scale what instrument layers were active based on several different factors. They use MIDI playback, which is where a file with just raw note data is processed by a virtual

instrument to play those notes on the fly, which gives them a lot of flexibility in how they modify their music during gameplay. This allows them to add and remove different layers, and also easily replace the instrument being used to play those notes. Pikmin 2 took this and added a lot more complexity to the system, by having the music be not only adaptive, but also procedurally sequenced, meaning different chunks of a song would play together seamlessly in random orders at different times. Using MIDI data also allows for them to easily quantize to different rhythms, such as changing all the music to have a swing feel when using a different character. My goal for this project was to implement adaptive audio into a game in a similar fashion, but instead of using MIDI data attempt to use normal audio files, but just have multiple audio tracks for each instrument and have them organized as such. Those first Pikmin titles were designed for GameCube, whose disc format had a size limit of around 1.46 GB. Audio is typically very large in size, especially for uncompressed and sounds longer in length, so MIDI data was a useful necessity. However, it often lacks the character of using real instruments (although the sounds do have their own often silly and nostalgic quirks), so that's why I want to attempt to use real audio for this project.

Many other games use adaptive audio, and one recent indie example that stuck out to me was the game Celeste. It will usually have separate different music tracks per level, which will seamlessly flow between each other based on what the intensity of the gameplay is. This game also uses FMOD Studio to manage this system, and the audio designers have publicly released the FMOD project for the game, allowing anyone to learn what tricks they used.

Design

My game is a third-person puzzle 3D platformer where the player character must explore a small level to find jars. These jars give the player different active effects depending on their type (what they have inside), and are able to be thrown by the character to cause different effects to enemies or the environment. The player can stack a high number in their hands at once, which will stack both visibly and actively with any status effects they contain. For example, if the player is holding a feather jar, they will be able to have increased jump height and decreased fall speed, which will also stack and increase accordingly with how many they're actively holding. They will also be able to throw the jar that's currently at the top of the stack (most recently picked up), which in the case of the feather jar will send any movable blocks/objects a fixed distance in a direction, and will also send nearby enemies flying backwards. There are several different jars, such as a feather jar, fire jar, and a slime jar, each with their own special effects

that can interact with the player, enemies, and the environment in unique ways. This allows the player to determine how they can use the different effects of the jars to progress through levels, obtain the key, and reach the exit. The key gets placed at the top of the stack, and as it's not a cylindrical jar it prevents any additional jars from being picked up, making it so the player must strategically plan their exit pathing.

This core mechanic allows me to have the music adapt differently based on the different jars being currently held, along with the intensity of the current gameplay due to enemy interactions. As the amount of jars increases, different instruments and versions of those tracks with modified notes layer on each other based on those different types of jars. The intensity of the current track also increases based on if the player is in a state of peace, wariness, or danger, adding faster percussion with more intense instrumentation. All of this occurs as seamlessly as possible by utilizing FMOD's features. Additionally, as the player enters new areas of the level, the song can add new sections or switch to a different section of the track entirely using FMOD loop regions and transition markers.

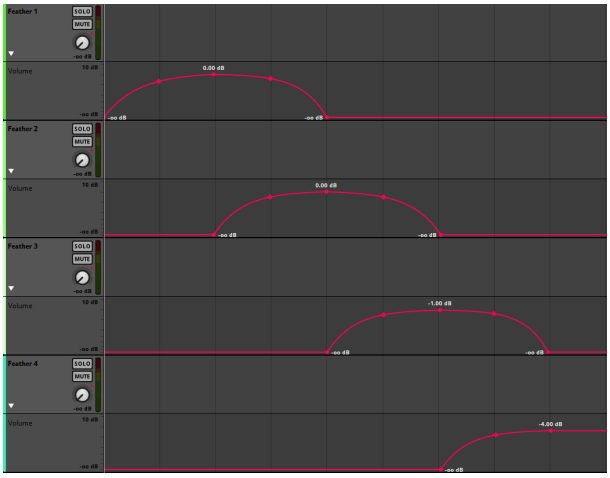
Implementation

The main Player game object has a PlayerMovement script that allows for fluid, third person movement that uses the direction the Cinemachine third person camera is facing as the forward direction. This player game object has a separate script that handles grabbing the jar objects that are located around the levels. When the player presses the jar-pickup button, it spawns a hitbox that checks for jars and will pick up any jars that are in that hitbox. It adds them to a stack holding all of the jar types, and it also updates a dictionary that is keeping track of the amount of jars of a certain type (eg. feather, fire, slime) that are currently being held.

Jars each inherit from a main Jar class that processes what happens when it's thrown or held by the player. That parent class handles the general movements of the jar, while the individual child classes can apply their own modifications to what happens when they're held or thrown. For instance, the FeatherJar class that inherits from the Jar class will update the player's move speed and jump force the moment it's added to the player's inventory, and will revert it back when it's thrown. It also will apply additional wind knockback to any applicable objects it hits when it shatters.

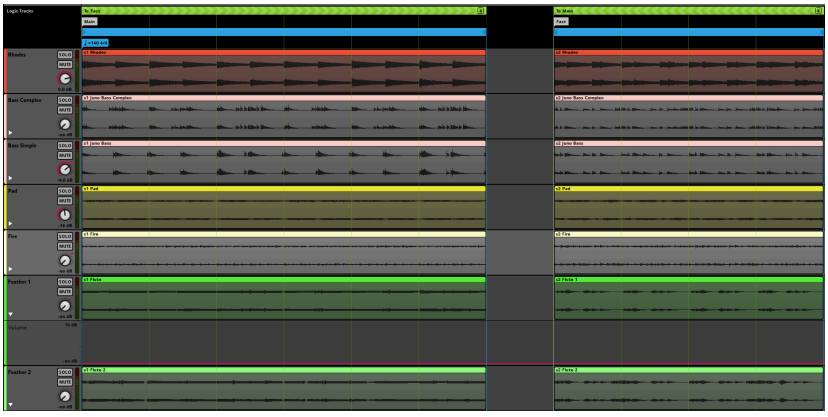
When jars are picked up, it additionally calls a function in the AudioManager singleton that tells it how many jars of that type and how many total jars are currently picked up. The AudioManager will then interface with FMOD to update the respective parameters of the current music instance that's playing. The FMOD project is set up so that there are music event instances

that hold the individual tracks for the songs such that separate instruments can be modified in real time in the game. When those parameters in FMOD are updated, the music events are set to change different parts of specific tracks in the music event, such as swapping out a melody track with another, changing the volume of certain tracks, and changing the panning of the track. This is done using FMOD's parameter automation, which allows us to map values of variables our game sends to FMOD into values of parameters in FMOD's sound engine.



The song is implemented in FMOD such that when the player has zero jars, it will only play the core chords of the song with a Rhodes electric piano along with a simple bass line. Once you hold a single jar of any type, the song will smoothly add a drum layer and a warm synth pad on the next beat of the song, making the track feel whole. At this same time, it will also add a new layer of the song based on the type of jar that's being held. Fire jars add in a plucky alternating chord track, slime jars add in warbly synth chords, keys picked up add in a high bell melody, and since feather jars are the most utilized they have four synth flute melodies that increase in their complexity. The bass also triggers to subtly become more complex after holding 2 or more jars. As you come close to an enemy, it changes the state of an enumerator to "wary" which in turn triggers maracas and a more alert synth lead. Once the enemy starts attacking you, it triggers to a "danger" state, which adds more alarming chords and a hi-hat pattern.

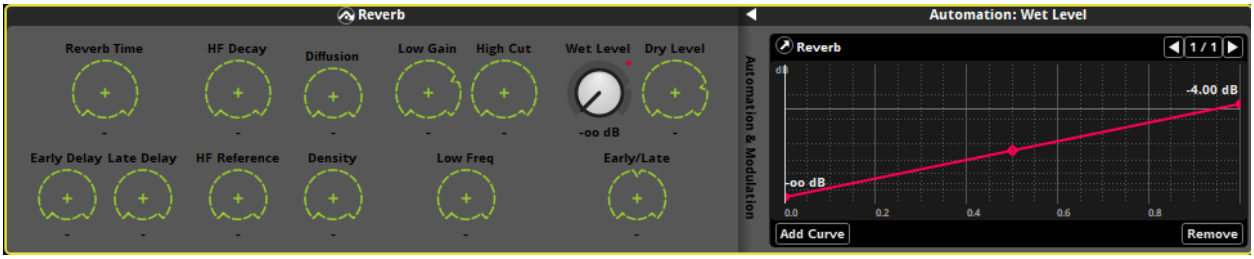
I initially thought FMOD would have a way to quantize the changing of the parameter in-engine, meaning it would wait until the next beat in the track to start changing the appropriate values. However, after additional research, it seems that it doesn't directly support this kind of parameter quantizing, so I had to approach the problem in code. I implemented a custom beat event callback method I found that was able to interface directly with the FMOD timeline in order to get information from it, such as the current beat of the song and the last marker that was reached. I then set up a static event action in the AudioManager that would invoke every time the beat is changed. This event lets me quantize anything in the game so it's synced with the music, so I can now have the jar parameter wait until the next beat update in order to make music changes as smooth as possible. I also



configured sound effects like player footsteps to play on the beat, making them a lot more musical than before.

I also set up the parameters in FMOD to be continuous numbers rather than integers as they were stored in code, as FMOD is able to smoothly interpolate between values of continuous numbers, but not with integers. Doing this allowed me to have the different music tracks fade in and out when triggered, rather than immediately adding a new instrument at max volume as that can make the new tracks sound a bit jarring.

I set up audio trigger objects in the game world so that when the player entered the collision trigger, it would update a parameter in the FMOD music event. This parameter is hooked up to a transition region in FMOD so that once that parameter is true, it will wait until a certain quantization interval until jumping to the specified marker in the track (this quantization is only set up for marker transitions and unfortunately not for other parameter transitions). This allows me to change the music based on the player's progression through the level, making the song a lot more dynamic and immersive. I also have triggers in large rooms that add reverb to all sound effects that occur there to give the illusion of those sounds echoing in the space. This is done using an FMOD snapshot, which are instances of the mix with modified parameters that can be triggered in code. This means I am able to save a reverb effect that has the wet volume parameter (which controls how much reverb occurs) set at a certain value, allowing me to run a function to enable that wet volume parameter to change to that saved state at any time. I have that reverb snapshot configured to only affect the sound effects group bus, making it so the other audio in the game does not have the reverb effect.



Analysis / Verification

The majority of the playtesting sessions occurred before other impactful effects such as particles or screenshake were implemented, which helps us discover more about the feedback and core impact the audio had on the players. The first playtesters were confused about the core mechanics of the game as it was initially unclear how they functioned, especially with no button prompts to let them know what controls they could use, so tutorial prompts were added that helped clarify the mechanics. The additional music tracks were only linked to the total number of jars held instead of differing based on the type of jar during the initial playtests, and were a bit more subtle. Additionally, the initial song in the game was slower, calmer, and a bit repetitive, as it didn't have any significant changes throughout other than adding a small additional part. Playtesters initially felt the music was more atmospheric and chill, and most of them didn't notice that the music was changing during the gameplay. Having players not notice the music

changing was a good sign, as I wanted to make sure it wouldn't be distracting to the player having layers come in and out while they're playing. Players noted that the changes weren't jarring at all, but the game was a bit loud and players wanted a way to adjust volume levels. This initial song may have been too chill and repetitive for the game, as one playtester noted this saying platformers need more fun and engaging music, as they found it to be a bit boring.

This feedback was very useful, as it seemed that the initial adaptive implementation was good, but the music just needed to be improved to be more impactful with these changes. I made a new song that was more interesting, added separate layers for different types of jars being held, and also added a new part to the song with a different intensity that triggers in certain parts of the levels. I also implemented a sound options menu that allows for players to change the levels of music, sound effects, and the environment soundscape individually. Playtesters found these changes to be a lot more impactful, as although it was more noticeable that the music was changing, it also made the core mechanic a lot more impactful to interact with. Testers noted that when you have no jars, the music slows down in complexity and makes the game feel a lot slower and open, but as you gain more jars it makes things feel faster and more powerful, making you want to keep that momentum going. They really enjoyed the separate layers that specialty jars added, noting the key sound felt very satisfying. Testers found the "wary" and "danger" tracks to be a bit jarring, so I adjusted them to blend more with the current track. After all of the changes were made, players really enjoyed how the music and sounds added to the experience while still not being too distracting during gameplay.

Future Work

The core gameplay and audio functionality is there, but I would definitely love to add more of a story to the game. A story would really help add more worldbuilding, which is able to pair with how the music and sound effects sound to make the game world feel more believable to the player. With that, more enemy designs and types could be added, as there is only one rudimentary enemy at the moment. Additionally, more levels and jar mechanics could be added, with different theming and new obstacles that coincide with those new jars to create interesting platforming and puzzles. New parts of the music could be added so that there's more variety and the emotions of the player can evolve while exploring the level.

These would help make the game a lot more interesting, but at the same time new enemies, jars, and levels raise a serious question about the complexity of the music needed for the current implementation rules. Each new level theme would need its own song, with various different sections of the level having evolving sections of the music. Each jar will then need numerous individual parts recorded for every new song (as long as that jar is used in the level). Once there are multiple enemies, it could also be set to have different instrument parts be added for every separate enemy type that's being encountered (although the current implementation is only based on the state of danger the player is in). It's important to keep the scope of a project and its different audio adaptations in mind, as the scale of the work for music scoring can quickly increase with a small change.

Conclusion

I'm very happy with how the project has turned out. Being able to work on a project with audio middleware has been something I've always wanted to try doing, but never had the opportunity to until now. This project also got me a lot more familiar with all the different aspects of game development Unity has to offer, and got me comfortable with its workflow. Using FMOD and discovering all of its vast features and various quirks helped me solve different challenges not just for implementing sound in games, but also with recording and writing music. With how important audio is for games, I feel that this game has done a great job emphasizing it in a memorable way that leaves an impact on the player.

Bibliography

Music in Pikmin 2. Pikmin Wiki, January 20, 2024, Retrieved January 26, 2024, from https://www.pikminwiki.com/Music_in_Pikmin_2

Screenshot of Pikmin (Gamecube, 2001). MobyGames, March 16, 2002, Retrieved June 9, 2024, from https://www.mobygames.com/game/5674/pikmin/screenshots/gamecube/24247/

Delving into Pikmin 2's Procedural Cave Music. Scruffy, December 6, 2023, Retrieved January 26, 2024, from https://www.youtube.com/watch?v=QPFWzSlg-78

FMOD - Appendix: Celeste Getting Started Guide. FMOD, Retrieved January 26, 2024, from https://www.fmod.com/docs/2.02/studio/appendix-a-celeste.html

FMOD + Unity Beat Tracking. ColinVAudio, January 19, 2021, Retrieved February 25, 2024, from https://www.youtube.com/watch?v=hNQX1fsQL4Q

Pikmin. Nintendo, 2001.

Pikmin 2. Nintendo, 2004.

Celeste. Maddy Makes Games, 2018.