GeoBlade: An Audio-Driven Action-Adventure Game

A Major Qualifying Project

submitted to the faculty of

Worcester Polytechnic Institute

in partial fulfillment of the requirements for the

Degrees of Bachelor of Science in

Interactive Media and Game Development

And

Computer Science

by

Owen Aguirre

Luke Bodwell

Cameron Jacobson

APPROVED:

Brian Moriarty and Keith Zizza (IMGD), Lane Harrison (CS), Advisors

28 April 2022

Abstract

We designed and developed *GeoBlade*, a 3D third-person action-adventure game, with the goal of learning and making use of industry standard tools and techniques to craft a memorable, interactive experience in which audio plays an integral role. The game blends melee combat and puzzle-solving gameplay in a narratively rich, location-driven story. It was developed in Unity and makes heavy use of Wwise Audio Engine and Houdini Engine. We conducted two rounds of user testing and gathered feedback from 40 playtesters which helped to further refine the gameplay experience.

Acknowledgements

This project was made possible by the guidance of our advisors Prof. Lane Harrison, Prof. Brian Moriarty, and Prof. Keith Zizza. We would like to thank them for their help, as well as Kiley Hoot for lending her voice to the role of Iris.

Contents

1. Introduction

GeoBlade is a 3D third-person action-adventure game that takes place in a distant post-apocalyptic world. A civilization of humanoid beings known as the Hepheons have gone into cryostasis deep within a vast underground city called The Geode following the near-depletion of a naturally occurring energy source known as Geo, upon which they had become overly dependent. Experiments in digitizing organic lifeforms led to the creation of Surface Reconnaissance Units – Hepheon cyborgs that were re-purposed by an AI system known as The Core to scout the surface and determine when to re-awaken the remaining Hepheons. Over the years, Geo has begun to return to the planet and various fauna on the surface have evolved alongside it. These beings are known as Geomorphs, and some of them, possessing a vast amount of inert Geo energy, have developed sentience.

Iris is a sentient Geomorph who comes from a village not far from the buried entrance to the Geode. The Hepheon civilization exists only as a mythos to the Geomorphs, with most of them believing it never truly existed in the first place. Iris is a researcher intent on proving this ancient civilization to be real, and one day, she stumbles upon The Geode, and subsequently the facility which houses The Core's main systems as well as the Surface Reconnaissance Units. Inside, she awakens SRU-02, a Surface Reconnaissance Unit controlled by the player, whom she nicknames Seru.

Due to his premature awakening, Seru experiences amnesia. Together with Iris, the two make their way through the facility and the city before ultimately escaping to the surface. They soon encounter a renegade Surface Reconnaissance Unit and begin to realize that the truth of this world goes far beyond what either of them had imagined. As such, the overarching gameplay loop is a combination of melee combat and puzzle encounters generally centered around audio and musical concepts such as pitch and rhythm.

In designing and developing this game, we set out with the goal of learning and making use of industry-standard tools and techniques to craft a memorable, interactive experience in which audio plays an integral role. We originally wanted the game to feature three levels, complete with a progressively expanding collection of player abilities, enemies, and characters, as well as over 300 lines of dialogue and an original soundtrack.

The game was developed using the Unity game engine and makes heavy use of two prominent pieces of middleware: Audiokinetic's Wwise Audio Engine and SideFX's Houdini. Wwise was used to create immersive soundscapes through dynamic, spatialized, state-based sound effects, interactive music, and fully voice acted dialogue, while Houdini made procedural asset generation workflows possible.

We conducted two rounds of user testing to gather specific feedback on the game's systems and used it to further refine the experience. The first round of testing was done during WPI's IMGD AlphaFest in November of 2021, and the second was done in Prof. Moriarty's IMGD 2900 class in April of 2022.

2. Design

2.1. Geo frequency puzzle

While we had initially intended to include multiple puzzle encounters in the game, we ended up having to settle for just one due to time constraints. The one puzzle that did end up being implemented was the Geo frequency puzzle.

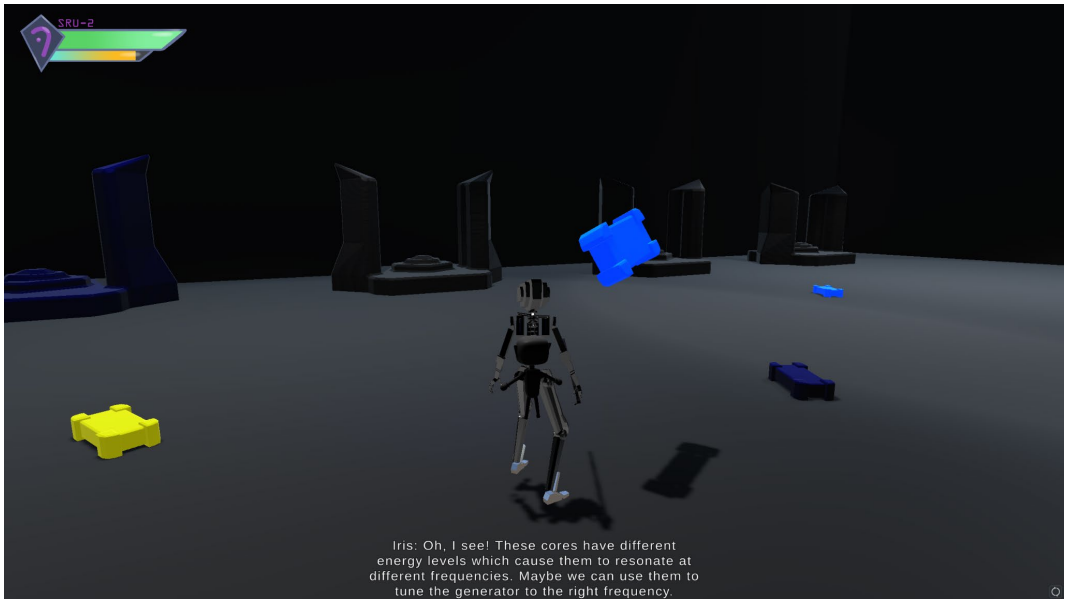


Figure 1. Geo frequency puzzle in-game. Source: Screen capture.

With the game's focus on audio in mind, we wanted to design a puzzle with mechanics that related to audio in some way and were ideally not too difficult to understand while still being engaging enough to provide some level of challenge.

Narratively, Seru and Iris had to divert Geo energy to a different area in the city by modifying a backup generator. What we arrived at mechanically was essentially a game of matching frequencies. There are four generator receptacles that each resonate at a certain frequency, giving off a unique tone in sequence, as shown in Figure 1 above.

Also seen in the figure, there are multiple “Geo cores” scattered across the floor that each have a different energy level indicated by their color. Our ongoing design aesthetic of blue representing low-energy Geo and orange representing high-energy Geo is maintained here by the color gradient between cores.

The Geo cores can be picked up and inserted into the Geo receptacles in order to properly tune the generator. Each receptacle has a target energy value that maps to the note it is emitting musically, as well as a current energy value that is equal to the sum of the individual energy values of each core inserted within it up to a maximum of seven. When a receptacle’s current energy is greater than or equal to one, it produces a second tone in a register two octaves higher than that of the target frequency and glows blue. When the target energy and current energy of a given receptacle are equal, and the two notes played are therefore exactly two octaves apart, that receptacle is considered to be in a solved state, glowing orange. Once all four receptacles are simultaneously in a solved state, the entire puzzle is solved. An explanation of how these mechanics were implemented on a technical level can be found in Section 5.6.

```
Puzzle Mapping (E minor scale):
E -> 1
F# -> 2
G -> 3
A -> 4
B -> 5
C -> 6
D -> 7

Solution:
A, B, F#, G
(4, 5, 2, 3)
```

Figure 2. Geo frequency puzzle solution and musical mapping,

The solution to the puzzle is actually an excerpt from the main guitar riff in Seru’s musical theme (Figure 2). This piece is in the key of E minor, so that was used as a starting point for developing a mapping of energy values to musical notes. There are seven possible energy levels and seven notes in a minor scale, so the obvious solution was to directly correlate the energy values with scale degrees.

One of the main areas of feedback we received in our second round of playtesting centered around the difficulty of this puzzle. Some participants found the puzzle rather easy, while others found it nearly impossible due to the lack of visual feedback. For players without any musical training in identifying intervals between pitches, the puzzle largely came down to trial and error. With this in mind we set out to fully redesign the puzzle from a visual perspective while keeping its mechanics largely intact. We were unable to implement all of the improvements we hoped to make in time for the presentation of the project, but the specifics of this feedback and our proposed changes can be found in Section 6.2.

2.2. Writing

2.2.1. Characters and motivations

Geoblade features four core characters: Seru, Iris, Renegade and The Core. Originally, Seru and Iris had other names – Seru was named Android and Iris was named Fairy. Iris takes inspiration from companions such as Navi from *The Legend of Zelda* and Paimon from *Genshin Impact*. Seru and Iris were based off of the hero and mascot companion trope. Renegade was designed to fit the anti-hero archetype, taking inspirations from characters such as A2 from *Nier: Automata* and, to an extent, Metaknight from *Kirby* and Archer from *Fate/Stay Night*. We wanted The Core to be a manipulative overlord character, taking heavy inspiration from GLaDOS from *Portal*.

It was very important for us that each main character had distinct motivations and personalities. That way, the player could more easily distinguish them and we would have a stronger, more developed cast.

Iris is distinctly different from the other characters. She represents the new civilization and the nature of the world. Therefore, we needed her to have no strong background or history with the Hepheon society. She is cheery and upbeat, a refresher from the rest of the cast, and she is motivated by her drive to prove her research of the "old civilization."

We wanted Seru to be servient, righteous, and naive. He is subservient due to his nature as a robot designed to help people. He is righteous because we wanted a compassionate hero who would help those in need. And he is naive, not only contrast with Renegade's character, but also to let himself be overly trusting of others and deceived by The Core.

We wanted Renegade to be a stuck-up bully, reflecting what Seru might become if he were to abandon his mission and stay on the surface. Renegade had made friends with an older generation of sentient Geomorphs, and developed a distrust in The Core. He has been activated for longer than Seru, and has more experience. Compared to Seru, much of his old naiveté has been washed away by time. Renegade is motivated by his drive to protect the surface from the Hepheons and The Core.

The Core was programmed with one purpose, and it will do anything it can to revive the Hepheon civilization. It is a very "the ends justify the means" type of character. The way it differs from Renegade is in its methods of getting what it wants. Renegade attacks directly and take what he wants by force, while The Core is more coy and manipulative.

2.2.2. Environments

The story of *GeoBlade* centers around Geo, a non-renewable energy source that was once ubiquitous, but has been used up and drained by the Hepheon race.

We planned two main areas for the game, the underground city named The Geode and the surface world. The underground city is where the Hepheons used to live before they ran out of Geo and cryogenically froze themselves. Now The Geode is run by an AI known as The Core, whose main objective is to wait until Geo is restored to the planet and wake up all of the Hepheons. As such, The Geode, while filled with developed structures, is eerily empty and devoid of life. On the other hand, nature has reclaimed the surface of the planet, which is brimming with life and large amounts of harvestable Geo. This sets the stage of our story.

Our intention was for *GeoBlade* to serve as an allegory about environmentalism. Geo represents non-renewable fuel sources like fossil fuels, Hepheons represent the human race, the underground represents our current and immediate future, and the surface represents our planet's past and future – what it was and what it could be.

3. Art

3.1. Concept art

Before working with any finalized designs, we first created concept art to draft designs over several iterations. For all of our concept art, we used a combination of 2D assets made in Photoshop and 3D assets made in Blender.

3.1.1. Environments

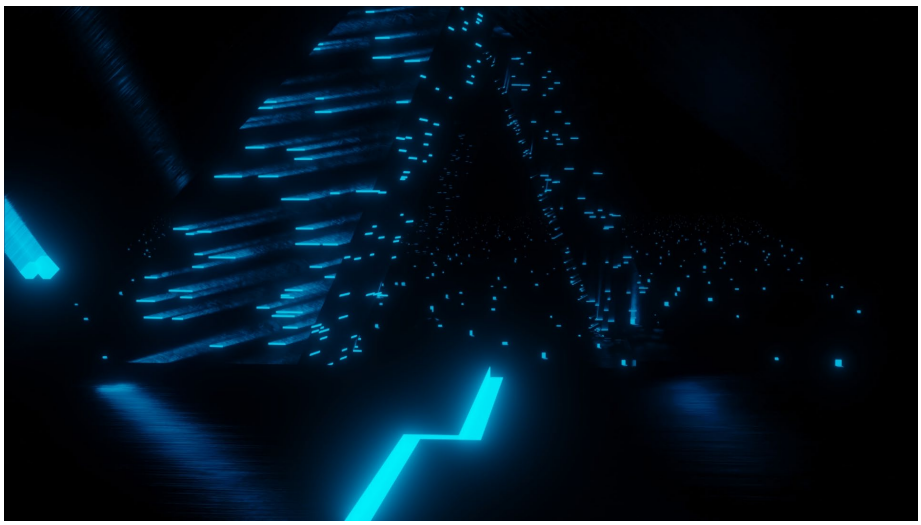


Figure 3. An early concept of the underground area. Source: Original art.

When we first pitched our idea, we knew that we wanted to make a high-tech vs. nature themed game. Our initial concept art for the game was a *Tron*-esque landscape inspired by the Ancient Hylian Shrines from *The Legend of Zelda: Breath of the Wild* (Figure 3). Knowing that this exact landscape probably wasn't going to be added into the game, this highly detailed landscape is actually just made up of very primitive geometry with some shaders. Viewing this environment from any other angle makes the scene fall apart, but it illustrates the feel of what we wanted to create.

Figure 4. Concept art of The Spiral. Source: Original art.

The Spiral is the first main section of the underground level. The structure has decayed overtime, so it looks decrepit. Figure 4 was prepared to help create an asset list for this area. As such, it's filled with various props such as boxes, lights, broken wires, and cracks in the ground.

Figure 5. An early 2D desert concept painting, made in Photoshop. Source: Original art.

Figure 6. Early concept art for the desert area. Source: Original art.

We knew we wanted the surface to be very natural and void of many buildings and marks of living civilization. At first, instead of a 3D model, we came up with a 2D landscape of a desert populated with Geo obelisks (Figure 5). This was painted in Photoshop. Using this painting as inspiration, we then made another small 3D environment for a desert (Figure 6).

Figure 7. Concept art for the tree village. Source: Original art.

The concept for Iris' village was a giant hollow tree (Figure 7). The tree would have rooms and homes carved into the tree, and the way one would navigate the village would be walking around the tree on supported bridges. The village would be surrounded by a village wall at the base of the tree, with various exits and entrances to important sites.

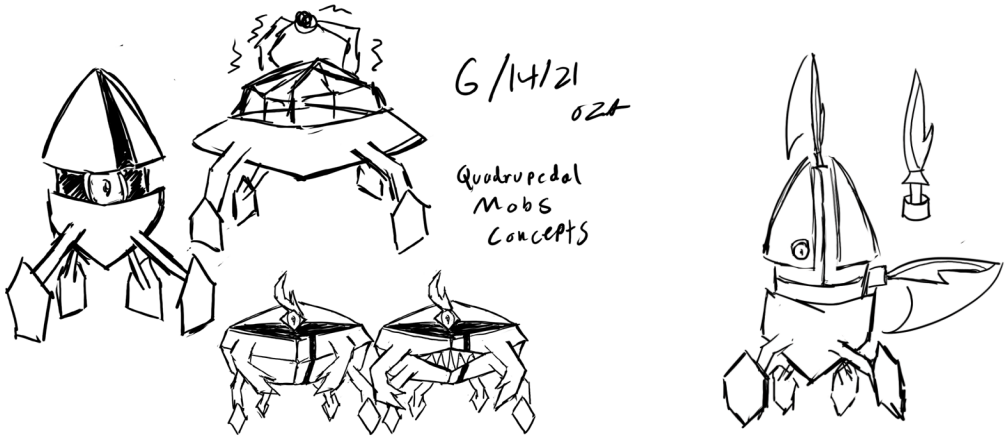


Figure 8. Early concept art for Hepheon drones. Source: Original art.

3.1.2. Enemies

One of the first things we worked on was concept art for the enemies. Beginnning with only a very basic concept of nature vs. machine for the game, and an animation system for walkers, we were able to draw up some basic *Portal*-like drones. These would soon later evolve into the first basic drone concepts, shown in Figure 8.

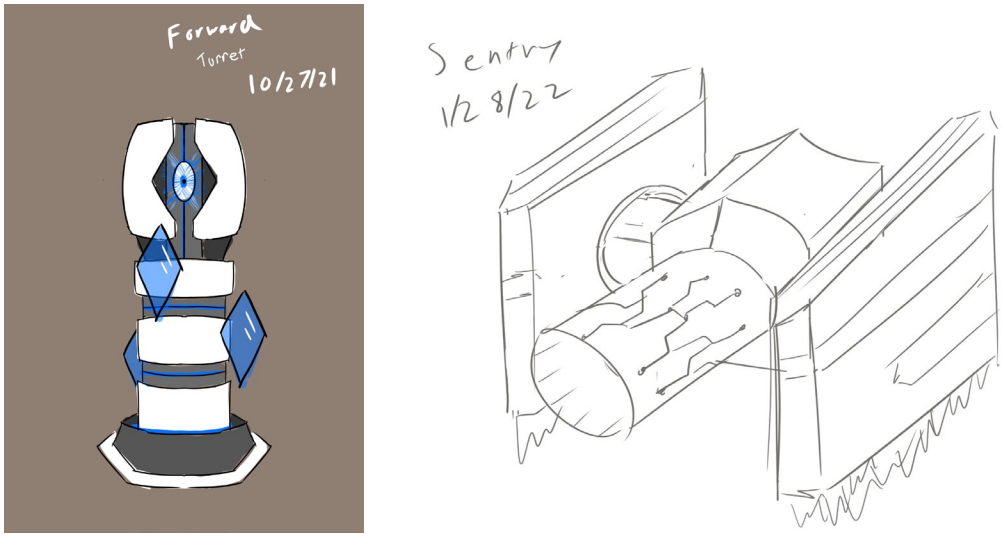


Figure 9. Concept art for turret and sentry drone mobs. Source: Original art.

Other Hepheon enemy mobs took after a similar style, with tracks on segmented sections, smooth outside frames, and sharp thin geometric features (Figure 9).

Figure 10. Concept art of a Geomoprh turtle. Source: Original art.

Geomorph enemies followed a different design style. Instead of hard-edged robots, geomorphs are organic creatures with a touch of Geo corrupting/morphing them. As such, the high-tech industrial look of the drones was replaced by organic curves and animal-like features. The first Geomorph design was based on a turtle (Figure 10).

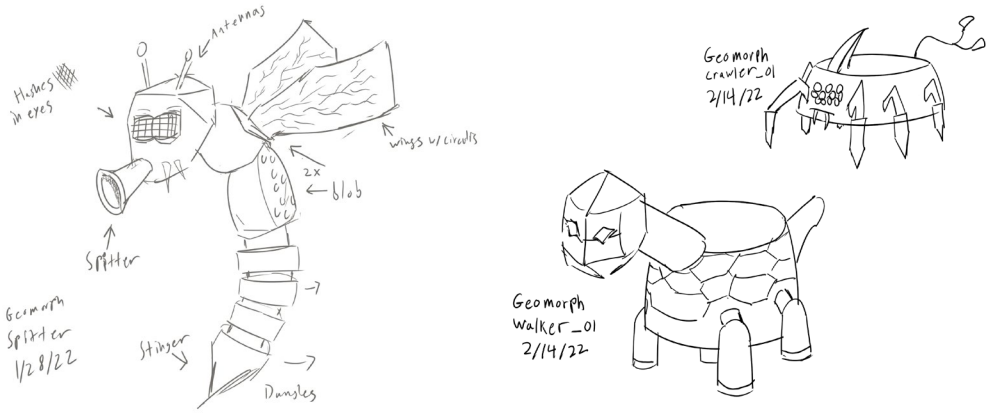


Figure 11. Concept art of Geomorph Spitters, Walkers and Crawlers. Source: Original art.

The flying Spitter Geomorph was an easy addition, but we also wanted a ground-based unit. Two concepts we considered were the Walker and the Crawler (Figure 11). These were inspired by the animation system detailed in Section 5.8.1. We decided to go with the Crawler for its faster-looking design.

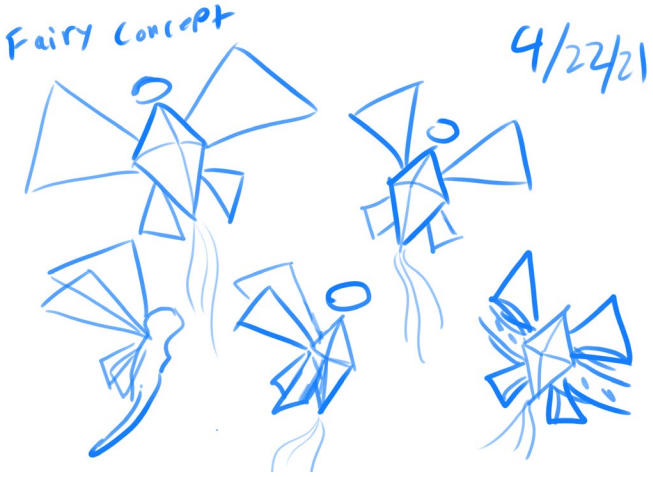


Figure 12. An early set of Iris concepts. Source: Original art.

3.1.3. Iris

The concept for Iris, originally codenamed Fairy, changed heavily over the project. One of our early concepts for Iris resembled a geometric version of Navi from the *Legend of Zelda* series (Figure 12).

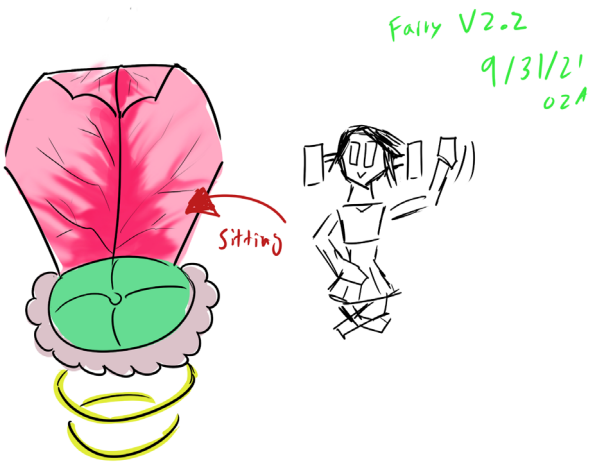


Figure 13. A second-iteration Iris concept. Source: Original art.

To make Iris take on a more natural look, we took inspiration from cactus flowers. We made a design where a more humanoid Iris was sitting on a flower-like hovercraft in order to stray away from technological underground themes (Figure 13). The hovercraft had a huge petal as a backrest in order to make a stronger silhouette, but we were still unhappy with the design because it didn't make sense for the explorer Iris to sit on a throne all the time.

Figure 14. Refined Iris concept art. Source: Original art.

To address concerns about Iris' design, we came up with a block-like design which later developed into Iris' next iteration (Figure 14). The cactus flower colors remained, but the problem of Iris looking too robotic resurfaced.

Figure 15. Colored Iris concept art. Source: Original art.

We were happy with Iris' more independent nature, but to address the issue of her design being too robotic, we replaced her mechanically jointed body with more human-like features. We also gave her a staff for which she uses to adventure.

Figure 16. Iris in orthographic view. Source: Original art.

After finalizing this design, we created some orthographic sketches to be used for modeling (Figure 16).

3.1.4. Seru and Renegade

In the story, Seru and Renegade share the same base model. It was only after Renegade's life on the surface that he had to change out his parts and change his appearance. So when designing the two androids, we started with Seru's design. Once we had that where we wanted it, we used it to create Renegade's design.

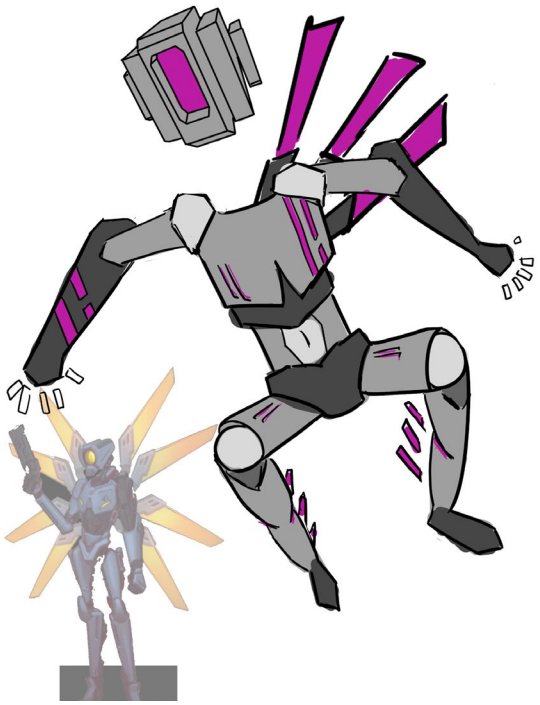


Figure 17. Early Seru concept art inspired by Ultrakill *(bottom left). Source: Original art.*

The androids are heavily inspired by the main protagonist of *Ultrakill* (Figure 17). The wings behind Seru were an early concept of what Geoblades were meant to be, floating energy swords that could be telepathically manipulated around Seru. The head in this concept art was modeled in Blender and traced to better match the 2D style.

A problems with this design was that it wasn't blocky enough. Geo stands for geometry, so we wanted more geometric-looking shapes. We also didn't like the pose, which seemed a bit goofy, so the body went through a full redesign.

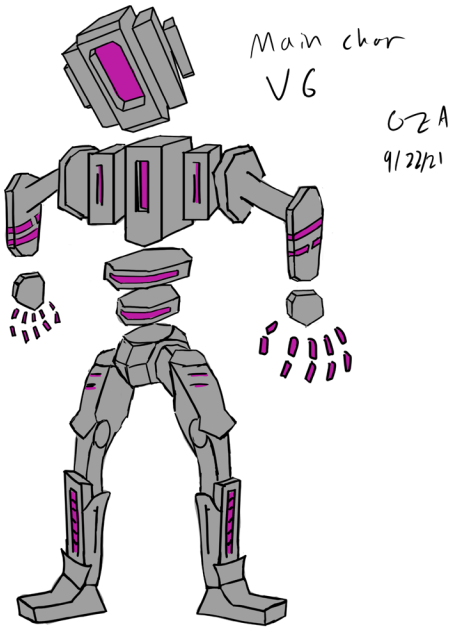


Figure 18. The final Seru concept art. Source: Original art.

The next version of Seru had much more geometric features and a more basic pose (Figure 18). This version also more closely matched the style we had for the head.

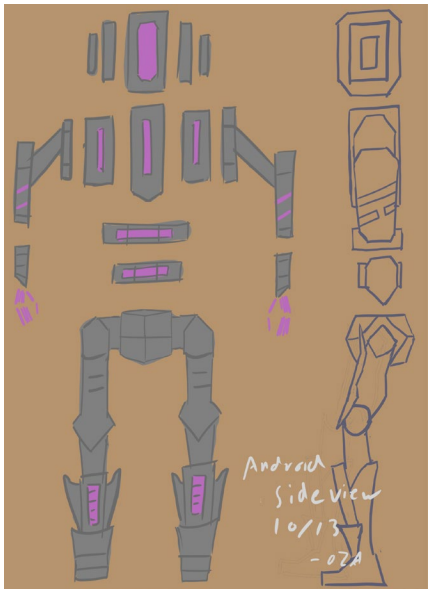


Figure 19. Seru in orthographic view. Source: Original art.

Once we were satisfied with this, we prepared an orthographic version of Seru for modeling, shown in Figure 19.

Renegade's concept was very easy from here. We just needed to edit Seru's concept and make it more worn-out from time and life on the surface.

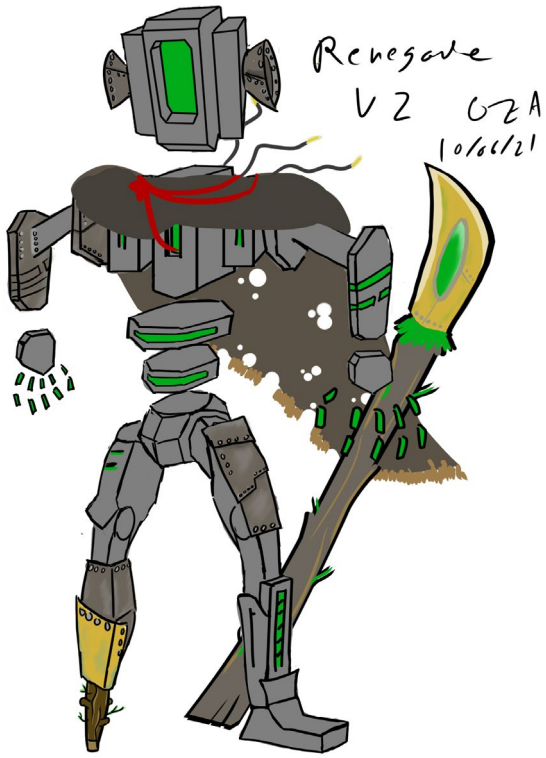


Figure 20. Renegade concept art. Source: Original art.

Renegade's time on the surface has caused him to lose some of his parts that needed to be repaired with materials on the surface, so these pieces are reflected in his design (Figure 20).

Renegade no longer uses Geo as his main energy source. He has instead found a bioenergy that makes him radiate green energy as opposed to the neon-blue of Geo. Renegade is equipped with a cloak and spear.

3.2. Character model workflow

Our character modeling workflow consisted of Blender for modeling and rigging, Mixamo as a source for animations, and Quixel Mixer for texturing.

3.2.1. Modeling

Character modeling was done using Blender, an open-source 3D modeling application. The geometric nature of many of the characters made this more a matter of lining up simple shapes rather than sculpting complex geometry. This kind of workflow is not necessarily more or less difficult than a traditional character modeling workflow. However, it presented a unique set of challenges.

Working with so many separate objects within one character model introduced a few artifacts when exporting to an FBX file. The transformations of each part of the character needed to be baked individually before exporting. This did however make rigging an easier process as weight painting was largely a non-issue since bones could be parented with entire objects.

While the game is geometric in style, using extremely simple geometry result in edges that are too sharp. In the real world, there's no such thing as a perfectly sharp edge, so leaving such edges on our model would result in an unnatural look even if the graphics were stylized.

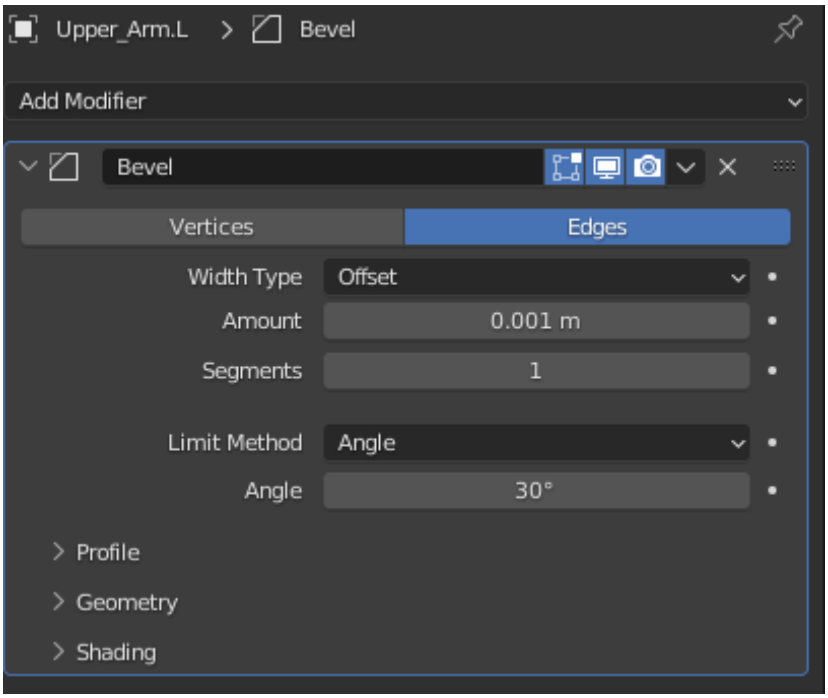


Figure 21. Interface for Blender's bevel modifier. Source: Screen capture.

The bevel modifier in Blender was used to add detailing to the edges to make them appear softer (Figure 21). Using a procedural modifier to bevel edges reduced a lot of the workload that would have been spent baffling every edge in the model individually.

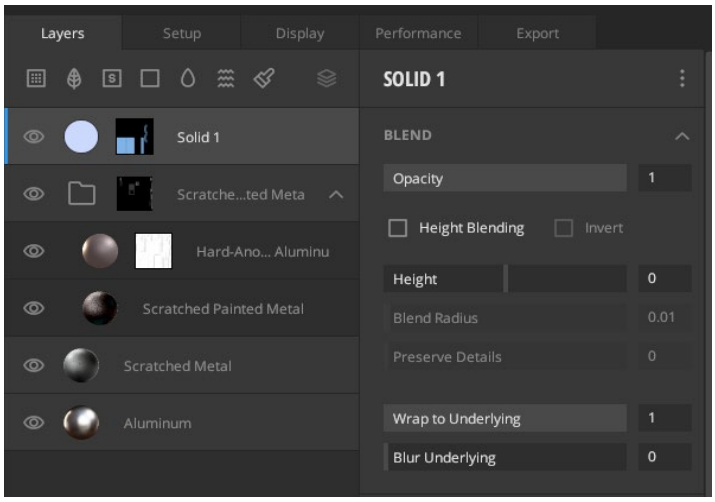


Figure 21. Example of material layers used in Quixel Mixer. Source: Screen capture.

3.2.2. Texturing

We generated textures for our models using both Substance Painter and Quixel Mixer. Both of these programs provided access to a standard library of materials along with a number of procedural materials (Figure 21).

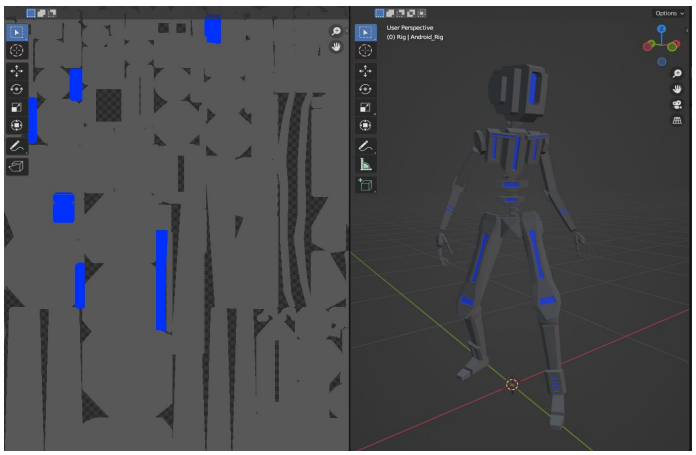


Figure 22. Color ID Map generated using Blender's cycles renderer. Source: Screen capture.

In order to assign different materials to different parts of the model, Quixel Mixer required a color ID map. This ID map was generated in Blender using the cycles renderer (Figure 22). Materials could be assigned to pieces of geometry within Blender due to the game's geometric art style.

3.2.3. Rigging/animation

We knew before modeling that Mixamo would be a great resource for production of our humanoid characters. Mixamo is an Adobe product that provides free use of hundreds of humanoid character animations. This was important to take into consideration because our characters would have to be modeled to be compatible with Miximo's rigs.

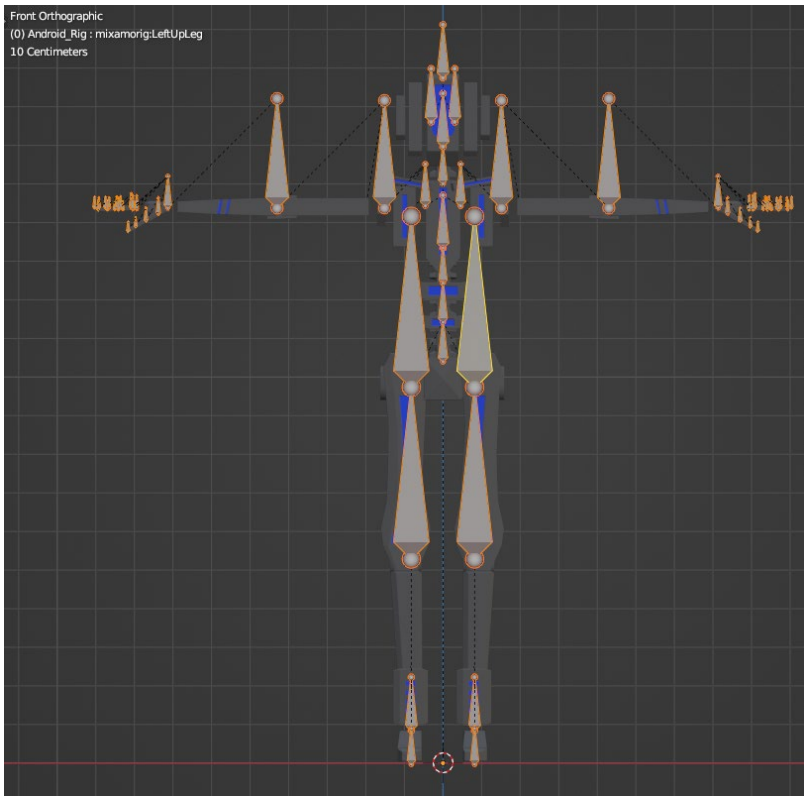


Figure 23. Lining up a character model with the Mixamo rig in Blender. Source: Screen capture.

After importing animations from Mixamo onto the rigs, some cleanup work was done within Blender to fix artifacts baked into the animations (Figure 23). Specifically, the root bone tends to drift over time. Extreme cases had to be corrected to avoid interference with gameplay by misaligning the character model from its collider.

3.3. Environment and prop workflow

The workflow for environment and prop production was largely the same as the workflow for character models, with a few key differences.

3.3.1. Kit-bashing

Kit-bashing and greyboxing were used to create concepts for in-game levels. Quixel Megascans was a great resource for environmental assets. This online service provided materials and models equipped with full LODs. Because of this, the concept art of several of our environments could be made with the same assets we would use in the final version in-engine. This approach worked better for natural environments because most of the available assets were scanned from nature.

While there was still an abundance of assets for artificial structures, most of those were of contemporary buildings. This was a far-cry from the high-tech, almost ritualistic, style of Hepheon architecture.

3.3.2. Modeling

Modeling for props was done in Blender and Fusion 360. Fusion 360's parametric workflow lent itself to modeling hard surface objects. The models from Fusion 360 were then exported into Blender to do some cleanup on the geometry. This workflow worked better for props because it allowed for rapid prototyping changing parameters such as size and tolerances where models fit together.

3.3.3. Texturing

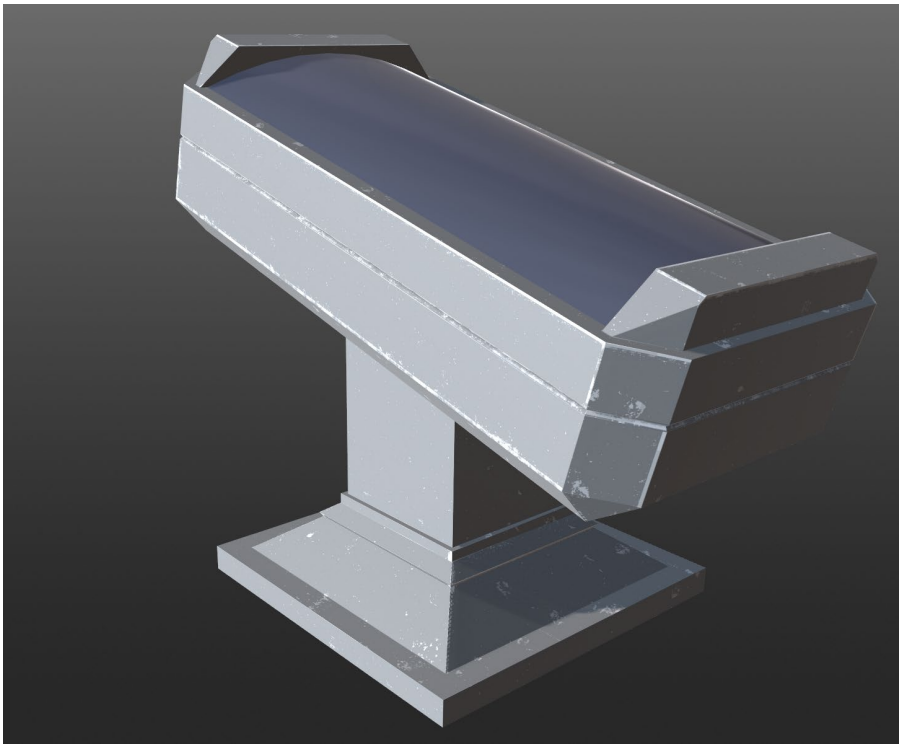


Figure 24. Quixel Mixer's procedural texturing on a stasis pod. Source: Screen capture.

Like the character models, Color ID maps were generated in Blender's cycles renderer and imported into Quixel Mixer for texturing. As seen in Figure 24, Quixel's procedural textures were used to add hints of erosion along the edges of some of the props to indicate their age.

4. Audio

4.1. Music composition

4.1.1. Inspiration

As soon as we got the basic locations of the game ironed out, we decided that we wanted the music to present a dichotomy between the digital and organic worlds, represented by The Geode and the surface respectively. The idea was that as the player progressed between areas, the music would become more and more organic in its instrumentation, transitioning from purely electronic sounds to a middle-ground made up of instruments like piano and electric guitars, eventually transitioning to a fully-realized orchestra. Throughout these changes, we wanted the music to evoke a mysterious and grandiose vibe that would grow to become more epic as the game progresses and the player begins to discover the truth of the world they find themselves in.

With this in mind, some of the music we looked at for inspiration were the soundtracks from games like the *Halo* and *Portal* series, anime such as *Attack on Titan*, *Jujutsu Kaisen*, *Made in Abyss*, and *Dr. Stone*, as well as *Game of Thrones*, and *Tron: Legacy*. All of these soundtracks have elements of both the electronic and orchestral sides that we wanted to capture.

Some of the tracks that we felt did a particularly good job at blending these two styles in the tone that we were going for were “Under Cover of Night” from *Halo: Combat Evolved*, “Eren Zahyo” and “Apetitan” from *Attack on Titan: Season 2*, “Self-Embodiment of Perfection” from *Jujutsu Kaisen* and “The Night King” from *Game of Thrones: Season 8*.

4.1.2. Concept music

Once we had our inspiration, work began on concept music. With the game having a focus on audio, it seemed important to get some musical ideas floating around as quickly as possible, even if they did not end up being used in the final product.

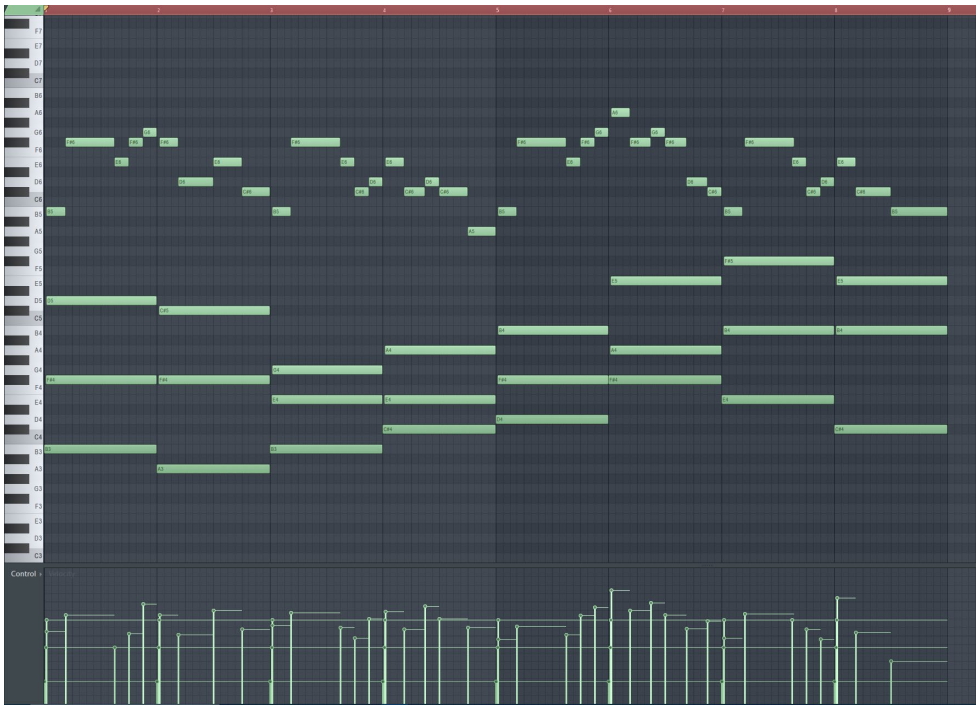


Figure 25. An early concept of the game's main melody on piano. Source: Screen capture.

The process began by writing a piano rendition of an early idea for a main theme using the digital audio workstation FL Studio. It consisted of a fairly simple chord progression and a heroic, yet mysteriously somber melody that we thought fit the tone of the game that we had in mind fairly well (Figure 25).

A key feature of the melody shown above is that many of the notes are not perfectly aligned to the grid. One of the major weaknesses of electronic music is that it can often sound too robotic. No matter how musically skilled, real humans cannot play rhythms exactly as they are notated in sheet music. Statistically speaking, they're always going to be at least a little bit off beat, even if by only a few milliseconds. Similarly, the velocities of the notes, indicated by the bars at the bottom of the piano roll, are likewise varied, as musicians don't play every note at the same volume either.

Introducing these sorts of imperfections helps give a more organic feel to the music, and the techniques were used quite a bit in the game's soundtrack. Nevertheless, we did want some of the music tracks to give off a more robotic feel in order to lean into the stylistic dichotomy described in the previous section.

Manipulating the instrumentation, tempo, and arrangement of this initial melody led to some interesting variations. For example, after transposing it to a different key, switching to a minor mode instead of a major one, and fiddling around with the chord progression, we arrived at the melody of the first track that plays in the game after Seru wakes up. It's played on a very soft, muted piano at first as Iris, representing the outside world, is introduced. Shortly afterward, as Seru and Iris begin to explore the ancient yet simultaneously futuristic-looking underground facility, the melody is doubled by a synthesizer. Conversely, taking the same melody contour, mixing things up a bit, and orchestrating it with brass and strings led to the foundation of a much different, yet obviously connected, piece that ultimately became associated with the surface.

4.1.3. Instrumentation and leitmotifs

In addition to different types of instruments representing different locations, we also used various instruments to give individual characters their identity. For example, Seru is characterized musically by keyboards, electric guitars, heavy percussion, and later in the game, brass, while Iris is characterized by woodwinds, harps, piano, and bells. Meanwhile, strings indicate the presence of Renegade, and modern, almost "dubstep-like" electronic sounds represent The Core.

Characters and the locations they are associated with, as well as specific ideas or situations, are often represented with musical leitmotifs. These recur throughout the game in such a way that they are not exactly the same every time they appear, but are clearly and distinctly identifiable. A great example of a famous leitmotif is "Binary Sunset" from *Star Wars: A New Hope*, also known as "The Force Theme." This theme occurs throughout the *Star Wars* films and related media when various characters who are either a part of or associated with the Skywalker lineage make use of the Force. When used in this sort of way, leitmotifs help to create a sense of context, purpose, and interconnectedness for what the listener is experiencing, and we employed them to make players feel as though the game's world is truly alive.

4.2. Sound Effects

4.2.1. Gathering sound effects

In order to create a rich library of sound effects to be used in the game, we combined field-recorded sounds with ones we found online through sources like freesound.org and various Humble Bundle packs. However, these sources were rarely used "out of the box." We combined and layered different sounds together in various ways to get new and interesting results.

Before creating these custom sound effects, we needed a way to keep track of which ones were needed. For this we made an audio asset list, a spreadsheet that listed which individual sound sources would comprise each final sound effect as well as which animations, objects, and/or in-game actions they would be associated with. We also kept track of important information about each sound source such as their file name, filetype, bitrate, and whether it was recorded in mono or stereo.

4.2.2. Audio editing workflow

With everything organized, in order to edit the individual audio sources into final sound effects, we made use of Reaper, a digital audio workstation similar in some ways to FL Studio, which was discussed in Section 4.1.2. The main difference between the two applications is that while FL Studio specializes in music production using MIDI patterns, Reaper is better suited for dealing with multi-track audio recordings with no distinct meter.

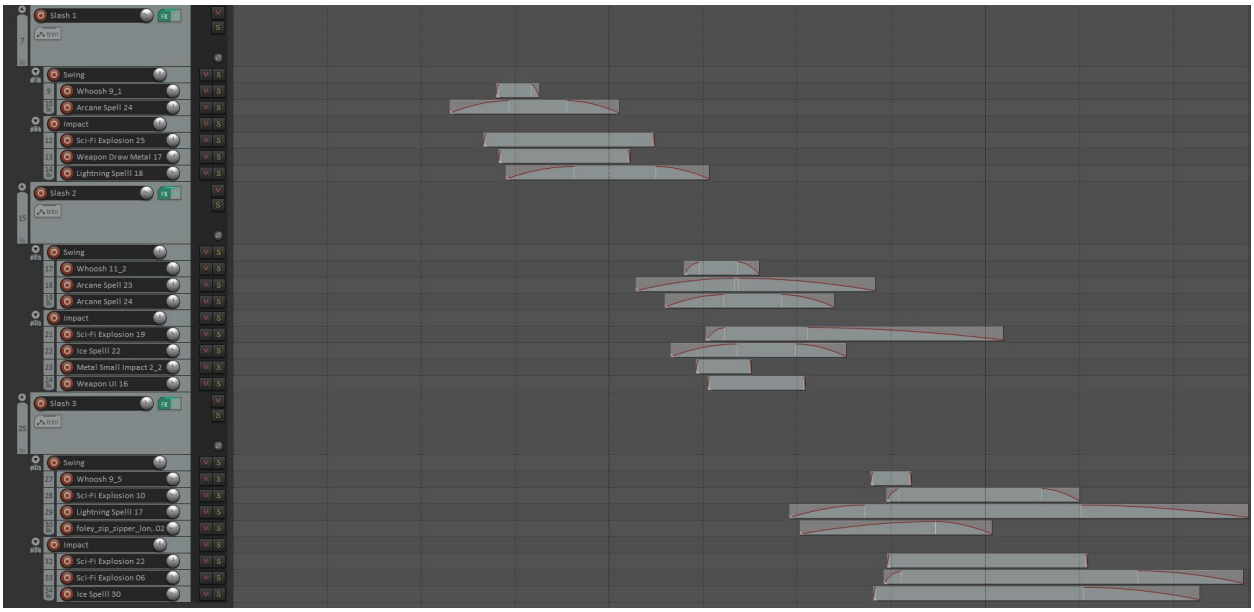


Figure 26. Attack sound compositing in Reaper. Source: Screen capture.

As alluded to above, one of the keys to engineering interesting sound effects is to use multiple, well-mixed, layered sounds. It's often impossible to capture the exact sound profile you're going for in a single recording, especially if you're trying to create other-worldly effects. An example of this layering technique can be seen in Figure 26, which showcases the timeline of the Reaper project for the sounds that Seru's weapon, the GeoBlade, makes when he attacks.

Notice in the figure above that we used a hierarchical structure for organizing all of the different sound sources. This allowed us to easily do things like apply effects to all sound sources that make up a single effect.

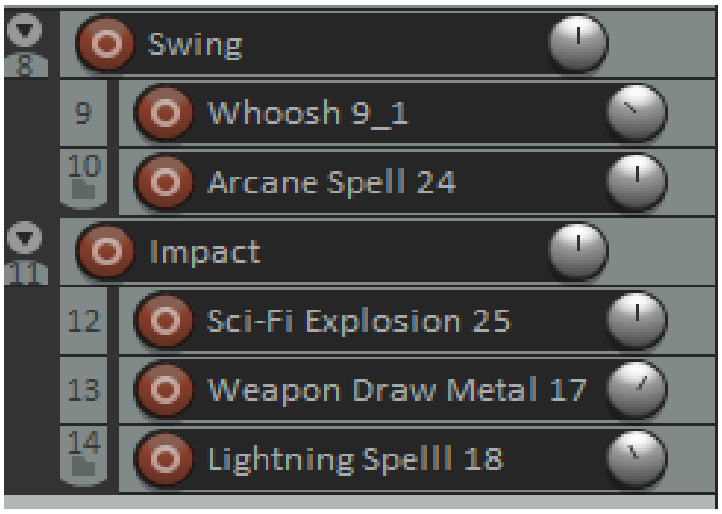


Figure 27. Subtrack mixing in Reaper. Source: Screen capture.

As shown in Figure 27, we created a separate track for each of the three attack sounds in the combo, with two subtracks within each of those for the swing and impact portions of the sound. This way, we could easily solo the appropriate parent tracks in order to render only one sound effect at a time.

We wanted the *GeoBlade* to sound like a blend between a high-tech energy weapon and a more traditional sword. A total of 23 unique sound sources were used to create these particular effects, several of which came from the Humble Bundle packs mentioned in Section 4.2.1 above. A lot of the sounds that make up the "energy" side of the weapon are actually from a collection of magic, elemental spell sounds. Layering things like the ice and lightning spells from this pack with subtle explosions, wind impulses, metal impacts, and zippers shaped the final sounds. Properly balancing the volume of each sound source relative to one another while ensuring that none of them cause the overall volume to peak above 0 dB is vitally important.

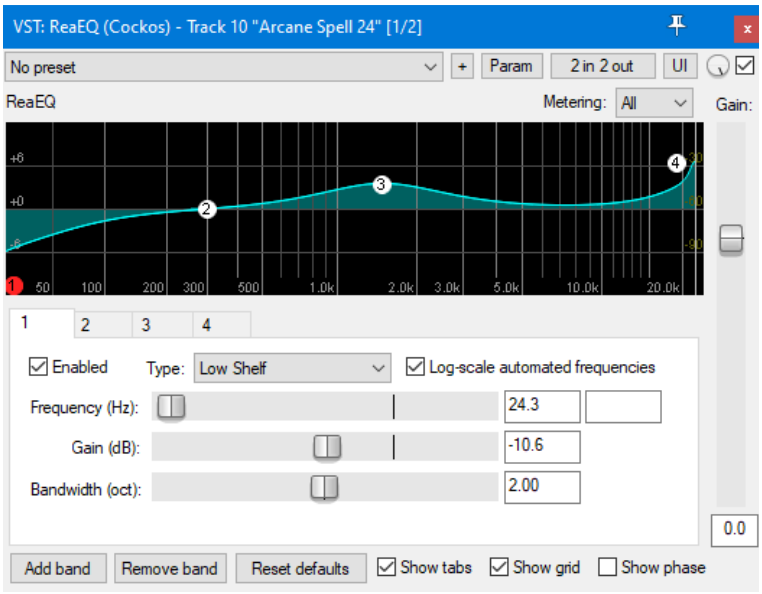


Figure 28. Equalization in Reaper. Source: Screen capture.

Volume is not the only important consideration. If a mix sounds muddy, then it's likely that two sound sources are competing for the same frequency range. EQ, or equalization, is a technique that lets engineers attenuate or boost the volume of certain frequency bands. Figure 28 shows an example of EQ being used on one of the "spell" sound sources mentioned above. It's used to roll off the low-end frequencies below 300Hz (also known as a high pass filter,) boost the midrange frequencies around 1.5kHz slightly to increase clarity, and give a larger boost to the highs in the 20kHz range to add some sizzle.

Some other commonly used effects are compressors and reverb. Compressors are used to "squash" an audio signal and reduce its dynamic range. It effectively reduces the volume of each point along a waveform above a certain threshold by a given threshold. Reverb is used to prolong the decay of a sound source through simulated reflections.

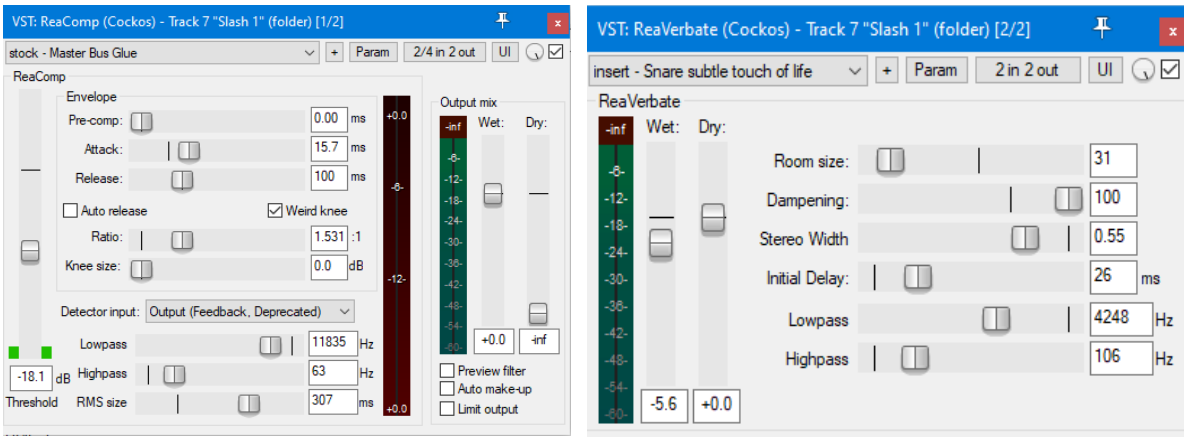


Figure 29. Compression and reverberation tools in Reaper. Source: Screen capture.

Examples of both compression and reverb being used on the parent mixer track for the first attack are shown in Figure 29. This technique of passing several child sound sources through a common, shared compressor is called glue compression, and is used to make a collection of sounds intended to make up a single whole sound more cohesive, or "glued together."

4.3. Recording dialogue

4.3.1. Process

We knew early on that we wanted the game to have fully voice-acted dialogue. Since, at its core, the game placed a major focus on audio, we felt that it was only natural to include it.

Almost as soon as we had the idea for the character of Iris, who was known as "Fairy" at the time, we thought of having Luke's sister voice the character. We decided to voice the three remaining main characters (Seru, Renegade and The Core) ourselves, as we were all interested in voice acting.

Each team member recorded scripted demos for all of the three characters. With the help of our advisors, we ultimately decided that Luke would voice Seru and Renegade, and Cameron would voice The Core. From there, we would have friends, family, and ourselves fill in the remaining minor roles that we planned to include.

Figure 30. Shure SM7B microphone setup for recording dialogue. Source: Original photo.

Dialogue was recorded into Adobe Audition using a Shure SM7B microphone connected to a Cloudlifter preamp and a Scarlett Focusrite 4i4 audio interface (Figure 30). Since the SM7B is a dynamic microphone with fairly low gain, an auxiliary preamp is almost a necessity in order to get a clean and clear signal at an appropriate volume. A large windscreen was used to prevent sibilance and plosives. Unfortunately, we were not able to record in a properly treated environment, so we relied on using things like mattresses and blankets to try to absorb some sound and prevent reflections. We did our best to limit background noise as well, going so far as to temporarily turn off computer fans while recording.

Ensuring proper gain staging was very important for recording. We needed enough gain to be able to clearly hear the voice of the talent, but we also needed to ensure there was enough headroom for effects like compression and limiting. We found that the sweet spot was approximately 50% gain on the interface in combination with the preamp, but had to make fine adjustments here and there, depending on which character was speaking and how they were performing. For example, whispered lines generally are going to need more gain, while lines that are yelled will require far lower gain in order to avoid overloads. Voice talent can also compensate for this by moving closer to and further away from the microphone. Both techniques were used in tandem to get the proper volume for each line, with particular consideration for

ensuring lines that are part of the same sequence are generally around the same volume, unless there's a good reason why they should not be.

We used Audition to edit the dialogue, unlike the sound effects, which were edited using Reaper as discussed in Section 4.2.2. We felt Audition was easier to work with when dealing with a single waveform representing one audio source. While Reaper is great for creating complex, layered sounds from multiple sources on multiple channels, we felt that Audition is better suited to fine adjustments of individual waveforms. To organize all of the dialogue files we created a hierarchical structure of *Level–>Section–>Character–>Line Type–>Line*.

Most of these subcategories are self-explanatory, but the "Line Type" warrants further description. We wanted to keep a "checkpoint" system of sorts for our dialogue files so that we could return to an older version if we felt a foundational change should be made late in the editing process. We created three different folders within each character's set of lines for a given section, Raw, Cut, and Final. The Raw folder contained the raw recording sessions. The Cut folder contained lightly edited and normalized versions of the takes that we planned to use for each line, and the Final folder contained the fully-edited lines with all effects processing applied.

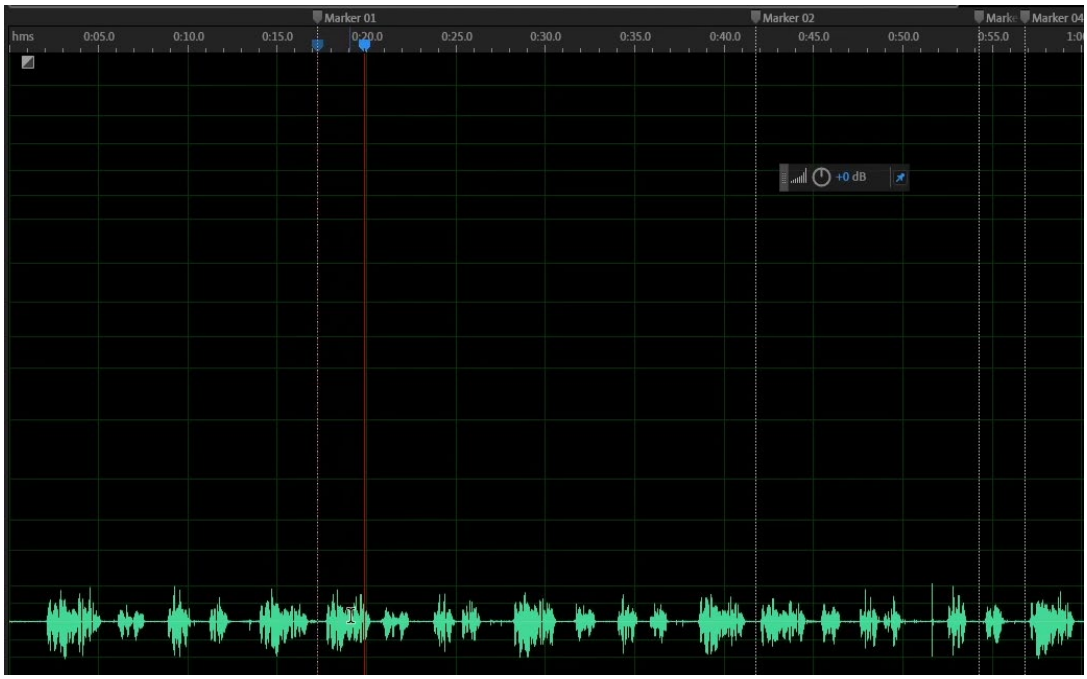


Figure 31. Marking promising sections of a line in Adobe Audition. Source: Screen capture.

4.3.2. Editing workflow

A lot of work needed to be done to the recorded audio before it was ready to be used in-engine. After saving a copy of the raw line in the appropriate Cut folder described above, the first step in the process was selecting the best take from a recording session that could be up to several minutes long. Ideally, there would be one single take where the voice actor nailed the line, but sometimes we would have to use individual words from different takes and combine them together in a way that sounded natural. Finding the best take or combination of takes involved listening through the whole file and placing markers at areas of interest (Figure 31).

Figure 32. Noise reduction in Adobe Audition. Source: Screen capture.

After making it all the way through, we would listen back to each of the marked sections and construct the Cut version of the line, connecting words and phrases together as necessary. Before cutting out all of the unused takes, we would look for a long section of silence that could be used to generate a noise profile for use in a noise reduction process via an FFT (fast Fourier transform), shown in Figure 32.

After completing this process, extraneous takes would be removed and we would move on to surgical adjustments of the peaks of the audio. Various consonant sounds like *Bs*, *Ps*, *Ds* can come across as very harsh in a recording even if every measure is taken to prevent them. In cases like these, the volume of specific sections of a waveform will often be edited down to make them a bit easier on the ears.

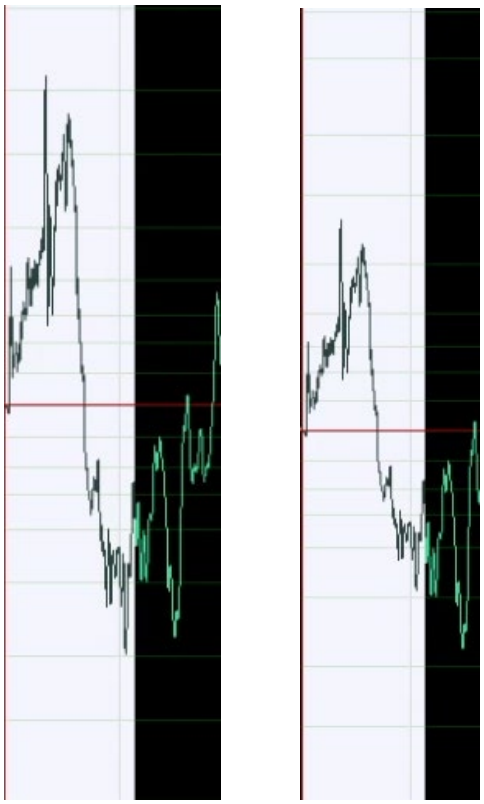


Figure 33. Comparison of a waveform before (left) and after (right) editing a harsh sound. Source: Screen capture.

Figure 33 shows an example of the difference between a particularly harsh B sound in one of Seru's lines on the left, and the volume-adjusted version on the right.

It wasn't just harsh sounds that needed to be toned down. Often a word or phrase within a line would be significantly different in volume than the rest of the line. The voice actor may have said one word particularly loudly compared to the rest of a phrase, or maybe they accidentally got a bit too close to the microphone, or perhaps we just needed to take the word from a different take in which the average volume was higher. Any of these situations could lead to a section of the waveform having a higher average volume compared to the full line. Conversely, the opposite of any of those situations could happen, resulting in a section of audio that was comparatively too quiet.

Outliers like this need to be adjusted to prevent unwanted results when normalizing audio. The way normalization works is by increasing the gain of the peak amplitude to a specified target (most often 0dB) and maintaining dynamic contrast by adjusting the gain of every other part of the waveform relative to the highest peak. It was important that we took the

time to make sure the lines would sound right once normalized, as it is much harder to correct abnormalities after the fact. This is without doubt the most time-consuming part of the editing process.

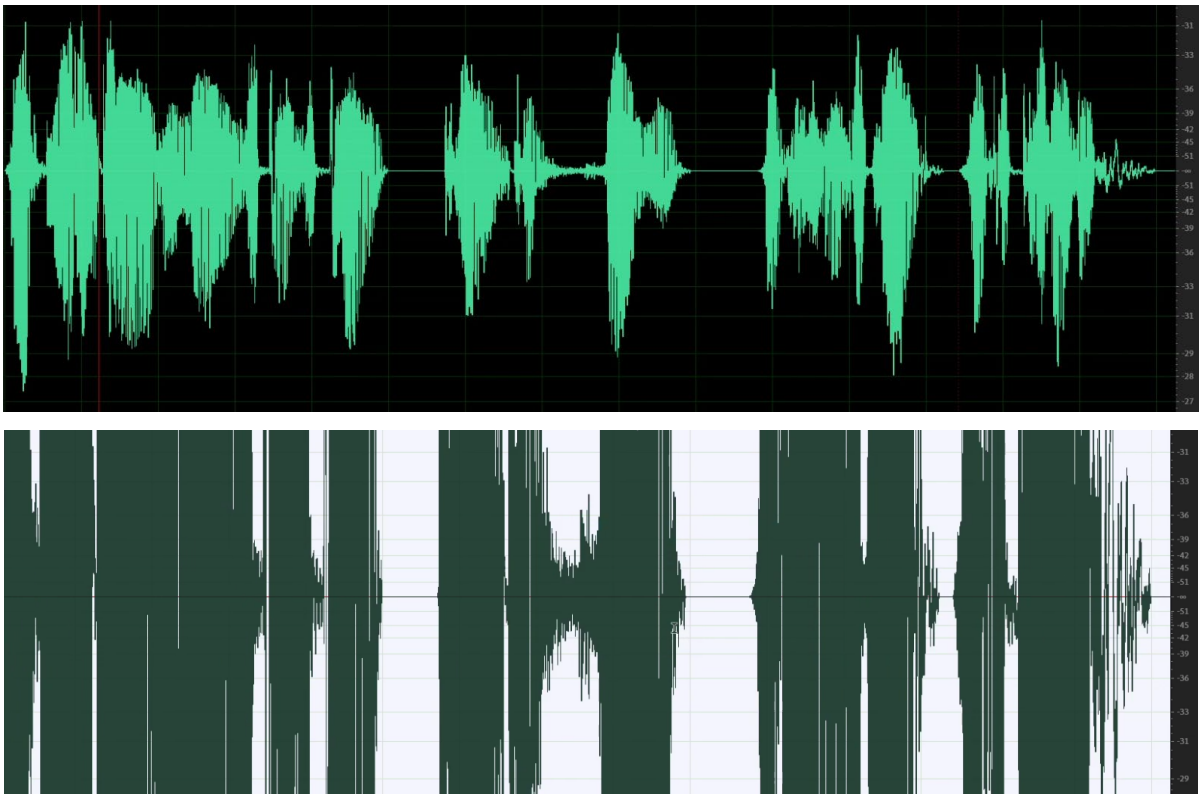


Figure 34. An original waveform (above) normalized to -6dB (below). Source: Screen capture.

Once we were happy with a line, the last step before we could start applying effects was to normalize the audio. Through some experimentation, we found that a value of -6dB tended to work quite well for our purposes, providing us with still enough headroom for the effects processing that would come next. Figure 34 shows an example of a waveform before and after normalization to -6dB at the same scale.

With this step complete, we saved another copy of the line into the Final folder and began working on effects processing and finishing touches. At this point, effects chain presets associated with each character could usually be applied. The preset chain for a given character would include a compressor and EQ, a de-esser for reducing sibilance, and a hard limiter at the end of the chain for ensuring the gain never exceeded a maximum threshold.

Figure 35. The effects chain preset for Seru's lines. Source: Screen capture.

The specific effects chain for Seru's voice (Figure 35) also included a slight delay on the left channel, a phaser and flanger in order to achieve the robotic timbre of his voice, and a pitch shifter to raise the voice just a bit higher.

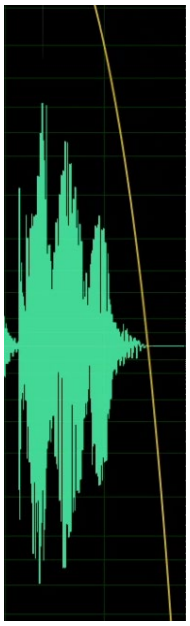


Figure 36. Example of a fade-out applied to a line. Source: Screen capture.

After the effects chain is applied, the final step in the process is to add a subtle fade-in and fade-out to the line so that it feels as if it starts and ends naturally. An example of this can be seen in Figure 36.

5. Technical implementation

5.1. Game engine

After minimal deliberation, Unity was the obvious choice for our game engine. All project members had at least some experience with it, and it was compatible with a broad variety of required plugins. While engines like Unreal offer more robust tool sets for 3D games, they don't offer the large community actively supporting Unity.

5.2. Wwise audio engine

5.2.1. Setup

Knowing that this game would make heavy use of audio to create dynamic soundscapes, we chose to employ a separate, third-party audio engine to help facilitate this. While Unity has basic audio support, a lot more interesting things can be done with dedicated audio middleware, such as spatialized audio, interactive music, state-based triggering of different sound effects, and easy parameter variation and randomization.

Our two main choices for an audio engine were Wwise and FMOD. While FMOD is easier to get started with, Wwise offers many more features, with the tradeoff of its free tier being rather limiting, forcing developers to jump through hoops with their licensing model.

Ultimately, we decided to go with Wwise as our audio engine. Audiokinetic, the developers of Wwise, offer a Unity plugin that provides integration with Wwise. Setup was a bit particular in that, among other complications, we discovered that Audiokinetic requires developers to register projects on their website in order to qualify to use Wwise for longer than its one-month trial period. During this trial period, the engine is limited to only 200 samples – not nearly enough to support the extensive audio systems and number of dialogue lines we planned to use.

Though meticulous, the registration process was not overly difficult, and after having the project approved by an Audiokinetic staff member, we were finally able to begin using Wwise for the duration of the project without fear of losing any work.

Getting Wwise to integrate seamlessly was also a bit of an ordeal, as it required some careful version matching of Unity, Wwise, and the Wwise plugin for Unity. Once we found a stable configuration, we were able to do things like browse and generate Wwise soundbanks from directly within Unity, and easily assign Wwise objects to serialized properties within the Unity inspector. The most important thing though, was that the plugin provided us with a way to make calls to the Wwise engine from within Unity's C# scripts.

5.2.2. States and events

At its core, Wwise is a local server written in C++ that exposes an API allowing developers to make calls to it from game code. The Unity plugin serves as a C# wrapper for this API.

The primary use case of Wwise is triggering dynamic sounds influenced by the current state of the game world when a specific in-game action occurs. Wwise makes this possible mainly through two aptly named mechanisms: state groups and events. State groups allow you to create a collection of various possible states, each with their own uniquely identifiable name, for a single property. This system functions very similarly to how enums do in many programming languages.

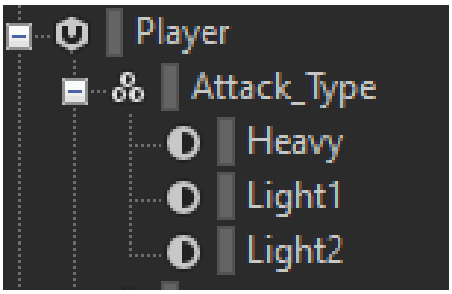


Figure 37. The Player_Attack state group in Wwise. Source: Screen capture.

Figure 37 shows how a state group is defined in Wwise. This particular example is our *Player_Attack* group, used to keep track of the type of attack a player is currently making so that the appropriate sounds can be played.

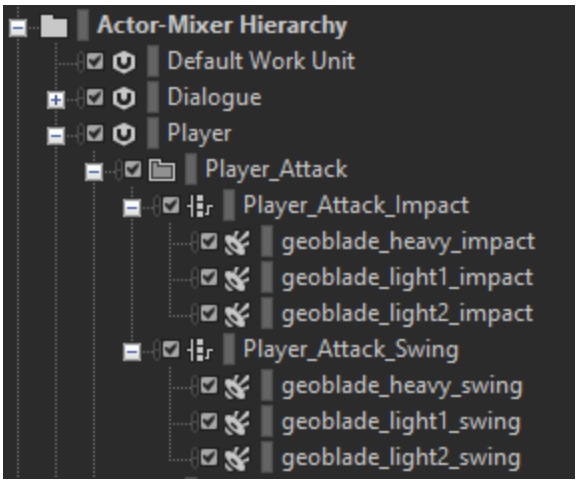


Figure 38. The Player_Attack virtual folder and its children within the Wwise Actor-Mixer Hierarchy. Source: Screen capture.

State-based sound cues are often used to trigger switch containers. A switch container allows you to assign a state group and define which children of the container will be played, depending on the current state of that group. Figure 38 shows an example from *GeoBlade's* Wwise Actor-Mixer Hierarchy.

The *Actor-Mixer Hierarchy* is where almost all sound effects reside in Wwise. Within a hierarchy, you can have multiple work units to keep sounds associated with a specific part of the game isolated. Work units can be individually loaded and unloaded to improve performance, so individual work units are usually prepared for specific areas of a game map. In *GeoBlade,* there are only a few sounds that need to be able to play at any given time, and they are all bound to the player. Because of this, we created a Player work unit that is loaded when the player object is created and unloaded when it is destroyed.

Under the Player work unit, there is a virtual folder called Player_Attack which has two switch containers as its children, Player_Attack_Swing and Player_Attack_Impact. This virtual folder is ignored when parsing the hierarchy tree and is used purely for organizational purposes. In effect, any children of a virtual folder can be thought of as actually just being a child of their least distant ancestor that serves a functional purpose, in this case the work unit. Both of the containers each have three children, one sound for each attack in the three-hit combo.

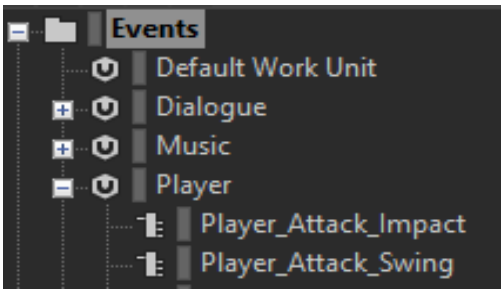


Figure 39. The Player_Attack_Impact and Player_Attack_Swing Wwise events. Source: Screen capture.

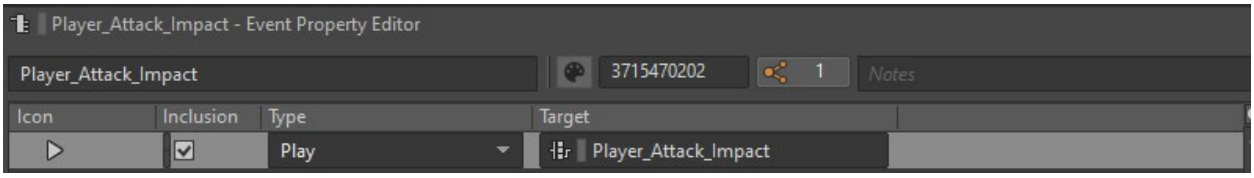


Figure 40. Player_Attack_Impact event targe. Source: Screen capture.

We also have two events of the same name which trigger their respective containers, shown in Figures 39 and 40. These happen to be the same sounds used as examples of audio creation in Section 4.2.2. This demonstrates the entire pipeline of getting sound effects into a game right up to the point that they're actually made triggerable from within the code.

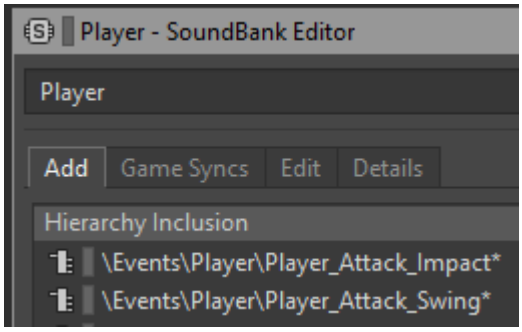


Figure 41. Player SoundBank event inclusion. Source: Screen capture.

Once implemented in Wwise, all that's left is to assign the events to a SoundBank as seen in Figure 41, generate it, and load it into Unity.

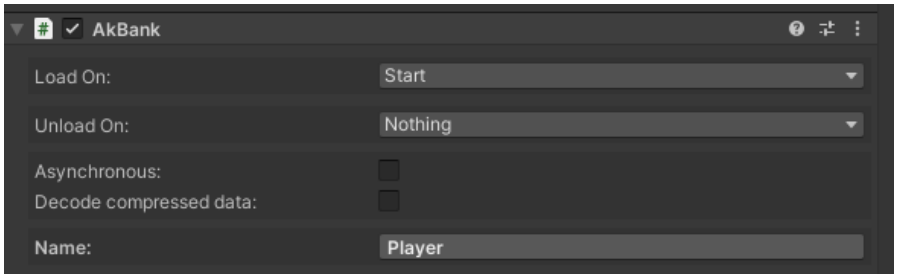


Figure 42. Loading the Player SoundBank in Unity. Source: Screen capture.

In this case, we assign the attack sounds to the Player SoundBank which is loaded on game start using an AkBank component on the Player GameObject, and never unloaded until the game is exited (Figure 42).

Many sounds are not needed throughout the entire game. By organizing sounds into banks based on when and where in the game they're triggered, they can be dynamically loaded and unloaded to improve performance. However, for this particular situation, it makes sense to keep the bank constantly loaded, as player attack sounds can be triggered at virtually any time during gameplay.

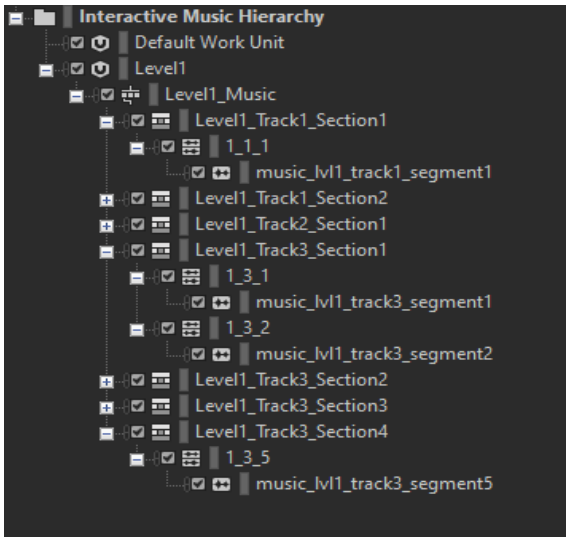


Figure 43. The Wwise Interactive Music Hierarchy. Source: Screen capture.

5.2.3. Interactive music

Similar to the Actor-Mixer Hierarchy described above, Wwise projects also contain an Interactive Music Hierarchy. This allows you to define how the state of the game informs what music is playing, as well as how the transitions between different sections of music behave. Figure 43 shows a work unit named Level1 within the Interactive Music Hierarchy.

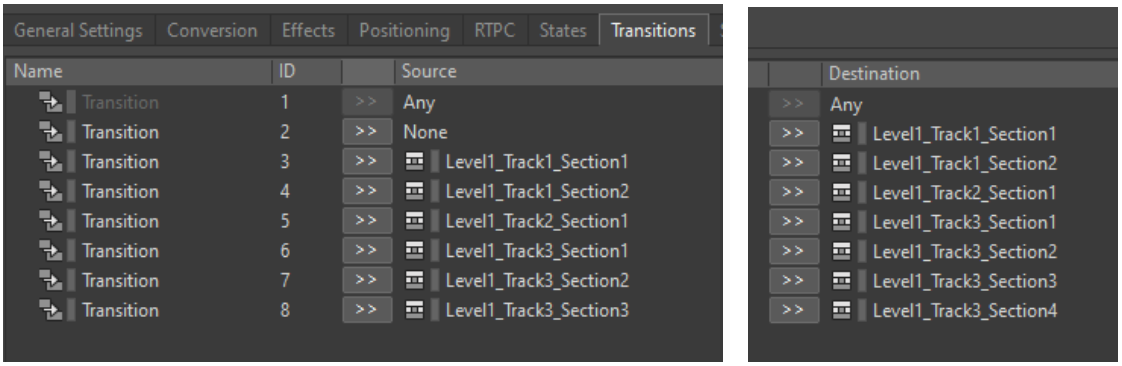


Figure 44. Music switch container transitions. Source: Screen capture.

The top level of the hierarchy under the work unit is a music switch container, a specialized version of the switch container described in Section 5.2.2. On top of the standard functionality of a switch container, music switch containers allow the definition of transitions between their children, music playlist containers, as seen in Figure 44.

Within a music playlist container, there must be one or more music segments, which have some special functionality, but were essentially used as wrappers for music tracks for our purposes. These transitions allowed us to do things like have the current section of music crossfade to the next section if they're part of the same piece.

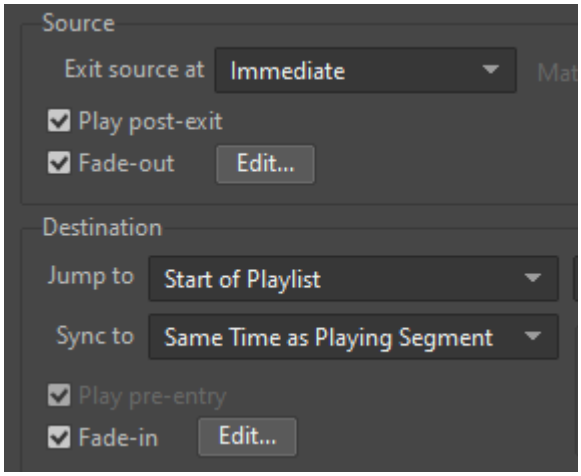


Figure 45. An example transition between two sections of music.
Source: Screen capture.

An example of a crossfade between two sections of the same piece can be seen in Figure 45. Both sections have the same form; the only variance is their instrumentation. In this case, on a transition we exit the source playlist immediately but allow its post-exit phase to fade out while the destination playlist's pre-entry phase fades into the start of the playlist in time with the source segment. The subjective effect is that the common elements of the two sections appear interrupted, while the differences blend together cleanly.

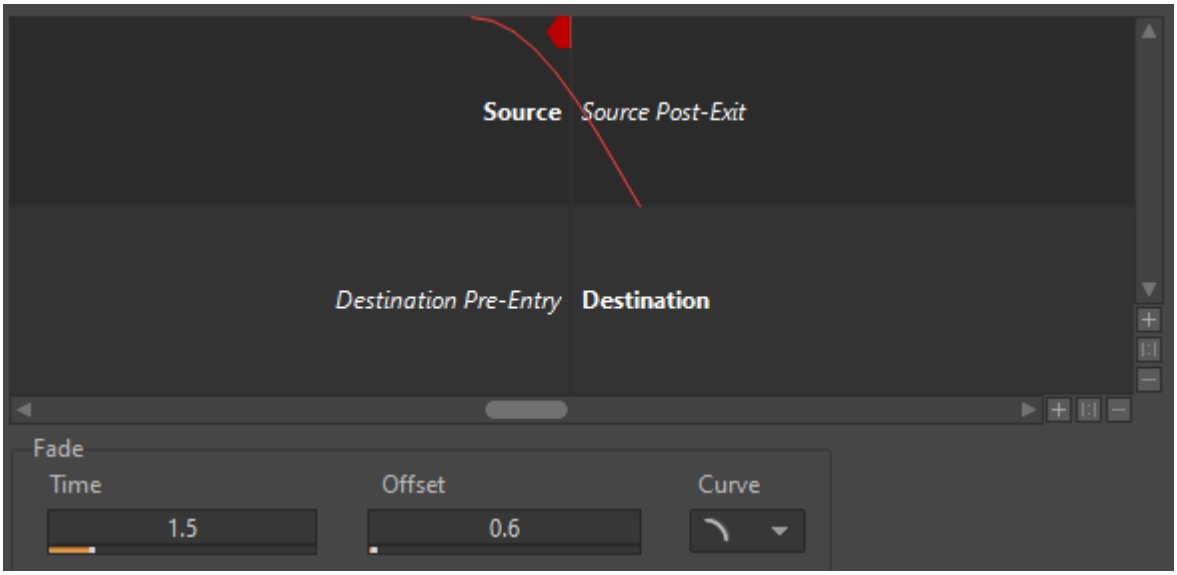


Figure 46. The fade-out curve for the transition shown in Figure 45. Source: Screen capture.

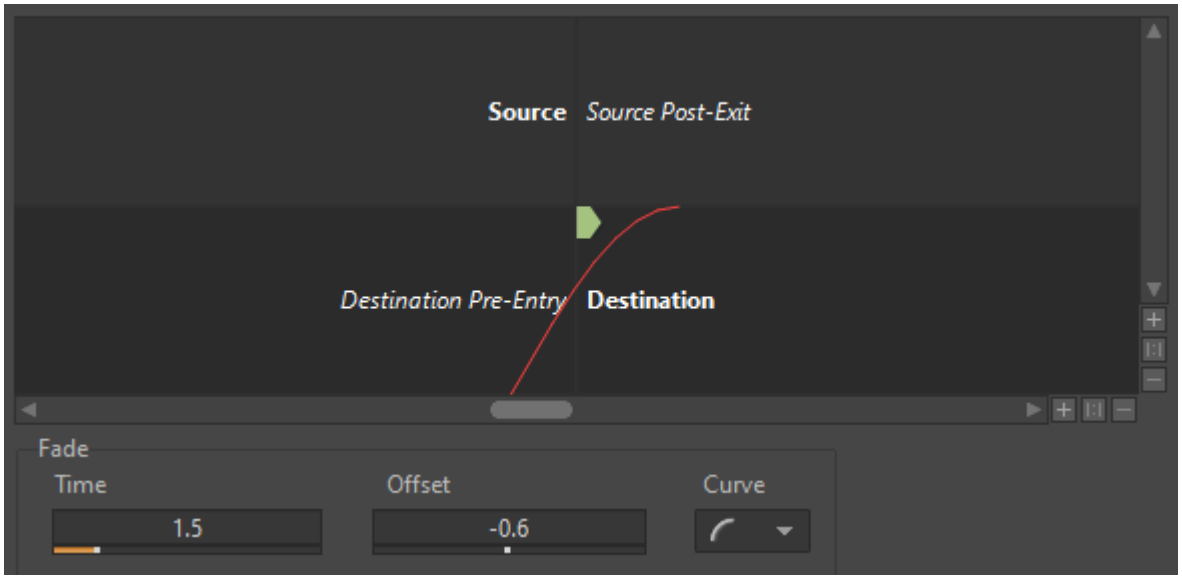


Figure 47. The fade-in curve for the transition shown in Figure 45. Source: Screen capture.

Additionally, custom fade curves can be created to fine-tune the timing and volume of the two ends of the crossfade. Figures 46 and 47 display the fade-out and fade-in curves that we used for this particular transition.

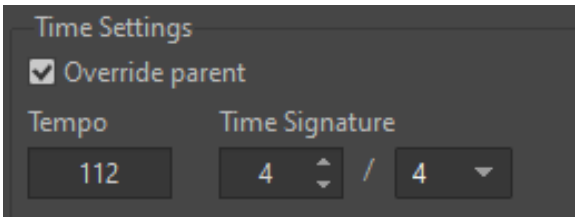


Figure 48. Tempo and meter settings of a music playlist container. Source: Screen capture.

Conversely, to transition between two different pieces of music, we can have Wwise wait for the next bar, beat, or any other desired unit of metric measurement before switching to the next section. This is done by properly setting the tempo and meter of each music segment so that Wwise is able to keep track of how far along the playback is from a musical perspective, as shown in Figure 48.

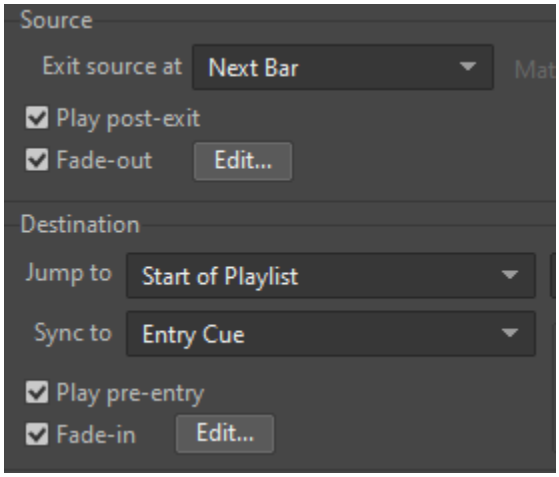


Figure 49. An example transition between two different pieces of music. Source: Screen capture.

Figure 49 illustrates a transition in which the source playlist exits at the next bar and the destination playlist syncs to its entry cue.

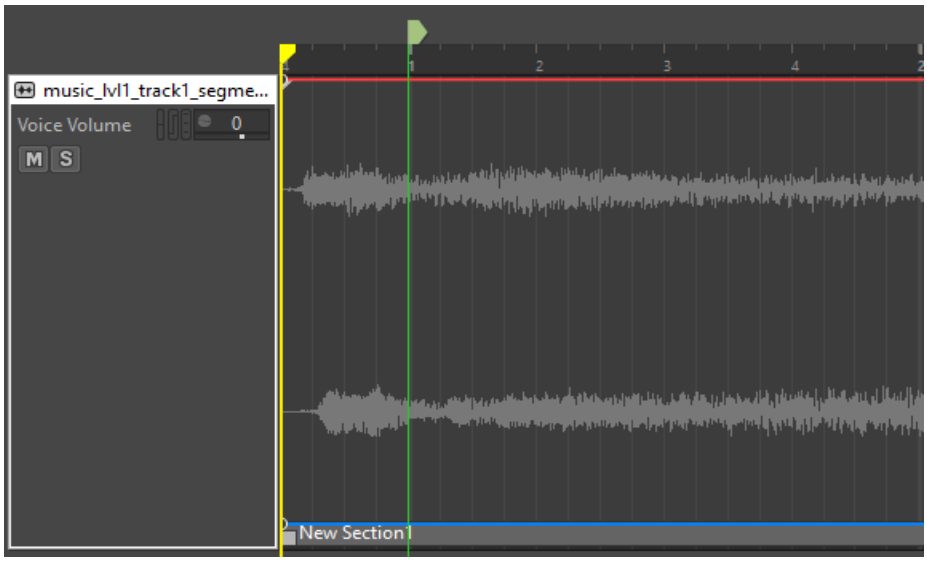


Figure 50. A music segment with an entry cue. Source: Screen capture.

Entry cues, as well as exit cues, are used to mark where and how a particular music segment should loop. These are especially useful for tracks that have a pickup measure or intro that should only play once but not on repeat, or tracks with a stinger that should play only on the last repeat. Figure 50 shows an example of a music segment with a one-bar pickup measure before the entry cue.

Entry cues are limited to the length of the post-exit phase of the source playlist in order to make state-based transitions simpler. If, for example, you need one music segment that is eight bars long to play before switching to another one and looping it indefinitely, that's when music playlist containers are useful. This type of behavior is static and doesn't require a state change, so you can define a continuous sequence of individual segments, each with their own looping behavior.

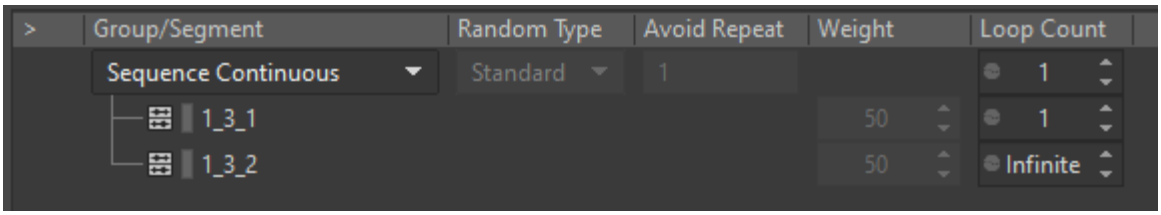


Figure 51. An example of a sequence of music segments within a playlist container.
Source: Screen capture.

Figure 51 is an example of the situation described above. One segment is triggered a single time before allowing the following segment to loop until it is interrupted by a transition.

With all of this properly configured, sections of music can be triggered from the game code identically to sound effects without having to worry about the inner workings of the transition setup, so long as the appropriate events are loaded through a SoundBank in Unity, as explained in Section 5.2.2.

5.3. Houdini Engine

5.3.1 Licensing

We decided to use the Houdini Engine in order to provide a framework with which to create procedural geometry and artwork.

The process of licensing Houdini Engine was especially challenging. Houdini is split up into two programs for this purpose: Houdini Engine and Houdini. Houdini is a standalone tool used to make 3D assets procedurally. Houdini Engine takes these assets and renders them in a game engine (in our case, Unity) with the same procedural parameters that were used in Houdini. These assets are exported from Houdini to Houdini Engine through a Houdini asset file.

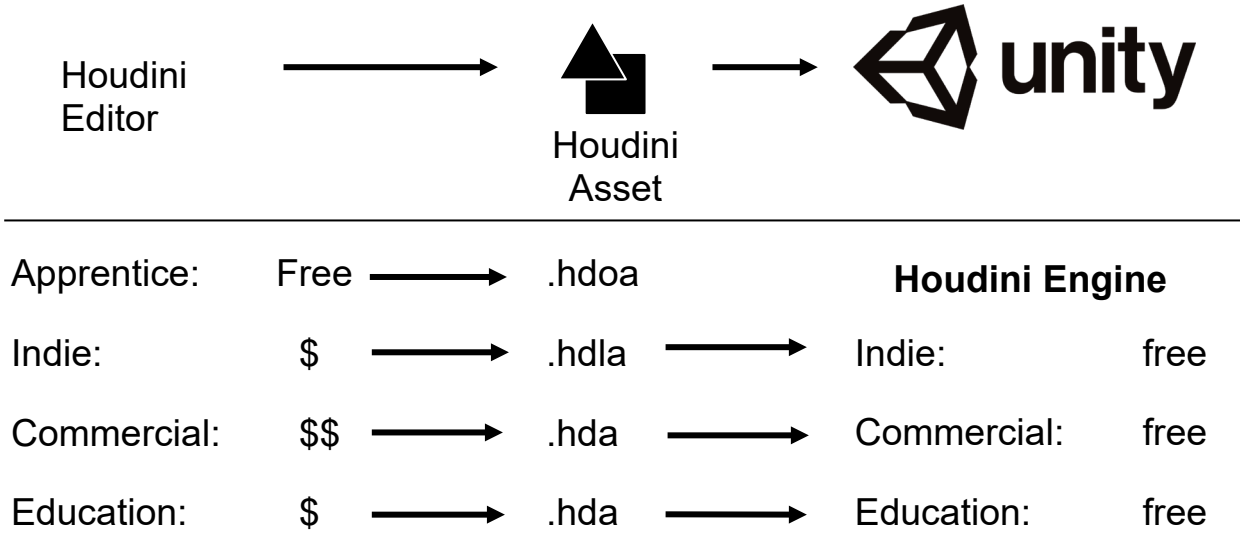


Figure 52. Houdini Engine licensing model. Source: Authors.

These asset files caused our team some trouble with licensing. Houdini comes in four different license types: Apprentice, Indie, Commercial, and Education (Figure 52). For Houdini standalone, all but the apprentice license cost a significant amount of money. For Houdini Engine, all of these licenses are free. Houdini has a policy that Houdini editor and Houdini engine must have the same license in order to be compatible. This is enforced by having each Houdini editor export a different file type. Each version of Houdini Engine can in turn only import their corresponding file type. As shown above, Houdini apprentice exports an HDOA file, Houdini Indie exports an HDLA file, and Houdini commercial exports an HDA file.

Since there is no apprentice license available for Houdini Engine, we had to get Houdini files in the form of HDLAs or HDAs in order to use them within Houdini Engine. This seemingly put an impassable roadblock in our use of Houdini. We would have to purchase a license that cost hundreds of dollars in order to create files for Houdini Engine. After a few days of searching, happened upon a tool that SideFX (the company that distributes Houdini products) provides for free to transition files from the HDOA format to the HDLA format. This discovery essentially gave us a free route for converting our apprentice files into Indie license files.

5.3.2 Procedural assets

Houdini uses a mixture of nodes and a proprietary language called VEX with a syntax similar to C to enable the creation of procedural assets. These tools allowed us to take functions commonly found in 3D modeling software (such as extrusions, sweeps, and vertex transforms) and implement them non-destructively. These actions could then be driven by parameters provided to the Houdini asset wherever it's rendered.

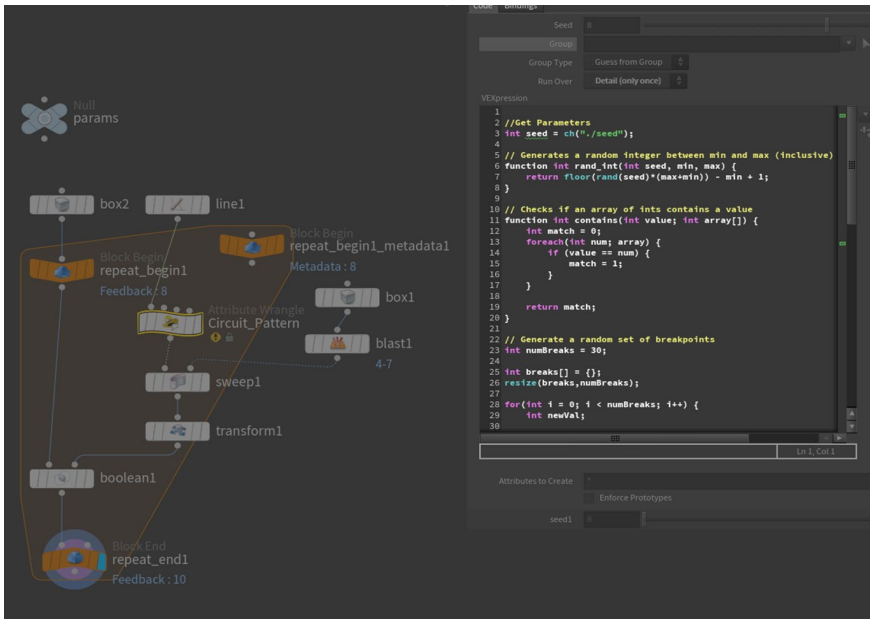


Figure 53. Houdini's non-linear node editor and VEX language used to create a procedural wall pattern. Source: Screen capture.

While several assets were created using Houdini's procedural tools, only a limited set used Houdini Engine to remain procedural within Unity. One such application was detailing for walls. Using booleans to cut out sections of wall in a circuit pattern would have been extremely time-consuming to model by hand, so Houdini was used to automate the process. As shown in Figure 53, some VEX code was written to generate lines of vertices that would turn by 45 degrees at random points along their length. A piece of square geometry was then swept along these lines and use to cut away pieces of geometry from a base mesh.

A section of a wall generated from this process is shown in Figure 54 below.

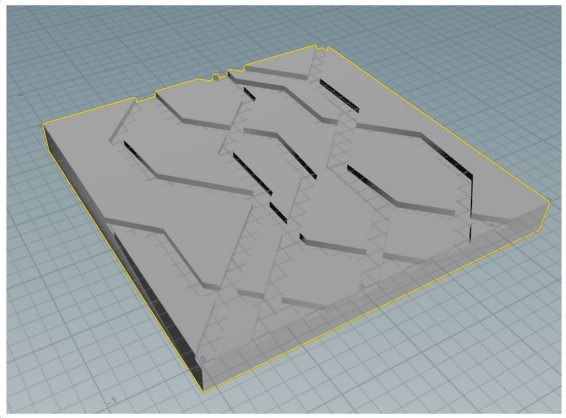


Figure 54. A resulting wall segment from our circuit pattern generator. Source: Screen capture.

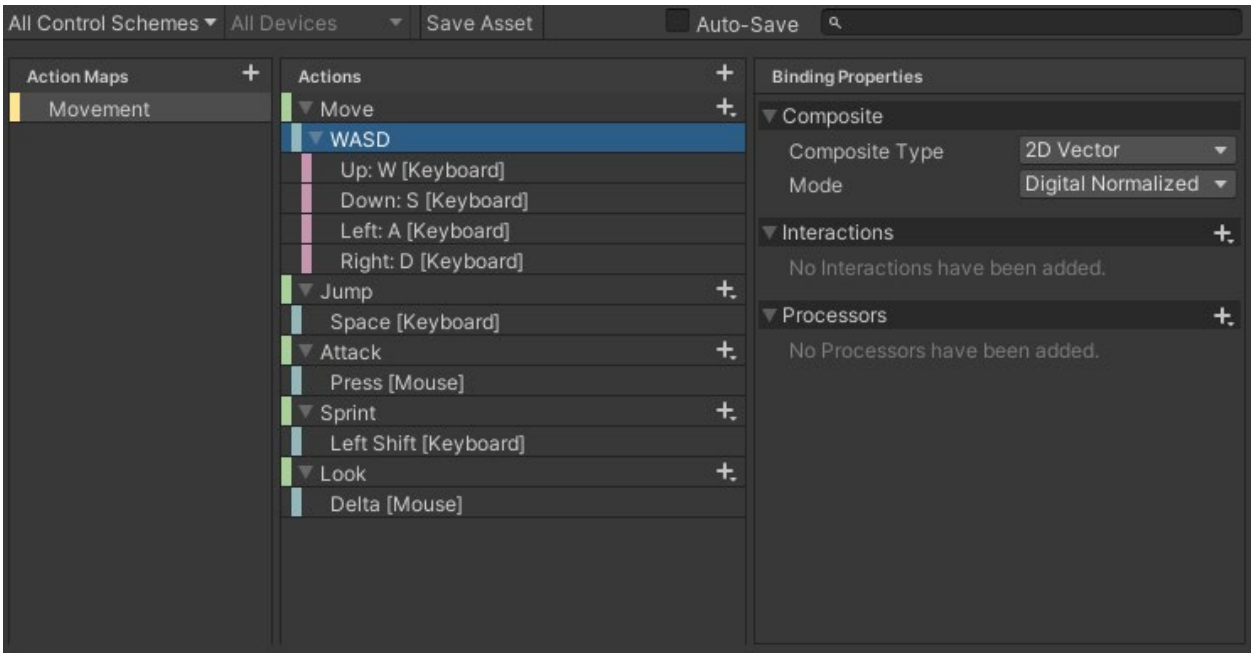


Figure 55. Control Scheme Editor within Unity. Source: Screen capture.

5.4. Player Controller

5.4.1. Input system

Unity has two systems for handling user input: the input manager (legacy) and the input system (new). Unity's new input system adds some features that automate some of the boilerplate work that the old system left to the developer (Figure 55). These included small parts of process input like normalization, dead zones, differentiation between holding and tapping, and several others. These all took the form of interactions and processors, two types of layers that are processed on top of the raw input signals. The new input system also allowed for controls to be handled in-code completely ambiguous of the control-scheme the player was using, allowing for quick changes to in-game controls without refactoring code.

The old input manager, by contrast, did not implicitly handle any of this. Developers would often rely on paid third-party tools to add such boilerplate features. It used strings for each input that would have to be referenced as constants within the code. For all these reasons, we opted to use Unity's new input system for the most part. We did, however, end up handling input with both systems as some legacy features still used the old input manager.

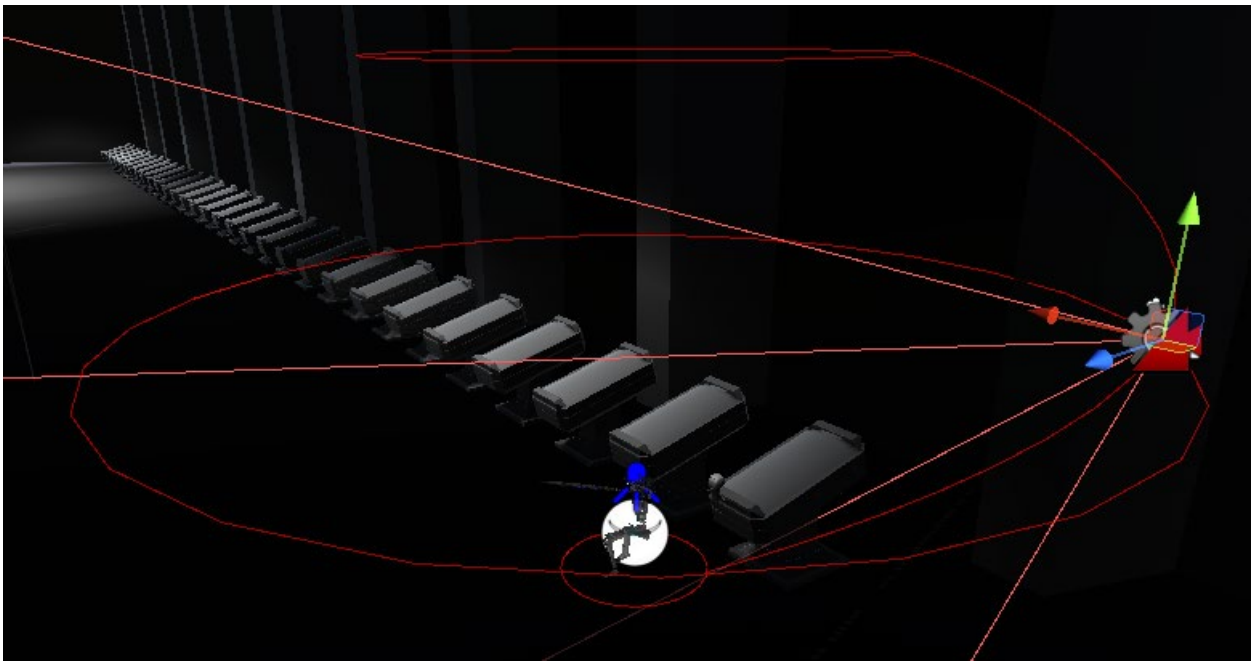


Figure 56. Cinemachine Freelook Track. Source: Screen capture.

5.4.2. Camera controls

The camera controls were created using the Cinemachine FreeLook camera (Figure 56). Since this was a tool from earlier in Unity's history, it used Unity's legacy input manager for camera movement. It was one of the reasons we had to include build support for both the input system and input manager.

Typically in third person games where the camera orbits the player, The camera's orbit is not actually perfectly spherical. The path of camera followed would depend on the kinds of environments that the player would be traversing. Since our game took place in interior spaces with more horizontal room than vertical, the pathar camera would follow was a more squashed horizontal ellipsoid. The Cinemachine Freelook machine for your camera made tuning the shape a lot easier by allowing us to separately adjust the camera's orbit radius between a bottom, middle, and top section. It would also handle transitioning between these sections smoothly.

Some machine also came out of the box with camera damping. This meant that, when the camera started or stopped moving, cinemachine would have the camera accelerate or decelerate rather than immediately changing its velocity. During our Alphafest playesting session (see section 6.1), we received feedback that this system actually acted to the detriment of the camera's responsiveness. Since the camera was controlled by mouse input, players expected the

speed of the mouse and the speed of the camera to be one to one. The camera damping, while making the camera movement more cinematic, undermined this expectation making the camera feel less responsive. Simply removing damping made a major impact in the responsiveness in most parts of the game.

```
if (!controller.isGrounded) {
    _velocity.y -= gravity * Time.deltaTime;
}

controller.Move(finalVel * Time.deltaTime);
```

Figure 57. Factoring deltatime into player movement. Source: Screen capture.

5.4.3. Movement

Implementing the player movement was fairly standard fare for our third person game. The matrix transformations necessary to make the player move in a direction relative to the camera have been long established in game development. Unity's built-in player had a function to handle movement and gravity. However, we opted to create our own for more granular control over how the player. This caused one small hiccup that only became apparent after testing on different machines: the player's speed was tied to the game's framerate. The delta time had to be taken into account to remedy this issue (Figure 57).

While movement remained largely unchanged throughout development (aside from the regular adjustment of speed and gravity), One major change was implemented later in the process. Players often found the combat to be unresponsive largely due to a lack of feedback when attacking. This issue was remedied in part by limiting player movement while attacking. Allowing the player to move like normal during attacks made the subsequent animations feel unnatural for their corresponding movement. Limiting the player's movement during these attacks both improved the believability of the attack animations and gave implicit hints to the player that they would need to take wind-up time into account when attacking.

Figure 58. Dialogue Manager inspector options. Source: Screen capture.

5.5. Dialogue system

5.5.1. Dialogue spreadsheet

While Wwise has support for a lot of audio-related systems, one thing that we felt could greatly benefit from a custom-built solution was a dialogue system. This system was designed from the ground-up to be flexible and performant–supporting localized, properly synchronized subtitles. We wanted to be able to streamline and automate playing dialogue lines within a sequence, but we also wanted the customizability of being able to control things like how long subtitles stay on-screen and delays between lines in a sequence. Figure 58 above illustrates the flexibility of the system.

```
private void Awake() {
    if (Instance != null && Instance != this) {
        Destroy(obj: this);
    } else {
        Instance = this;
    }
```

Figure 59. Initialization of dialogue manager as a singleton. Source: Screen capture.

Changing the locale to another supported one or toggling subtitles within the UI can be done directly from the dialogue manager component in Unity's Inspector. This component is the one and only instance of the dialogue manager script as it is set up using the singleton design pattern, as seen in Figure 59, so these settings can be configured globally here. This pattern is very useful and we make use of it throughout the game's codebase.

To accomplish the design goals outlined above, we used a spreadsheet to represent lines of dialogue in which each row contains the text to display for the written subtitles in each supported language, an ID to use to trigger the audio with Wwise, as well as various metadata about the line such as the speaker, a reference to the next line in the sequence, any delay that should occur after the line finishes playing, etc.

A section of this file can be seen in Figures 60 and 61 below.

	A	B	C
1	line_id	line_en_US	line_test
12	lvl1_opening_cutscene_iris_08	Great, then let's go!	This is an example of translated text.
13	lvl1_stasis_room_seru_01	What is our destination?	This is an example of translated text.
14	lvl1_stasis_room_iris_01	Hmm. I think I've done enough exploring for today, so we should probably start making our way out of this place.	This is an example of translated text.
15	lvl1_stasis_room_iris_02	It's a little creepy in here to be honest, and, well, I'm kind of lost myself, so I'll leave the navigating to you! You must know your way around, right?	This is an example of translated text.
16	lvl1_stasis_room_seru_02	Actually, my memories of this facility are… scattered. It seems deeply familiar, yet so distant. I will do my best.	This is an example of translated text.
17	lvl1_stasis_room_iris_03	Don't worry, I'm sure your memory will come back soon! You seem like a nice person, so I'll trust you to figure it out!	This is an example of translated text.
18	lvl1_spiral_room_iris_01	Hey, Seru, what are all these doors for?	This is an example of translated text.
19	lvl1_spiral_room_seru_01	They are… I apologize. I cannot recall their purpose.	This is an example of translated text.

Figure 60. Line ID and per-language subtitle text in the dialogue spreadsheet. Source: Screen capture.

D	E	F	G
line_speaker	line_duration	post_delay	next_line
Iris		0	END
Seru	1.5	0	lvl1_stasis_room_iris_01
Iris	7.75	0	lvl1_stasis_room_iris_02
Iris	10.5	1	lvl1_stasis_room_seru_02
Seru	10.5	0	lvl1_stasis_room_iris_03
Iris	7	0	END
Iris	3	0	lvl1_spiral_room_seru_01
Seru		0	lvl1_spiral_room_iris_02

Figure 61. Line metadata in the dialogue spreadsheet. Source: Screen capture.

There are a few things to note about Figure 61. The next_line column contains the text "END" when a line is at the end of a dialogue sequence and no further lines should follow it. Otherwise, it simply contains a reference to the next line in the sequence. This "END" token will become important when we look at the specifics of the line scheduling process. Additionally, as can be seen in the above figure, we initially kept track of the duration of every single line in the spreadsheet and would use manual timeouts to decide when to show the next subtitle. This approach ended up causing a major issue which will be discussed further once we get into how the system interacts with Wwise.

5	lvl1_opening_cutscene_iris_04	Is this how it goes? Maybe likeâ€¦ this?
6	lvl1_opening_cutscene_iris_05	Oh, it worked! Hey, there! Iâ€™m Iris! Do you know who you are?

Figure 62. Encoded special characters in the converted CSV file. Source: Screen capture.

The formatting and full suite of associated features provided by Google Sheets is very beneficial to us when editing this spreadsheet, but it introduces unnecessary overhead when trying to make use of the file in-engine. To solve this, we converted the spreadsheet to a CSV file and created a much simpler CSV parsing script to increase efficiency and ease of implementation (Figure 62).

A notable side-effect of this is the fact that special characters and some punctuation marks – apostrophes and ellipses for example – end up looking rather strange in CSV files. A lack of support for special characters is to be expected, but the reasoning behind simple things like apostrophes being encoded this way made little sense to us at first glance. As it turns out, single quotes (which are visually identical to apostrophes unless special ASCII characters are introduced) are often used to enclose a string in CSV files. This is alright though, as our parser is well-equipped to deal with these discrepancies.

```
public DialogueLine(string text, string speaker, float duration, float postDelay, string nextLine) {
    Text = text;
    Speaker = speaker;
    Duration = duration;
    PostDelay = postDelay;
    NextLine = nextLine;
}
```

Figure 63. Object data model used for dialogue lines. Source: Screen capture.

5.5.2. Implementation

The CSV parser loads in all of the dialogue lines once the game launches with the correct text based on the currently selected locale. If no supported locale is found, it defaults to en_US. If any optional metadata fields are left blank, they are populated with default values. Once all of the lines have been loaded, the dialogue manager constructs objects out of them modeled by a DialogueLine class (shown in Figure 63) and organizes them into a hashmap by their line IDs.

From this point, a dialogue sequence can be triggered from any starting line anywhere in the code. This can happen when the player loads into an area, enters a collision volume, kills an enemy, or pretty much anything else.

Figure 64. An example of subtitles being displayed in-game. Source: Screen capture.

Once the first line in a sequence is triggered, the dialogue manager will display the appropriate subtitles for the correct amount of time based on the duration of the line (shown in Figure 64), wait for the specified post-delay, and then trigger the next line in the sequence.

Unless otherwise interrupted, this process continues until the “END” token mentioned above is supplied by the current line in place of a reference to the ID of the next line. When it’s time to play the next line in a dialogue sequence, the state of a Wwise state group called Dialogue_Line is set to the current line’s ID. Each line ID maps to a corresponding state in this group which will trigger a different audio file. Once the state is set accordingly, the Dialogue_Trigger event is posted to Wwise, passing in the corresponding GameObject of the speaker of the line and some additional parameters needed to register for a Wwise callback which will become important later. The point of doing this is so that Wwise can properly spatialize and attenuate the audio. Any Unity GameObject that contains an AkGameObj component can take advantage of positional audio. Some example code for these two Wwise calls is shown in Figures 65 and 66 below.

```
AkSoundEngine.SetState( in_pszStateGroup: "Dialogue_Line",  in_pszState: lineId);
```

Figure 65. Setting the Dialogue_Line state group. Source: Screen capture.

```
dialogueEvent.Post(PlayerManager.Instance.player,  flags: (uint) AkCallbackType.AK_EndOfEvent, EventCallback);
```

Figure 66. Posting the Dialogue_Trigger event. Source: Screen capture.

One major problem that we found with the dialogue system in our testing was that sometimes lines would play over each other when a new dialogue sequence was triggered before the previous one ended. This would result in behavior where if the player quickly ran through an area faster than the intended pace, they could trigger multiple overlapping dialogue sequences with no ability to stop them until they finished playing fully.

```
private async Task<string> PlayLine(string lineId, CancellationToken token) {
```

Figure 67. Cancellation token propagation within an asynchronous task. Source: Screen capture.

To remedy this, we made use of asynchronous tasks and cancellation tokens provided by the threading module of C#’s standard library. In the re-designed system (Figure 67), the playing of a sequence is handled by an asynchronous task which accepts a cancellation token and

propagates it down to all sub-tasks scheduled from within it, such as playing individual lines and waiting for a specified amount of time.

These asynchronous tasks not only allow the dialogue manager to do the majority of its heavy lifting outside of the main thread, thereby improving performance, but they also give us the ability to cancel a dialogue sequence at any time. For example, if the player enters the second area of the first level before the conversation that begins playing after the opening cutscene ends, we don't want the player to have to wait for the entire dialogue sequence to end. Instead, the line that is currently being played should be allowed to finish before canceling the active sequence and starting the new one. In almost all cases, this is precisely the behavior we want, but we also designed the system in such a way that a line can be interrupted in the middle of playback if it becomes necessary.

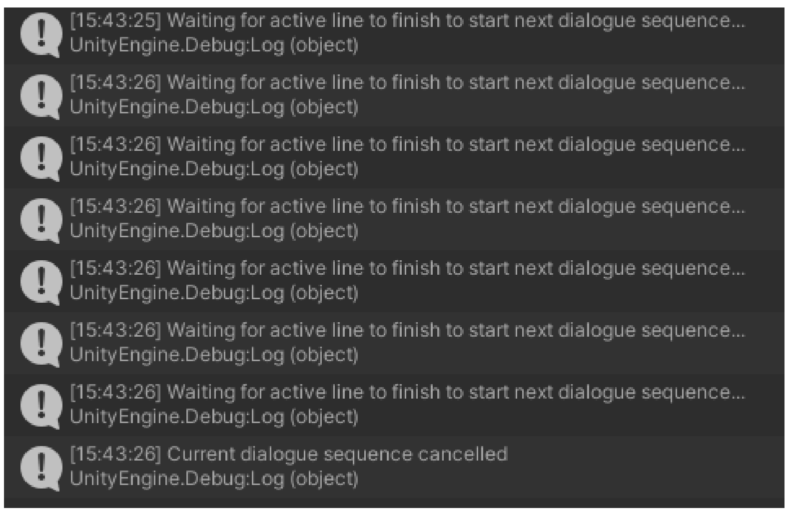


Figure 68. Scheduling a new dialogue sequence. Source: Screen capture.

Ideally, we wouldn't run into these sorts of situations too often, but not every player likes to play at the same pace, and there will always be those who develop "speedrun tactics," if you will. For this reason, we decided it would be better to play it safe than sorry. The odds are, if we ran into this issue in our testing, then at least one user would during gameplay as well. An example of this system in action can be seen in Figure 68, where the debug output shows that when a new dialogue sequence is scheduled to start, the dialogue manager makes it wait for the active line to finish before canceling the current dialogue sequence and ultimately starting the new one.

The dialogue manager will check ten times per second to determine whether the current line is over or not. This degree of granularity achieves a nice balance of not bogging the thread down by occurring too often, while also occurring frequently enough to ensure it has a chance to stop the current dialogue sequence before it tries to play its next line.

Another problem we ran into with our initial implementation of the dialogue system, which was alluded to above, was that the internal timers we were using to keep track of when a dialogue line is finished would keep ticking even when the game window was out of focus due to being run within an asynchronous task. Meanwhile, Wwise would halt all audio playback, which is the desired outcome. This discrepancy caused the two systems to get out of sync and sometimes play adjacent dialogue lines simultaneously. In order to solve this, we made use of the Wwise API's callback system. This allowed us to register a function as a callback whenever a specific lifecycle event in Wwise occurred, like a sound source finishing playing. Referring back to Figure 67 above, the callback type and callback function are passed to the event post call.

Figure 69 below shows the callback function's implementation.

```
private void EventCallback(object cookie, AkCallbackType type, AkCallbackInfo info) {
    if (type == AkCallbackType.AK_EndOfEvent) {
        _lineActive = false;
        Debug.Log( message: "End of dialogue line");
    }
}
```

Figure 69. Wwise callback event handler implementation. Source: Screen capture.

```
while (_lineActive) {
    await Task.Delay(250, token);
}

if (subtitlesTextBox != null) {
    var textBox = subtitlesTextBox.GetComponent<TextMeshProUGUI>();

    if (textBox != null) {
        textBox.SetText( sourceText: "");
    }
}

if (line.PostDelay > 0) {
    await Task.Delay((int) (line.PostDelay * 1000), token);
}

return line.NextLine;
```

Figure 70. Dialogue system waiting for current line to finish. Source: Screen capture.

Now, we can break out of the spinlock within the function responsible for playing a dialogue line, clear the subtitle text, wait for the requisite post delay, and continue on to the next line. Figure 70 shows the code for this. Note how the cancellation token is propagated down to all child tasks to ensure that the dialogue sequence can be properly alerted that it needs to be canceled no matter what point it's at in its execution.

```
while (true) {
    if (nextLine == "END") {
        break;
    }

    try {
        _sequenceActive = true;
        nextLine = await PlayLine(nextLine, _token);
        await Task.Delay(100, _token);
    } catch (OperationCanceledException) {
        Debug.Log( message: "Current dialogue sequence cancelled");
        break;
    } finally {
        ResetCancellationToken();
    }
}

_sequenceActive = false;
```

Figure 71. Dialogue sequence main loop. Source: Screen capture.

Finally, the main loop will continue playing the next line in the sequence until it is either interrupted by the cancellation token or reaches the “END” token as defined by the dialogue spreadsheet. An outline of this main loop is shown in Figure 71.

With all this in place, we were able to forgo our old implementation of waiting for a predetermined time and hoping no external factors caused our dialogue system and Wwise to become out of sync. This new implementation is much cleaner, as it eliminates the need for what was effectively a redundant property. The durations listed in the dialogue spreadsheet were simply the actual durations of the audio files rounded to the next highest quarter of a second for the sake of simplicity, so the accuracy of subtitle timing is increased as well by just letting Wwise handle this for us through its callback system.

```
private void Update() {
    if (!isPuzzleActive) return;
```

Figure 72. Geo puzzle guard clause. Source: Screen capture.

5.6. Geo frequency puzzle

This section picks up where *Section 2.2.3* left off in explaining the mechanics of the Geo frequency puzzle, with the aim of providing some more technical details. The logic for the puzzle section was largely handled by a single script that follows the singleton design pattern. Every frame, the script will first check to see whether the puzzle is active or not, so that it isn't wasting any processing power doing further unnecessary checks or calculations. This acts as a guard clause, forcing the function to return if the condition is not met (Figure 72). The puzzle is considered active if the player is inside the puzzle room and it has not already been solved.

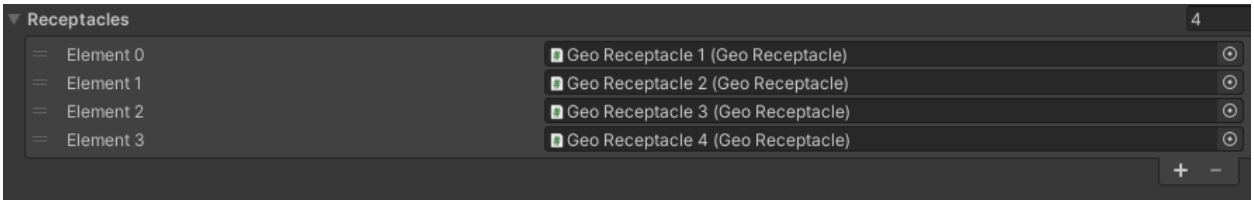


Figure 73. Geo puzzle receptacles list. Source: Screen capture.

Assuming the above condition is met, the necessary tones are played via Wwise if enough time has elapsed, otherwise it will continue to wait. The timeout for this note is set such that the generator tones are played as half notes in time with the limited, percussion-based background music, which has a tempo of 112 BPM. Through a bit of math, we arrive at a timeout of 1.0708 seconds, which makes sense intuitively given that a slightly faster tempo of 120 BPM would equate to exactly one half note every second. An internal member variable is used to keep track of which Geo receptacle is up next in the sequence, and its *GameObject* is passed along to Wwise to properly spatialize the audio. These receptacles are all part of a serialized array list exposed to Unity's Inspector (Figure 73).

```
if (Time.time > _nextNote) {
    _nextNote = Time.time + NoteTimeout;

    var receptacle = receptacles[_noteSequenceIndex];
```

Figure 74. Geo puzzle note timeouts. Source: Screen capture.

```
AkSoundEngine.SetState( in_pszStateGroup: "Generator_Hum_Note",  in_pszState: "Note_" + receptacle.targetEnergy);
AkSoundEngine.PostEvent("Play_Generator_Hum", receptacle.gameObject);

if (!receptacle.IsEmpty()) {
    AkSoundEngine.SetState( in_pszStateGroup: "Resonant_Frequency_Note",  in_pszState: "Note_" + receptacle.totalEnergy)
    AkSoundEngine.PostEvent("Play_Resonant_Frequency", receptacle.gameObject);
}
```

Figure 75. Geo puzzle Wwise calls. Source: Screen capture.

The target and total energy values of the current receptacle are used in order to form the string representing the state name given to Wwise when updating the state groups for the two different tones. The code for all of this behavior is shown in Figures 74 and 75. For more information on Wwise Audio Engine and its use throughout the project, see Section 5.2.

In order to handle the interactions between the player, the Geo receptacles, and the Geo core pickups, we first check to see whether the player is within range of a receptacle. If so, the first priority should be to try to interact with that. If the player is already holding a Geo core, it will be placed within the nearest receptacle. Likewise, if the player is not already holding a Geo core, but the nearest receptacle contains one or more, the stack representing the list of cores in the receptacle is popped and the player picks it up. If the player is not holding a Geo pickup already, is not within range of a receptacle, and there is a Geo pickup on the floor within a minimum distance, the player will pick it up. Otherwise, when the player is holding a Geo core and is not within range of a receptacle, they will simply drop it on the floor. If none of these conditions are met when the player presses the “interact” key, nothing will happen.

```
Array.Clear(_collidersBuffer, index: 0, _collidersBuffer.Length);
Physics.OverlapSphereNonAlloc(player.transform.position, radius: 5, _collidersBuffer);

GeoPickup nearestPickup = null;
var minPickupDist = double.MaxValue;

foreach (var coll:Collider in _collidersBuffer) {
    if (coll != null && coll.CompareTag("GeoPickup")) {
        var dist:float = Vector3.Distance(a:player.transform.position, b:coll.transform.position);

        if (dist < minPickupDist) {
            nearestPickup = coll.gameObject.GetComponent<GeoPickup>();
            minPickupDist = dist;
        }
    }
}

_currentPickup = nearestPickup;
```

Figure 76. Using an overlap sphere to find nearby Geo cores. Source: Screen capture.

From the above description, it's clear that deciding what the player should do when the "interact" key is pressed depends heavily on the positions of various components of the puzzle. Keeping track of the positions of the receptacles is fairly easy, as they are stationary and there is a limited number of them, so their distances from the player can be calculated with some basic vector math. Conversely, there may be any number of Geo cores on the floor at a given time and their positions are completely unpredictable. Because of this, an overlap sphere is used to find all objects with the "GeoPickup" tag within a radius of 5 in-game units, as seen in Figure 76. To save on performance, the version of this function that allocates no additional memory is used and the re-usable buffer is passed in each time it's called.

When the player places or drops a Geo core, its parent and position are updated accordingly, its rotation is reset, and its gravity and collider are enabled if it is not being placed in a receptacle. If it is, it's given an angular velocity of (0, 1, 0) so that it rotates clockwise around the y-axis slowly within the receptacle. When a Geo core is picked up by the player, its parent, position, and rotation are modified in the same way, its velocity is set to zero to prevent gaining speed by constantly picking up and dropping it, its gravity and collider are disabled, and it is a given a randomized angular velocity in the range of a unit vector. These behaviors (placing and picking up Geo cores) are shown in Figures 77 and 78 respectively.

```
if (isPickupInHand) {
    pickupTransform = _currentPickup.transform;

    if (isReceptacleInRange) {
        _nearestReceptacle.AddPickup(_currentPickup);
        var targetPos :Vector3 = _nearestReceptacle.GetNextOpenPosition();
        pickupTransform.position = targetPos;
        pickupTransform.rotation = Quaternion.identity;
        _currentPickup.GetComponent<Rigidbody>().angularVelocity = new Vector3( x: 0,  y: 1,  z: 0);
    } else {
        _currentPickup.GetComponent<Rigidbody>().useGravity = true;
        _currentPickup.GetComponent<BoxCollider>().enabled = true;
    }

    pickupTransform.SetParent( p: null);
    _currentPickup = null;
    isPickupInHand = false;
```

Figure 77. Logic flow of placing/dropping a Geo core. Source: Screen capture.

```
if (_currentPickup != null) {
    pickupTransform = _currentPickup.transform;
    pickupTransform.position = pickupDestination.position;
    pickupTransform.SetParent(pickupDestination);

    _currentPickup.GetComponent<Rigidbody>().useGravity = false;
    _currentPickup.GetComponent<BoxCollider>().enabled = false;

    var body = _currentPickup.GetComponent<Rigidbody>();
    body.velocity = Vector3.zero;
    body.angularVelocity = new Vector3( x: Random.Range(-1.0f, 1.0f),
        y: Random.Range(-1.0f, 1.0f),  z: Random.Range(-1.0f, 1.0f));

    isPickupInHand = true;
}
```

Figure 78. Logic flow of picking up a Geo core. Source: Screen capture.

5.7. Visual systems

5.7.1. Procedural creature animation

We implemented a procedural tool to automate the animation of several different creatures. We call it the crawler animator.

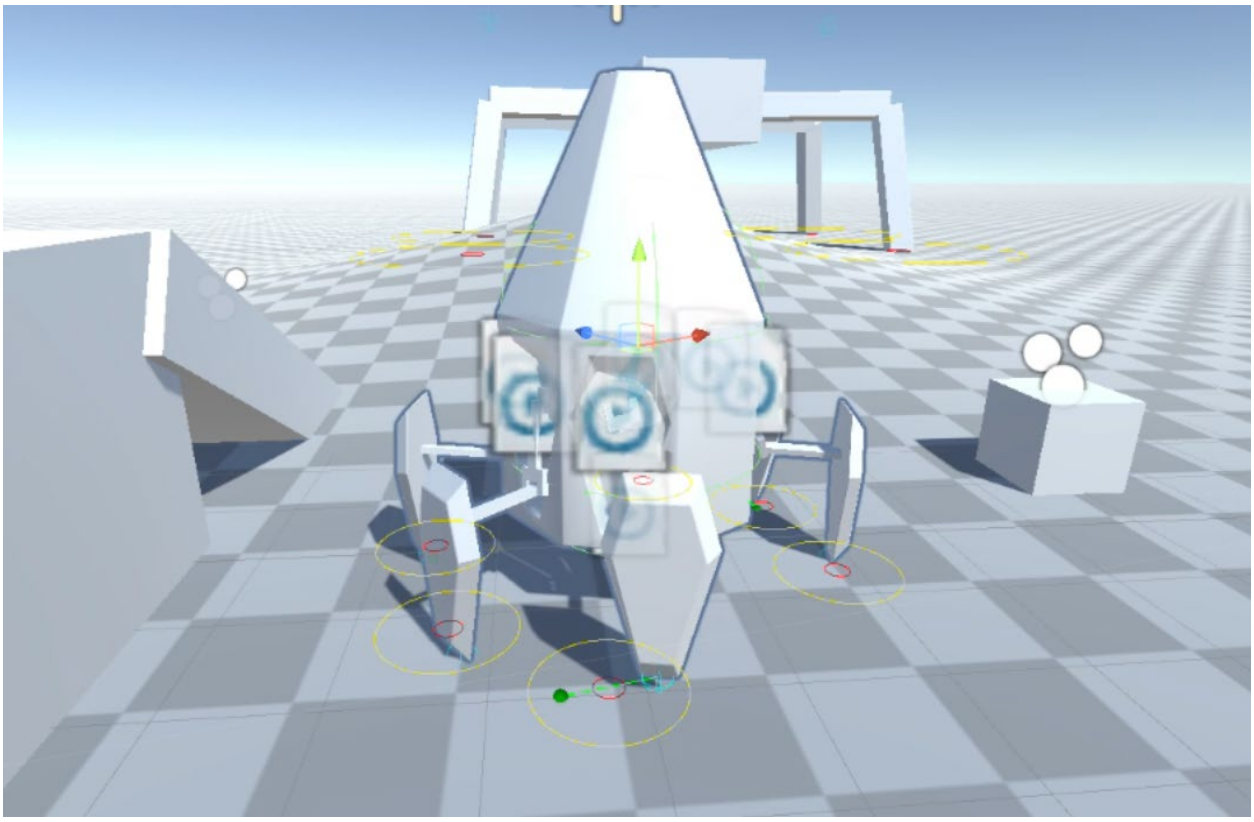


Figure 79. Automated leg targeting for animating creatures. Source: Screen capture.

The crawler animator worked by animating a creature's legs in response to the creature's movement. Each leg in this system was animated using Unity's built-in inverse kinematics package. They could then be articulated entirely by moving the target vector of their feet. Figure 79 shows a foot animating to reach a target vector (displayed in green). The target vector for each foot on the creature would be governed by the state machine in Figure 80 below.

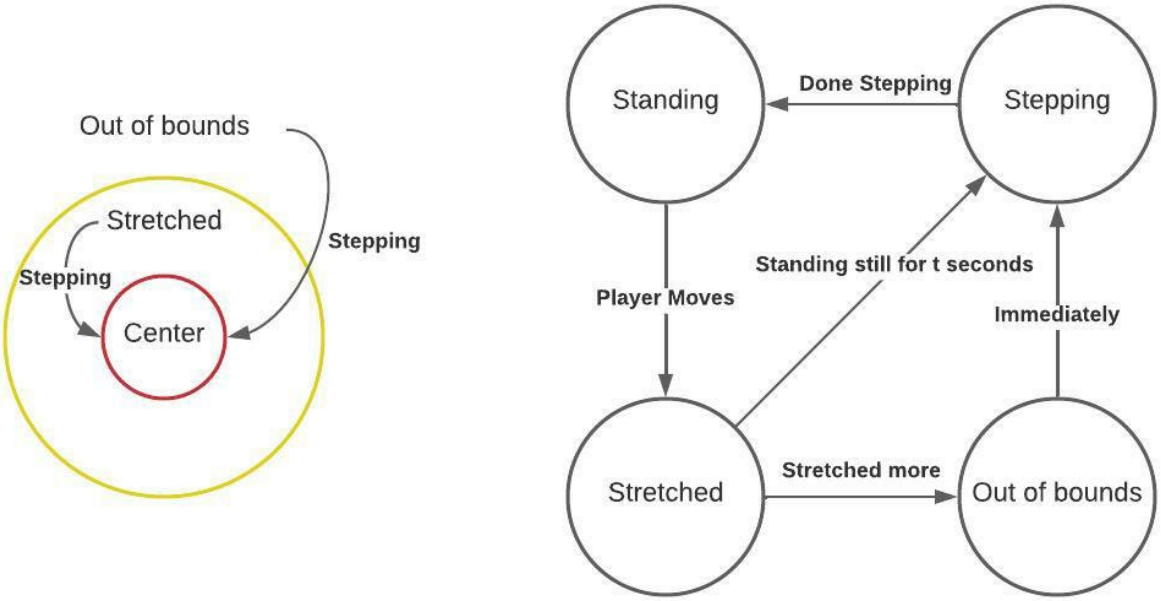


Figure 80. State machine for generalized crawling animator. Source: Authors.

The crawler animator contains an inner radius and an outer radius. These radii are centered around the point the foot "should be" relative to the center of the creature. If the foot of a leg is outside the inner radius, it is considered "stretched." If a foot is outside the outer radius it is considered "out of bounds." When in both these states, the leg will want to return to the "standing" state in which it is centered. The leg will enter the "stepping state" when it is attempting to re-center itself.

When a leg is stretched, it will wait for a set amount of time before re-centering itself. When a leg is out of bounds, it will re-center itself immediately. This results in a behavior that appears like walking if the creature is in motion, and appears like fidgeting when the creature is stationary.

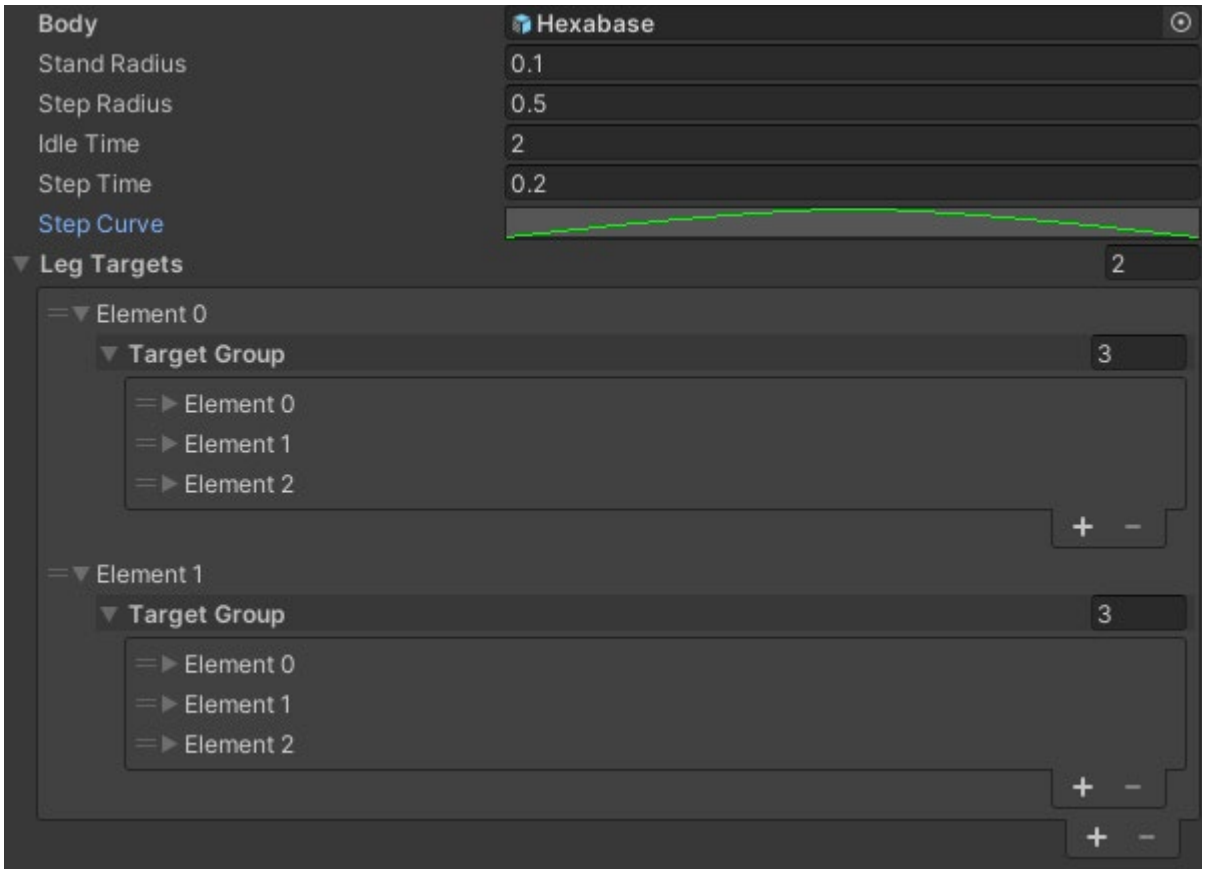


Figure 81. Serialized UI for grouping legs for walking animation. Source: Screen capture.

When in the "stepping" state, each foot's movement was governed by the "Step Time" and the "Step Curve" shown in Figure 81. The foot moves along the X and Z axes linearly to its target position for, in this case, 0.2 seconds. The y position of the leg during this process can be customized using the "Step curve." The height from the ground measured using a raycast is determined by the y value of the step curve at any point during the progress of the step.

The final part of the crawler animator is the list of leg targets. The list is a 2d array divided into groups. Only one group at a time can transition into the "stepping" state. This means that if any leg in any other group is currently stepping, then a leg cannot begin a step. This ensures that at no point during the animation are all legs off the ground. This lends itself better to creatures with more legs as other types of creatures may lift all their legs off the ground for some animations. This system is part of what drove the bug-like aesthetic for the design of some Geomorphs.

5.7.2. Rendering pipeline

Unity has three different rendering pipelines available as native packages: the built-in rendering pipeline, the URP (Universal Rendering Pipeline), and the HDRP (High Definition Rendering Pipeline). The built-in rendering pipeline is Unity's legacy renderer. The team knew we would end up using either the URP or HDRP, however the built-in renderer was used for the bulk of development as gameplay functionality was prioritized over graphical fidelity. After polish work began on the game's graphics, we transitioned all in-game materials to the URP.

Built-in Render Pipeline	URP (Universal Render Pipeline)	HDRP (High Definition Rendering Pipeline)
+ Older and more stable + Compatible with other legacy systems + Lower system requirements - Contains many visual artifacts	+ Lower system requirements + Support for shader graph + Unbounded light source count - Limited to 8 light sources per object	+ High quality out-of-the-box graphical fidelity + Support for shader graph + Unbounded light source count - Higher system requirements - Implemented unneeded effects

Figure 82. List of pros and cons for each of Unity's rendering pipelines. Source: Authors.

Figure 82 lists the pros and cons we identified for each of the available rendering options. Ultimately, we decided to use the URP. Realistically, we would not end up using the features that the HDRP provided within the scope of our game. The limit of eight light sources did present an obstacle for designing the visuals of the game; however, there was a workaround. Technically the limit was light sources per Game Object, so splitting the lights into multiple objects essentially meant that our number of renderable lights was uncapped.

6. Evaluation

6.1. AlphaFest

AlphaFest is an event held by the IMGD department at WPI in which students can present and have playtested any games they are actively developing. All IMGD MQPs are required to present their games at AlphaFest, so this served as our first round of testing, even though we weren't entirely satisfied with what we'd be showing off yet. Our testing methodology (Appendix A) involved briefing participants on our informed consent agreement (Appendix B) and COVID-19 risk mitigation strategy (Appendix C) before having them play the game and fill out a survey afterwards (Appendix D.) The full results of this playtesting session can be found in Appendix E.

From our testing during AlphaFest, the most important feedback we received was that the character was too difficult to control and the game's combat felt unresponsive and was not engaging enough. At this point, our combat systems were still in their infancy, with several broken animations, shaky hit detection, and various bugs with the enemy AI all still present in the build. Because of this we expected a lot of the feedback that we received.

Half of the responses we received said that the character was either "somewhat difficult" or "difficult" to control. 75% of playtesters also reported that the game's combat was unengaging to some degree. These metrics (shown in Figures 83 and 84 below, respectively) were quite far from where we wanted them to be, so improving the player controller and the combat experience became two of our big areas of focus.

How easy did you feel the character in this game was to control?

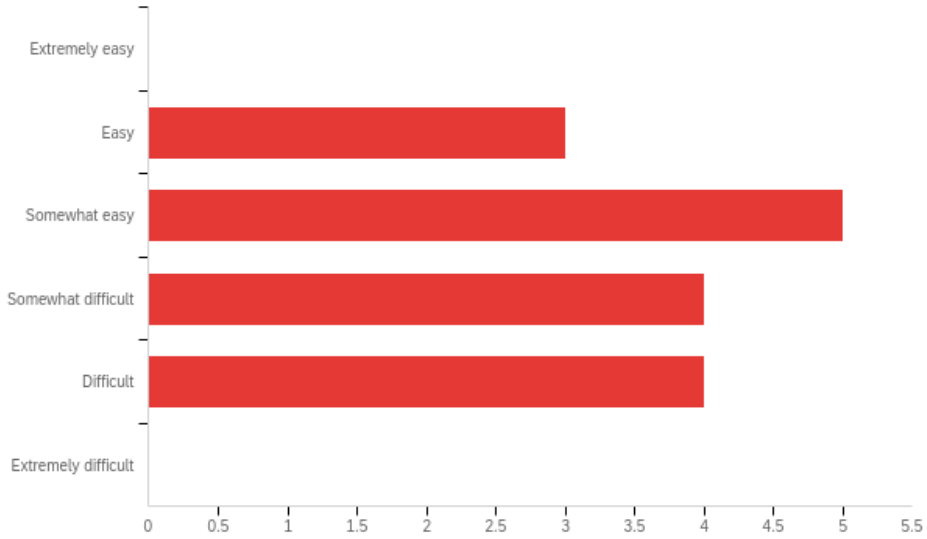


Figure 83. Responses to game control question.

How engaging did you find the game's combat?

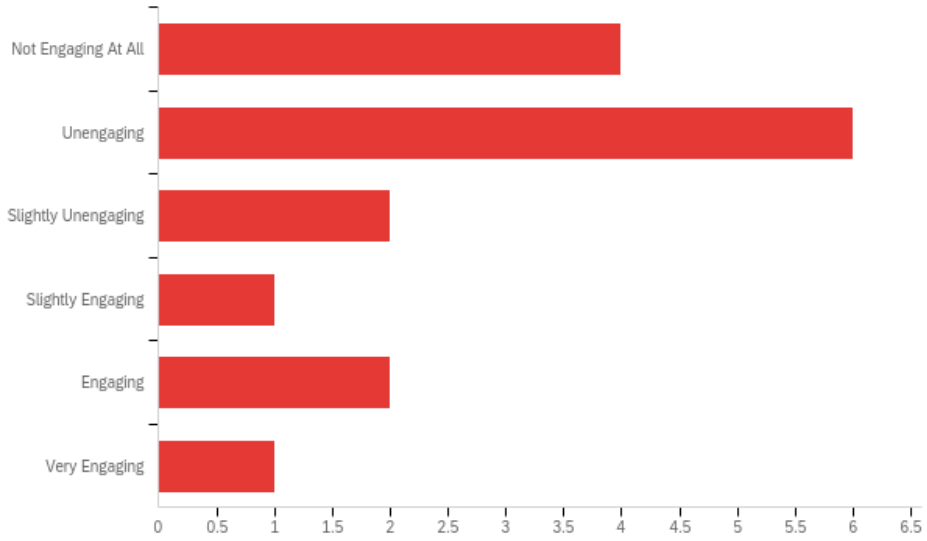


Figure 84. Responses to game combat question.

In the survey, we also asked playtesters how much they thought that the sound effects and music contributed to their enjoyment of the game. 100% of respondents said that the sound effects contributed to their enjoyment to some degree, providing comments such as, “The sound effects made swinging at fake enemies fun-ish.” 93.75% of respondents said that the music contributed to their enjoyment to some degree, providing comments such as, “I really enjoyed the background music,” “[It] adds to the atmosphere,” and, “[The] music was good.” We were happy to see these results, as the audio of the game was one of our big focuses going into its development.

6.2. IMGD 2900

Our second round of playtesting was done under the same methodology in Prof. Moriarty’s IMGD 2900 class–Digital Game Design I. The full results of this playtesting session can be found in Appendix F.

Our new build for this playtest featured some improvements to the camera control, player attack animations, and collision detection influenced by the feedback from our AlphaFest testing. This time around, we received some feedback like, “The character was very responsive and there was very little delay even in ridiculous actions like spinning around quickly in the air,” and “Controlling the character was intuitive based on other PC games.”

There were still some issues that needed to be resolved though, particularly regarding the use of the *GeoBlade* in combat. We received comments such as “The sword swings felt sluggish, and it would swing twice even though I only clicked once,” and, “I feel like the attack animation is too slow.”

The new build also included a much larger playable space, greatly improved visuals, voice-acted dialogue, and the Geo frequency puzzle. With the addition of the puzzle, we revised our survey to ask some specific questions about it. We found that exactly half of the participants were able to solve the puzzle, 65% found it engaging to some degree, and 70% found it difficult to some degree. The breakdown of these responses is shown in Figures 85-87 below.

Were you able to solve the puzzle and reach the demo-end screen?

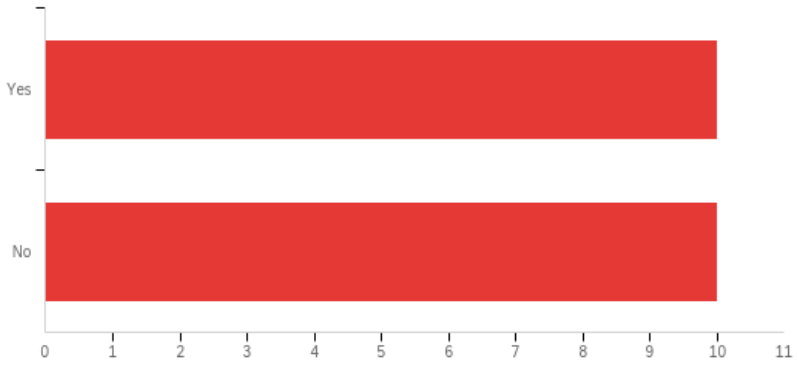


Figure 85. Puzzle question 1 data.

How engaging did you find the puzzle?

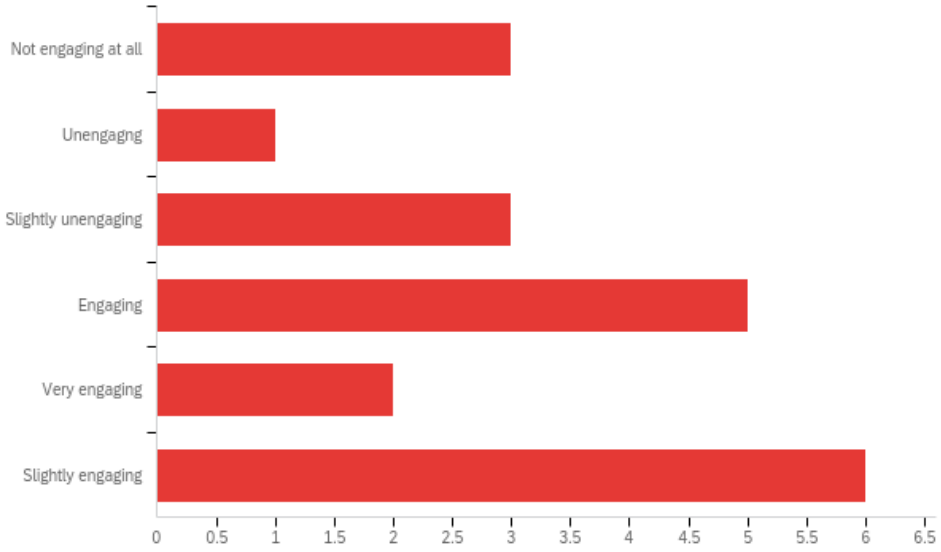


Figure 86. Puzzle question 2 data.

How difficult did you find the puzzle?

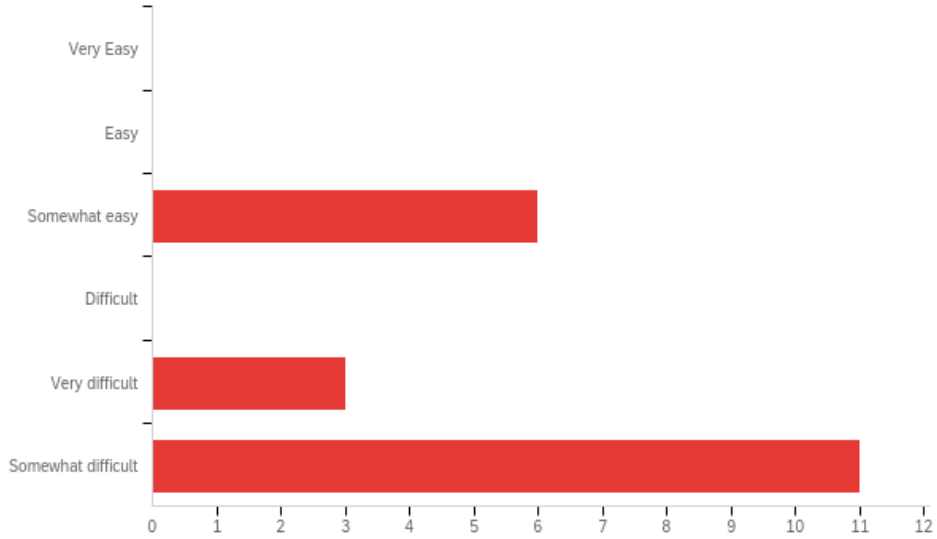


Figure 87. Puzzle question 3 data.

These metrics are decent, but some of the comments we received on the puzzle revealed that there was not enough visual information to properly convey the goal and mechanics of the puzzle. We received some comments such as, “It was a lot of trial and error even with a musical ear but it was very engaging so the monotony of constantly replacing geo blocks wasn't that bad,” “As someone who is essentially tone deaf, there were few ways I could have finished the puzzle without some hints,” and “It took me a bit to figure out that you have to make the pitches be exactly one octave above the pitch that the cores were already playing. Maybe that should be communicated more clearly.”

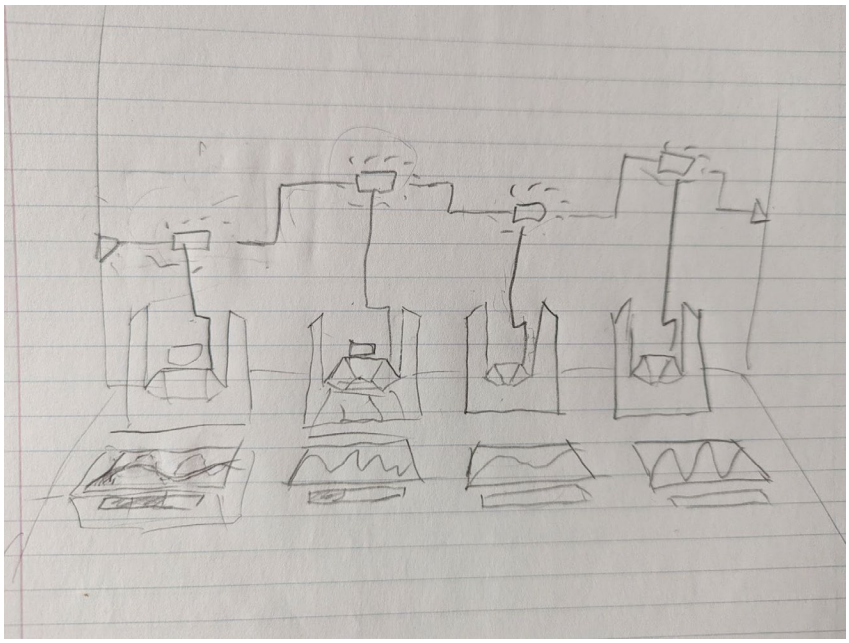


Figure 88. Geo frequency visual redesign mockup. Source: Original concept art.

With this feedback in mind, we set out to re-design the visuals of the puzzle. We unfortunately did not have enough time to fully implement all of these ideas in the final build of the game, but Figure 88 shows a mockup of what we had in mind. We wanted to add an indicator of how much energy was in a receptacle, display the frequencies of the two tones being played as graphs on the same axes, and include a circuit on the wall that the receptacles would connect to as they are solved in order to indicate overall progress.

Engagement with the sound effects and music remained largely the same, with some additional interest brought on by the addition of voice acted dialogue and the audio-based puzzle mechanics. Some of the feedback we received along these lines included, “I liked Seru's voice a lot,” “[The sound effects and dialogue] were my favorite part,” “The dialogue setting up the mysterious story of the game contributed to my enjoyment,” and, “The music was pretty good.”

We also received recurring feedback that the game's lighting was too dark and that it desperately needed a checkpoint system, brought on by the drastically extended play space compared to the build from AlphaFest. The lighting issue was made clear from comments such as, "Lighting in the beginning areas could help," and, "I couldn't see where I was going half the time." As for the checkpoint problem, we received comments like, "It was frustrating starting over from the beginning when I would die," and, "I accidentally fell during some of the parkour and was a little annoyed that I had to completely start over." As such, these were two of the main things we prioritized with what little time we had left before our final presentation.

Conclusion

Ultimately, we ended up with almost the entire first level of the game completed in time to present the project. Reflecting on our original plans, we had set our expectations far too high in hoping to complete all three levels. As the development of the game went on, it became more and more clear to us that we had vastly over-scoped. We had originally intended for the game's combat to be more reminiscent of a hack-and-slash style game, but as we continued to be forced to cut abilities and enemies, the game lost some of that identity. Because of this, the core gameplay loop feels a tad uninspired.

We think one of the major flaws of our development process was in where we focused our attention. We would often end up getting stuck going down a rabbit hole of writing and rewriting systems meant to support things which we didn't even know if we would end up needing. Additionally, we put a bit too much focus on the game's narrative, dedicating a lot of time to refining the script and recording hundreds of lines of dialogue that unfortunately ended up unused. Another issue we faced was proper management of both time and tasks. As alluded to above, we did not always prioritize the right tasks and would often find ourselves in a situation where we were waiting for another project member to complete a blocking task before we could proceed. This led to inconsistent bursts of productivity as well as some periods of crunch, particularly right before our two testing sessions, where we would sometimes complete the same amount of work in a few days as we had over multiple weeks during another part of the development cycle.

With all this said, we are still quite happy with what we were able to accomplish in the time we had. While not quite to the scale we had hoped it would be, we're quite pleased with the audio side of things in particular. One of our goals for this project was to learn more about industry standard development tools and techniques, and we certainly know a lot more about Unity, Wwise, Houdini, and the other various tools we made use of along the way than we did when we began development. Even the finished project was by far the largest game any of us had ever attempted to make, and our original plan was for it to be over triple the size. The experience of developing a game of this scale, and integrating player feedback into that process has been extremely valuable to all of us, despite not quite accomplishing all the goals we set for ourselves initially.

Works cited

Andy. “Answer to ‘Untrack Files from Git Temporarily.’” *Stack Overflow*, August 6, 2011. https://stackoverflow.com/a/6964322/13450925.

“Animation/Animator Problems.” *Unity Forum*. Accessed April 27, 2022. https://forum.unity.com/threads/animation-animator-problems.337987/.

Audition (version 22.1.1). Adobe, n.d. https://www.adobe.com/products/audition.html.

Carigé, Washington. “Answer to ‘Is It Possible to Apply Geometry Nodes?’” *Blender Stack Exchange*, December 16, 2021. https://blender.stackexchange.com/a/247256.

Destiny 2. Destiny. Bungie Inc, 2017. https://www.bungie.net/7/en/Destiny/NewLight.

“Documentation | Audiokinetic.” Accessed April 27, 2022. https://www.audiokinetic.com/library/edge/?source=Unity&id=index.html.

Dokutaa Sutoon. Animation, Action, Adventure. TMS Entertainment, 2019.

Fate/Stay Night: Unlimited Blade Works. Animation, Action, Adventure. Aniplex, Notes, Type-Moon, 2014.

FL Studio. Accessed April 27, 2022. https://www.image-line.com/fl-studio/.

“Freesound.” Accessed April 27, 2022. https://freesound.org/.

Fusion 360. Windows. Autodesk Inc. Accessed April 27, 2022. https://www.autodesk.com/products/fusion-360/.

Game of Thrones. Action, Adventure, Drama. Home Box Office (HBO), Television 360, Grok! Studio, 2011.

Genshin Impact. miHoYo, 2020. https://genshin.hoyoverse.com/en/.

"Glow with Bloom & Emissive Materials Which Are Invisible." *Unity Forum*. Accessed April 27, 2022. https://forum.unity.com/threads/glow-with-bloom-emissive-materials-which-are-invisible.1100920/.

Haas, J. *Unity* (version 2020.3.17f1). Windows. Unity Technologies, n.d. https://unity.com/.

Halo: Combat Evolved. Halo. Bungie Inc, 2001. https://www.bungie.net/.

"[HELP] How to Use Fog in Universal RP?" *Unity Forum*. Accessed April 27, 2022. https://forum.unity.com/threads/help-how-to-use-fog-in-universal-rp.843361/.

Houdini FX (version 19.0.455). Windows. Side Effects Software Inc., n.d. https://www.sidefx.com/.

Humble Bundle. "Humble Software Bundle: Unity Fantasy Games & Game Dev Assets." Accessed April 27, 2022. https://www.humblebundle.com/software/unity-fantasy-games-dev-assets-software.

Humble Bundle. "Humble Software Bundle: UNITY FPS Games and Game Dev Assets." Accessed April 27, 2022. https://www.humblebundle.com/software/unity-fps-games-and-game-dev-assets-software.

Jujutsu Kaisen. Animation, Action, Adventure. Mappa, TOHO animation, 2020.

Kojima, Masayuki. *Made in Abyss: Fukaki Tamashî No Reimei*. Animation, Adventure, Drama. Kinema Citrus, 2020.

Kosinski, Joseph. *Tron: Legacy*. Action, Adventure, Sci-Fi. Walt Disney Pictures, Sean Bailey Productions, LivePlanet, 2010.

"Learn Wwise | Audiokinetic." Accessed April 27, 2022. https://www.audiokinetic.com/courses/wwise201/.

Lucas, George. *Star Wars: A New Hope*. Action, Adventure, Fantasy. Lucasfilm, Twentieth Century Fox, 1977.

Made in Abyss. Animation, Adventure, Drama. Kinema Citrus, 2017.

"Mecanim Default Values." *Unity Forum*. Accessed April 27, 2022. https://forum.unity.com/threads/mecanim-default-values.266613/.

Mixamo. Adobe Inc. Accessed April 27, 2022. https://www.mixamo.com/#/.

Miyamoto, Shigeru. *The Legend of Zelda Series*. The Legend of Zelda. Nintendo, 1986. https://www.zelda.com/.

Ohman, Wiktor, James Lucas, and Christopher Widdowson. *Quixel Mixer* (version 2021.1.3 Beta). Windows. Quixel, 2020.

Patala, Arsi. *Ultrakill*. Microsoft Windows. New Blood Interactive, 2020. https://newblood.games/.

Photoshop (version 23.2.1.303). Adobe, n.d. https://www.adobe.com/products/photoshop.html.

Portal (Series). Portal. Valve Corporation, 2007. https://store.steampowered.com/app/400/Portal/.

Quixel Megascans (version 2022.0.2). Quixel, n.d. https://quixel.com/about.

REAPER (version 6.43). Cockos. Accessed April 27, 2022. https://www.reaper.fm/.

Roosendaal, Ton. *Blender* (version 2.93). Windows. Blender Foundation. Accessed April 27, 2022. https://www.blender.org/about/.

Roro, Laurent. "Answer to 'Unwanted Flickering in Rendered Animation.'" *Blender Stack Exchange*, October 29, 2020. https://blender.stackexchange.com/a/200320.

Shellam, Andy. "Answer to 'Switch Statement in C#.'" *Stack Overflow*, March 31, 2010. https://stackoverflow.com/a/2552532/13450925.

Shingeki No Kyojin. Animation, Action, Adventure. Wit Studio, Production I.G., Mappa, 2013.

"[SOLVED] Scale of Model Changes during Animation?" *Unity Forum*. Accessed April 27, 2022. https://forum.unity.com/threads/solved-scale-of-model-changes-during-animation.356070/.

Taura, Takahisa, and Isao Negishi. *NieR:Automata*. Drakengard. PlatinumGames, 2017. https://nierautomata.square-enix-games.com/en-us/home/.

Unity Documentation. "Unity Documentation." Accessed April 27, 2022. https://docs.unity.com/.

Westworld. Drama, Mystery, Sci-Fi. Bad Robot, Jerry Weintraub Productions, Kilter Films, 2016.

Wwise (version 2021.1.3.7665). Audiokinetic, n.d. https://www.audiokinetic.com/en/products/wwise.

Appendices

A. IRB Study Protocol/Research Methods

Title of Research Study: Audio-Based Adventure Game

Purpose of study

To obtain user feedback in order to determine if experience goals are being achieved, locate operational bugs, and identify opportunities for design improvement.

Project description

A third-person 3D action adventure, with an emphasis on storytelling through interactive audio and music, in which players uncover the mysteries of an ancient civilization. The player assumes the role of a scout drone tasked with exploring a long-abandoned planet to determine if it is suitable for recolonization. Along the way, players encounter enemies and allies as they learn the sinister truth behind this alien world.

Study protocol

Participants are directed to a test location where they are given the Opening Briefing (below) and asked to complete the Informed Consent and COVID-19 Mitigation Strategy Agreements.

After signing the Agreements and completing the game, participants are asked to fill out a short online survey to characterize aspects of their subjective experience and solicit suggestions for improving the experience.

Opening briefing for testers

“Hello, and thank you for volunteering to test our project. Before we begin, could you please read and sign these Informed Consent and COVID-19 Mitigation forms? [Tester signs forms.] Thank you. When your session is complete, we will ask you to complete a brief survey about your play experience. At no point during your test session, or in the survey after, will any sort of personal and/or identifying information about you be recorded. Please begin playing when you feel ready.”

B. IRB Informed Consent Agreement

Informed Consent Agreement for Participation in a Research Study

Investigator: Keith Zizza

Contact Information: kzizza@wpi.edu, (508) 831-4641

Title of Research Study: Audio-Based Adventure Game

Sponsor: WPI

Introduction: You are being asked to participate in a research study. Before you agree, however, you must be fully informed about the purpose of the study, the procedures to be followed, and any benefits, risks or discomfort that you may experience as a result of your participation. This form presents information about the study so that you may make a fully informed decision regarding your participation.

Purpose of the study: The purpose of this study is to obtain feedback on the project in order to facilitate design improvements and find/address operational bugs.

Procedures to be followed: You will be asked to play a brief game lasting less than ten minutes. Instrumentation in the game software will anonymously record your activity during play. After completing the game, you will be asked to complete a brief, anonymous survey describing aspects of your subjective experience.

Risks to study participants: There are no foreseeable risks associated with this research study.

Benefits to research participants and others: You will have an opportunity to enjoy and comment on a new game under active development. Your feedback will help improve the game experience for future players.

Record keeping and confidentiality: Records of your participation in this study will be held confidential so far as permitted by law. However, the study investigators and, under certain circumstances, the Worcester Polytechnic Institute Institutional Review Board (WPI IRB) will be able to inspect and have access to confidential data that identify you by name. Any publication or presentation of the data will not identify you.

Compensation or treatment in the event of injury: There is no foreseeable risk of injury associated with this research study. Nevertheless, you do not give up any of your legal rights by signing this statement.

For more information about this research or about the rights of research participants, or in case of research-related injury, contact the Investigator listed at the top of this form. You may also contact the IRB Manager (Ruth McKeogh, phone 508 831-6699, email irb@wpi.edu) and/or the Human Protection Administrator (Gabriel Johnson, phone 508-831-4989, email gjohnson@wpi.edu).

Your participation in this research is voluntary. Your refusal to participate will not result in any penalty to you or any loss of benefits to which you may otherwise be entitled. You may decide to stop participating in the research at any time without penalty or loss of other benefits. The project investigators retain the right to cancel or postpone the experimental procedures at any time they see fit.

By signing below, you acknowledge that you have been informed about and consent to be a participant in the study described above. Make sure that your questions are answered to your satisfaction before signing. You are entitled to retain a copy of this consent agreement.

_____ Date: _____

Study Participant Signature

_____

Study Participant Name (Please print)

_____ Date: _____

Signature of Person who explained this study

C. IRB COVID-19 Risk Mitigation Strategy

Investigator: Keith Zizza

Contact Information: kzizza@wpi.edu, (508) 831-4641

Title of Research Study: Audio-Based Adventure Game

Sponsor: WPI

At WPI, our primary responsibility related to research is to protect the safety of our research participants.

COVID-19 refers to the Coronavirus that is being spread across people in our communities. We need to provide you with important information about COVID-19, and to tell you about ways your study participation might change because of COVID-19 related risk.

If you are considering joining a study at this time or are currently enrolled in a study, **it is important that you consider the following information to determine if study participation is right for you at this time.**

How is COVID-19 spread?

COVID-19 is a respiratory virus spread by respiratory droplets, mainly from person-to-person. This can happen between people who are in close contact with one another (less than 6 feet). It is also possible that a person can get COVID-19 by touching a surface or object (such as a doorknob or counter surface) that has the virus on it, then touching their mouth, nose or eyes.

Can COVID-19 be prevented?

Current ways to minimize the risk of exposure to COVID-19 include "social distancing" which is a practice to decrease the potential for direct exposure to others who may have been exposed to COVID-19, for example by avoiding large gatherings or refraining from shaking hands with others. It is important to understand that since study participation may include increased travel outside of your home and increased exposure to others within a research site it may increase your exposure to COVID-19. At this time, there is no vaccination to prevent COVID-19 infection.

What are the risks of COVID-19?

For most people, the new coronavirus causes only mild or moderate symptoms, such as fever and cough. For some, especially older adults and people with existing health problems, it can cause more severe illness, including pneumonia. While we are still learning about this virus, the information we have right now suggests that about 3 of 100 people who are infected might die from the virus.

Who is most at risk?

Individuals over 60 and with chronic conditions such as cancer, diabetes and lung disease have the highest rates of severe disease from the infection.

What do we do to minimize risk for research participants?

a. All in-person research will take place on the WPI campus.

b. Participation in the study will be strictly limited to WPI students and faculty authorized to attend campus in-person.

c. **Research visits will strictly abide by all official WPI COVID-19 protocols in effect at the time of the test session.** These protocols specify campus-wide standards for COVID-19 mitigation, including (but not limited to):

- Visitors allowed on campus
- Required vaccination status of visitors
- Masking requirements
- Social distancing requirements
- Maximum room occupancy requirements

A summary of the latest WPI protocols is maintained at this URL:

https://www.wpi.edu/we-are-wpi

d. Regardless of current WPI protocols, all test administrators and subjects will be required to wear a face mask at all times during the test session.

e. Test subjects will visit the research site only once, and only long enough to review the Informed Consent Agreement, participate in the test and respond to the research survey.

f. The location where study subject visits take place will have hospital-approved hand sanitizer readily available for use before and/or after the test session.

g. All physical equipment handled by subjects during the test (keyboards, mice, game controllers, headsets, etc.) will be thoroughly sanitized with alcohol wipes before each test session.

If you have further questions about COVID-19 and your participation in research, please talk to your study team.

_____ Date: _____

Study Participant Signature

_____

Study Participant Name (Please print)

_____ Date: _____

Signature of person who explained this study

D. IRB Study Instrument

Study Instrument for Audio-Based Adventure Game

Proposed survey questions:

1. How would you rank the following aspects of gameplay in general from most to least important to your experience?

[Rank in ascending order]

- Visuals
- Music
- Sound effects
- Controls
- Novelty
- Narrative

2. How familiar are you with games of this genre (3D hack-and-slash)?

[Lickert scale 1-6] 1=Not familiar at all (never played a game like this before), 6=Very familiar (plays games like this all the time)

(Optional) Provide examples of other similar games you've played if applicable: [Short response]

3. How easy did you feel the character in this game was to control?

[Lickert scale 1-6] 1=Very difficult (character controls feel clunky and/or are difficult to understand), 6=Very easy (character controls were very intuitive and responsive)

(Optional) Comments: [Short response]

4. How engaging was the game's combat?

[Lickert scale 1-6] 1=Not engaging at all (fighting enemies is boring, detracts from the overall experience/story, or is otherwise detrimental), 6=Very engaging (fighting enemies mechanically enjoyable and complements other elements of the gameplay experience well)

(Optional) Comments: [Short response]

5. How interested were you in the world and lore of this game?

[Lickert scale 1-6] 1=Not interested at all (the narrative elements of the game were not compelling and/or hindered enjoyment of mechanical gameplay), 6=Very interested (the narrative elements of the game strengthen the engagement of the gameplay and make you want to learn more about what's going on)

(Optional) Comments: [Short response]

6. Would you agree with the following statement: "The sound effects contributed to my enjoyment of this game?"

[Lickert scale 1-6] 1=Strongly disagree (the sound effects were low quality, out-of-place, and/or distracting), 6=Strongly agree (the sound effects strengthened the experience and enhanced the story and gameplay mechanics)

(Optional) Comments: [Short response]

7. Would you agree with the following statement: "The music contributed to my enjoyment of this game?"

1=Strongly disagree (the music was low quality, out-of-place, and/or distracting), 6=Strongly agree (the music strengthened the experience and enhances the story and gameplay mechanics)

(Optional) Comments: [Short response]

8. (Optional) How would you describe this game to a friend?

[Short response]

9. (Optional) Do you have any other feedback or comments you would like to share?

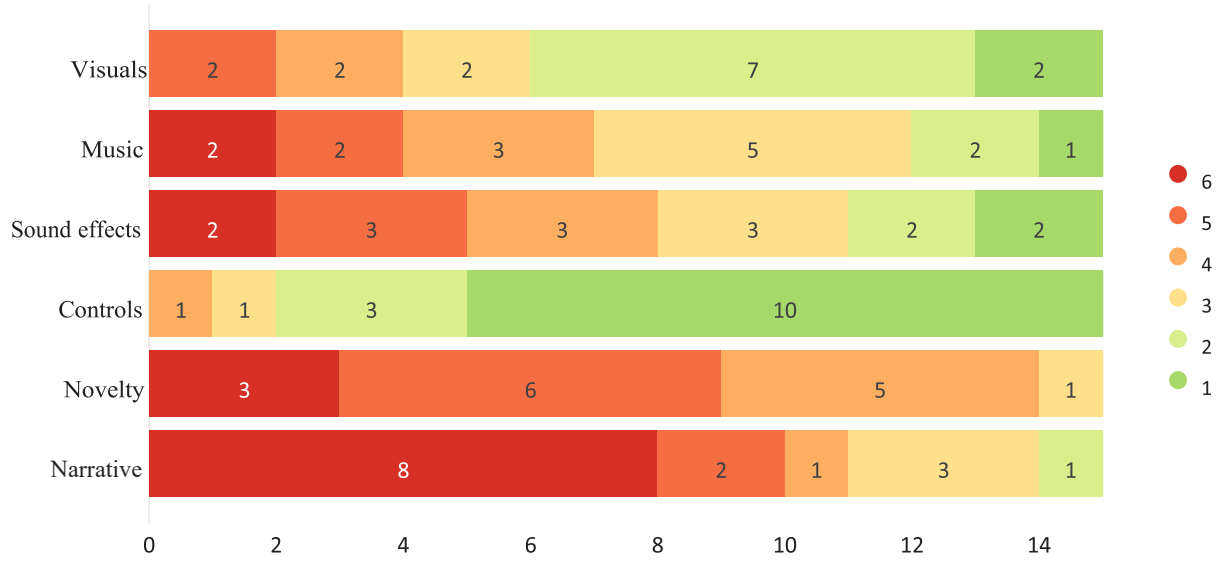
[Short response]

E. AlphaFest test data

Are you a WPI student/faculty member?

Field	Choice Count
Yes	16
No	0
Total	16

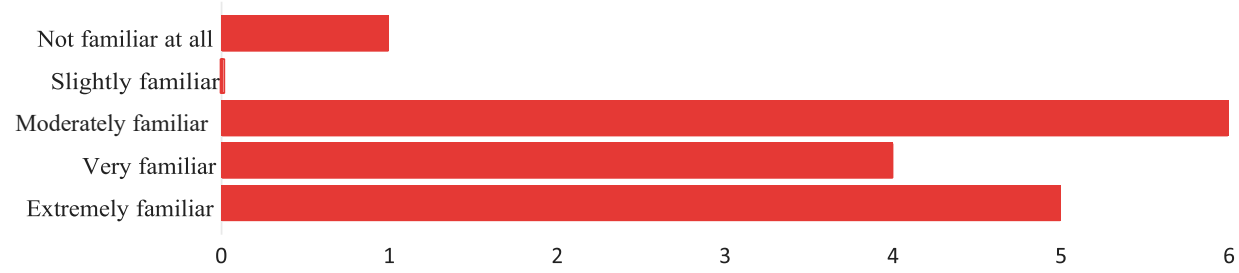
How would you rank the following aspects of gameplay in general from most to least important to your experience?



Field	Min Responses	Max	Mean	Standard Deviation	Variance	
Visuals	1	5	3	1	2	15
Music	1	6	4	1	2	15
Sound effects	1	6	4	2	3	15
Controls	1	4	2	1	1	15
Novelty	3	6	5	1	1	15
Narrative	2	6	5	1	2	15

Field	1	2	3	4	5	6	Total
Visuals	2	7	2	2	2	0	15
Music	1	2	5	3	2	2	15
Sound effects	2	2	3	3	3	2	15
Controls	10	3	1	1	0	0	15
Novelty	0	0	1	5	6	3	15

How familiar are you with games of this genre (3D hack-and-slash)?



Field	Choice Count
Not familiar at all	1
Slightly familiar	0
Moderately familiar	6
Very familiar	4
Extremely familiar	5
Total	16

(Optional) Provide examples of other similar games you've played if applicable

witcher 3, hades, bastion, dark souls

bayonetta, astral chain

Zelda, Metal Gear Rising

skyrim

Devil May Cry, Yakuza, Bayonetta, Transformers: Devastation

How easy did you feel the character in this game was to control?

Field	Choice Count
Extremely easy	0
Easy	3
Somewhat easy	5
Somewhat difficult	4
Difficult	4
Extremely difficult	0
Total	16

(Optional) Comments:

feels incomplete, no comment

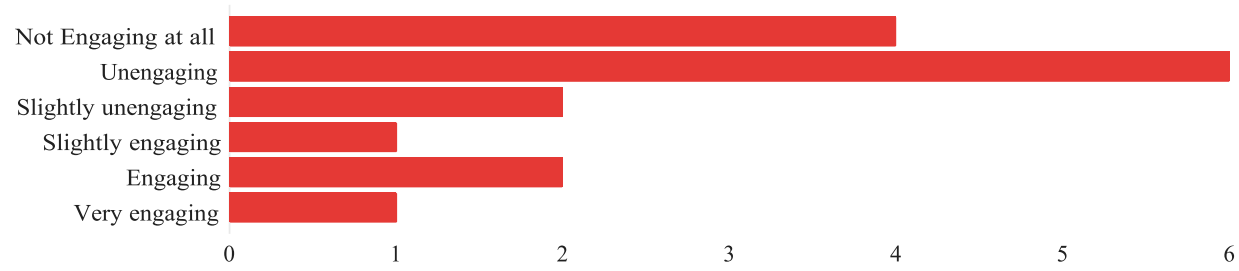
movement felt easy to control, though the camera often felt a bit difficult

The animations don't convey actions well. Additionally, the mouse sensitivity is far too low

a combination of all of the controls (wasd and mouse) were needed to get the character where I needed it to go, and it was hard to get used to it

The attack didn't really do anything, and the little character had some wacky camera controls

How engaging did you find the game's combat?

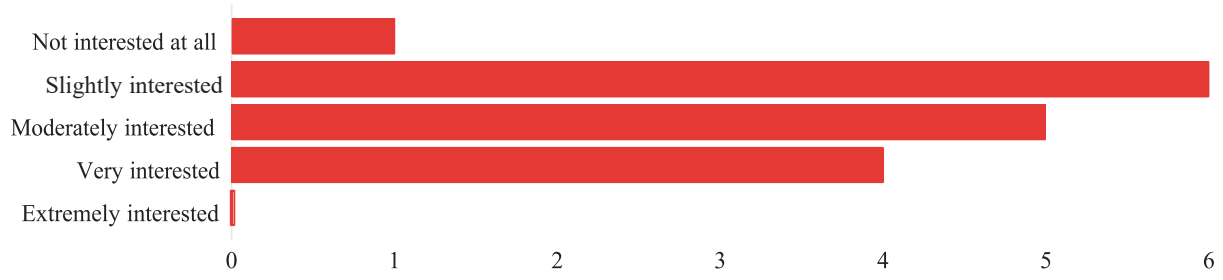


Field	Choice Count
Not engaging at all	4
Unengaging	6
Slightly unengaging	2
Slightly engaging	1
Engaging	2
Very engaging	1
Total	16

(Optional) Comments:

(Optional) Comments:
feels incomplete, no insight
I had no idea when I was hitting the enemy, could only know when I was being hit by constantly checking the health bar
The enemies don't do much visibly and are far too hard to beat.
Attack animation not played at the same time as your mouse click
i enjoyed hitting the little dudes

How interested were you in the world and lore of this game?



Field	Choice Count
Not interested at all	1
Slightly interested	6
Moderately interested	5
Very interested	4
Extremely interested	0
Total	16

(Optional) Comments:

I'm curious as to what the building I was in was and why the player and enemies were there

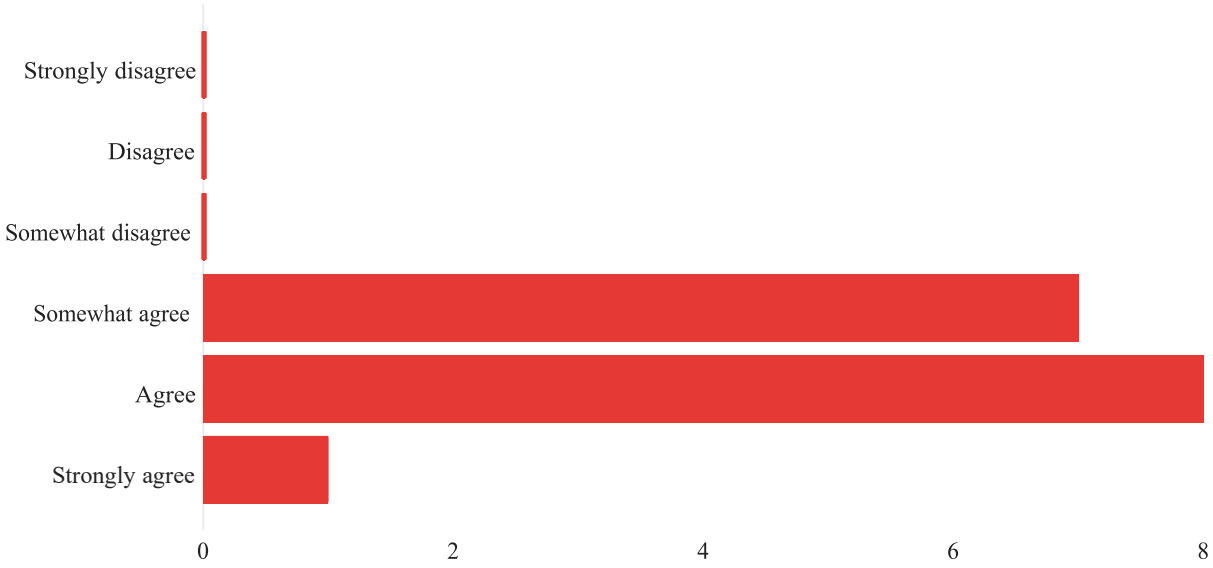
A cutaway might help focus the story

i am curious to know more about the origin of our character

what I was told about the game world seemed really fun

Lore?

Would you agree with the following statement: "The sound effects contributed to my enjoyment of this game?"



Field	Choice Count
Strongly disagree	0
Disagree	0
Somewhat disagree	0
Somewhat agree	7
Agree	8
Strongly agree	1
Total	16

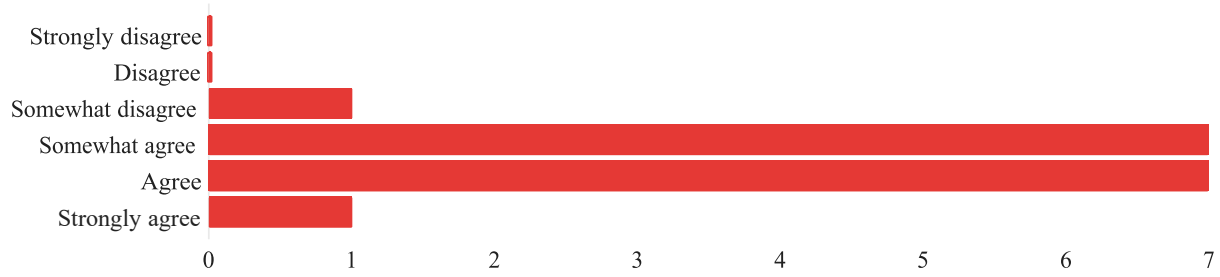
(Optional) Comments:

sound effects were good for the most part, though the combat sounds could get a bit annoying after a while

They feel a bit too loud and harsh

The sound effects made swinging at fake enemies fun-ish

Would you agree with the following statement: "The music contributed to my enjoyment of this game?"



Field	Choice Count
Strongly disagree	0
Disagree	0
Somewhat disagree	1
Somewhat agree	7
Agree	7
Strongly agree	1
Total	16

(Optional) Comments:

I really enjoyed the background music!

Couldn't hear it, sorry

adds to the atmosphere

music was good

(Optional) How would you describe this game to a friend?

3D hack and slash with simple combat mechanics

A 3D-hack-and-slash

you are a robot who's exploring this new place and you have epic fighting skills

movement felt good

I would describe this game as somewhat relaxing to just attack enemies and explore.

(Optional) Do you have any other feedback or comments you would like to share?

Seems like a great start to a project! I feel like better feedback to the player during combat would go a long way.

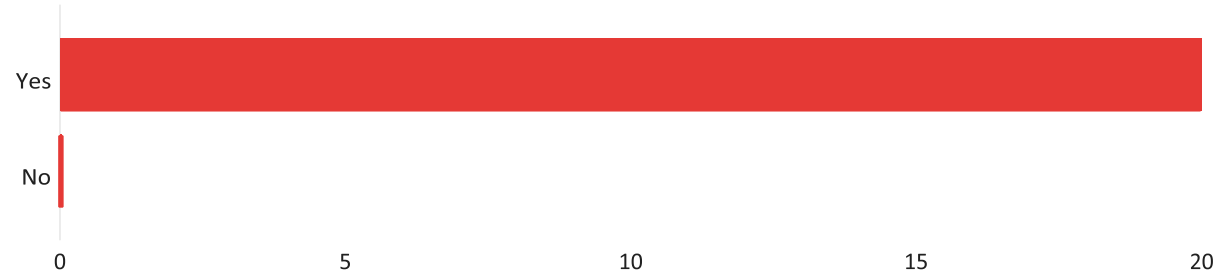
Keep it up! I love hack and slashes, personally. Polish is obvs needed but I like the idea of a robotic fairy (especially the art you had on the board)

The health bar sometimes became full, but not actually full, if you get hit again, you would return to the previous health

The motions and controls were quite smooth. It was very interesting overall.

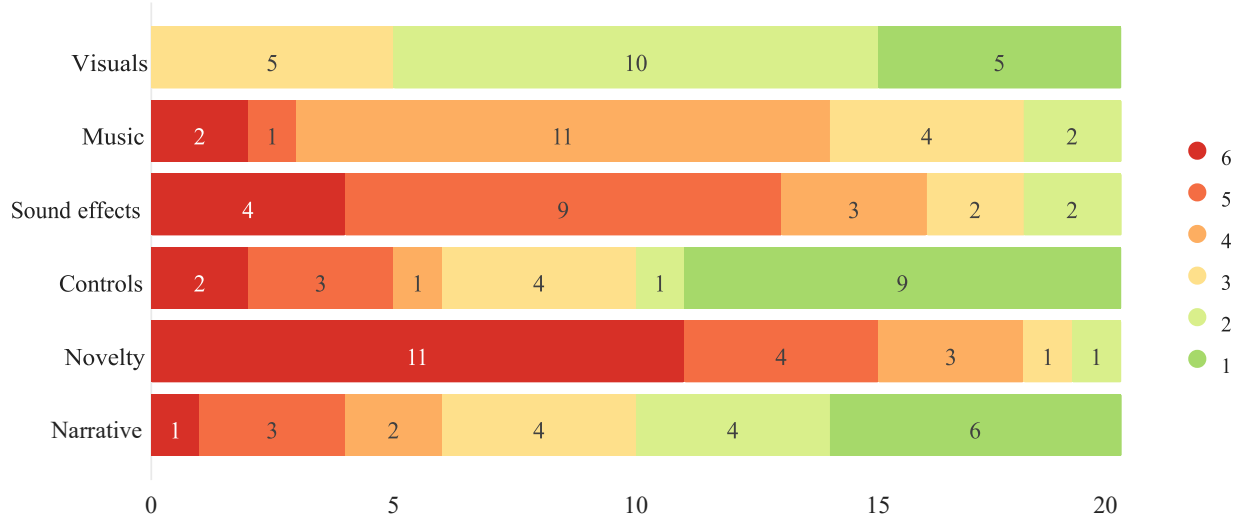
F. IMGD 2900 test data

Are you a WPI student/faculty member?



Field	Choice Count
Yes	20
No	0
Total	20

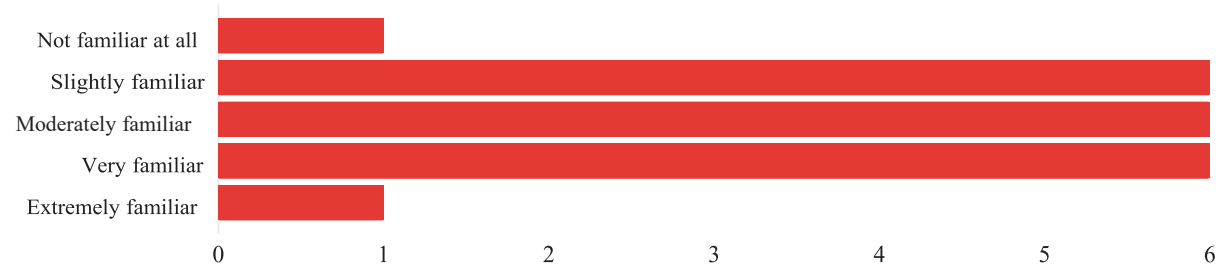
How would you order the following aspects of gameplay in general from most (1) to least (6) important to your experience? (Click and drag)



Field	Min Responses	Max	Mean	Standard Deviation	Variance	
Visuals	1.00	3.00	2.00	0.71	0.50	20
Music	2.00	6.00	3.85	1.01	1.03	20
Sound effects	2.00	6.00	4.55	1.20	1.45	20
Controls	1.00	6.00	2.70	1.82	3.31	20
Novelty	2.00	6.00	5.15	1.15	1.33	20
Narrative	1.00	6.00	2.75	1.58	2.49	20

Field	1	2	3	4	5	6	Total
Visuals	5	10	5	0	0	0	20
Music	0	2	4	11	1	2	20
Sound effects	0	2	2	3	9	4	20
Controls	9	1	4	1	3	2	20
Novelty	0	1	1	3	4	11	20
Narrative	6	4	4	2	3	1	20

How familiar are you with games of this genre (3D hack-and-slash)?



Field	Choice Count
Not familiar at all	1
Slightly familiar	6
Moderately familiar	6
Very familiar	6
Extremely familiar	1
Total	20

(Optional) Provide examples of other similar games you've played if applicable

Genshin Impact

Super Hot

A small amount of both Hyrule Warriors: Age of Calamity and RWBY: Grimm Eclipse

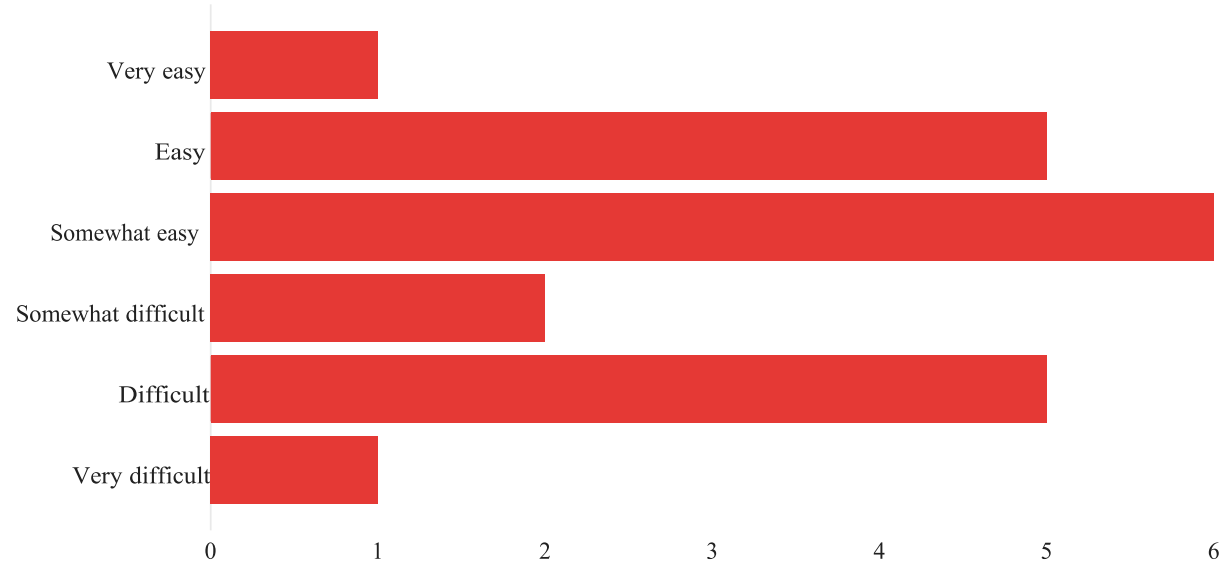
God of War, Souls games

Metal Gear Rising: Revengeance, Hades, Star Wars Jedi: Fallen Order

Were you able to solve the puzzle and reach the demo-end screen?

Field	Choice Count
Yes	10
No	10
Total	20

How difficult did you find the character to control?



Field	Choice Count
Very easy	1
Easy	5
Somewhat easy	6
Somewhat difficult	2
Difficult	5
Very difficult	1
Total	20

(Optional) Comments:

Trying to jump from platforms was hard.

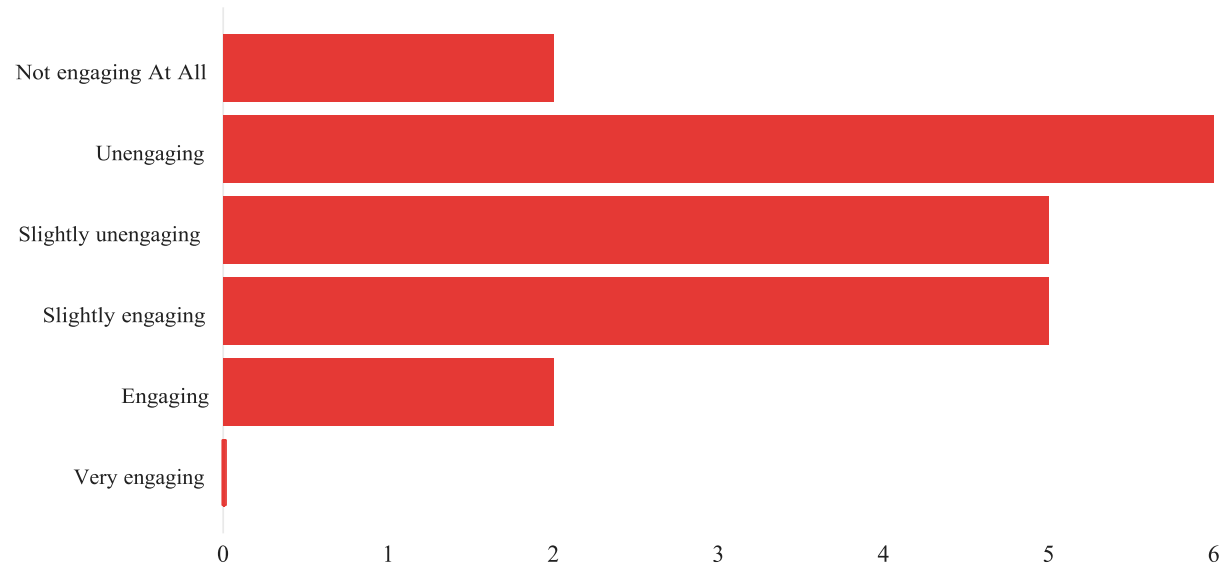
A state in the animation state machine could be canceled during actions when another is triggered.

The player stops whenever they attack. However, you can instantly start moving right after you stop.

1. The smoothing on the camera movement is too much. If it were me, I'd probably remove it altogether. But I would suggestdecreasing the smoothing at the very least.
2. When jumping while running off an edge, it would sometimes feel like my jump wasn't registering, which made me fallmultiple times. As a suggestion, adding a buffer to allow the player to jump for a few frames after leaving the platform would make this feel much more fair.
3. There was a section of floor jutting out of the ground right before the first big jump that you have to do at the start. Youcouldn't walk over it, so you had to jump on top of it to jump across the gap, which was awkward to do. Then, the issue I pointed out in (2) made it difficult to jump across.

There was some lag in the attack animation, but since there was nothing to attack, it didn't affect gameplay

How engaging did you find the game's combat?



Field	Choice Count
Not engaging At All	2
Unengaging	6
Slightly unengaging	5
Slightly engaging	5
Engaging	2
Very engaging	0
Total	20

(Optional) Comments:

I feel like the attack animation is too slow.

The seemingly inconsistent hit registration made combat feel unnatural.

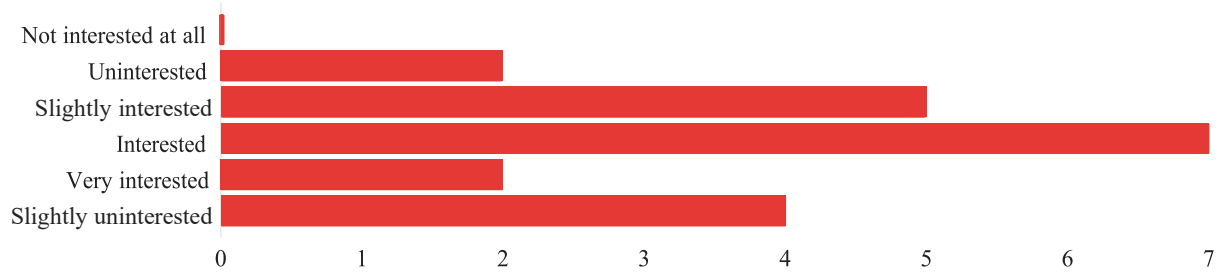
I could barley see the game, it was way too dark

1. I don't know exactly what the two bars in the top left represent. I can assume the top one is health, but I have no idea whatthe bottom one is.
2. The player's attack animation is very awkward and slow, and I couldn't tell when exactly I was attacking because of this.

Also, the lack of animations on the enemies meant that I had no idea when I was going to be hit.

I did not find any combat

10. How interested were you in the world and lore of this game?



Field	Choice Count
Not interested at all	0
Uninterested	2
Slightly interested	5
Interested	7
Very interested	2
Slightly uninterested	4
Total	20

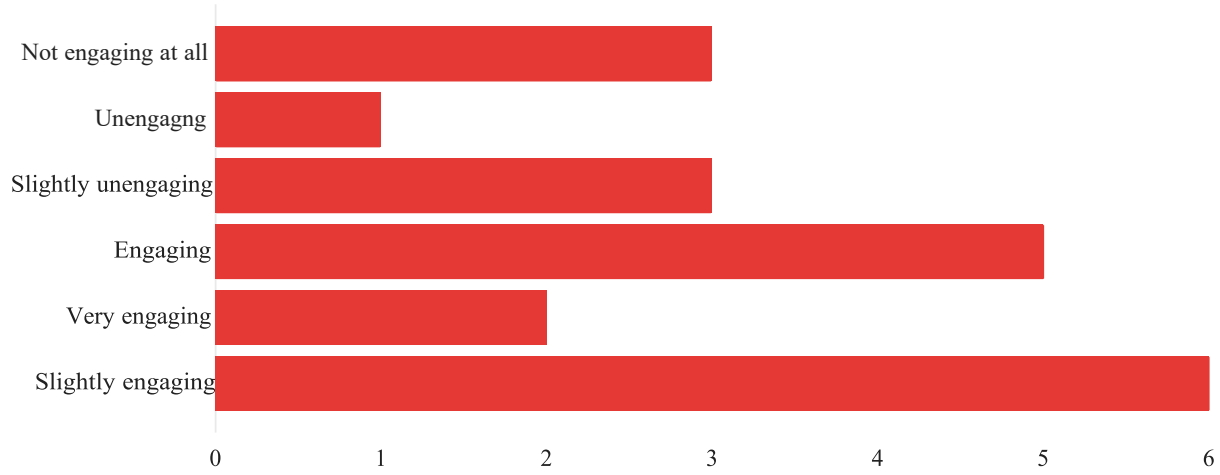
(Optional) Comments:

The overall idea and setting is nice. With a bit more color to the environment, it would be a lot more immersive.

The way it's described, Geo kind of seems to be a stand-in for "magic." But the name "Geo" makes me think it has to do with the ground or rocks. Which then makes me think that rocks are somehow inherently magical in this world, which seems really interesting to me from a world building perspective. Like, do people extract energy from rocks or something? How do they gather it? Also, what exactly can Geo do? How did it shape the "ancient society"? And what effect does it have in the modern day?

I am very interested in seeing how it develops! What is there already sets up a lot of mystery around what happened to the city and the geo power source, and why the player character isn't supposed to be awake.

How engaging did you find the puzzle?



Field	Choice Count
Not engaging at all	3
Unengagng	1
Slightly unengaging	3
Engaging	5
Very engaging	2
Slightly engaging	6
Total	20

(Optional) Comments:

screen was too dark to make any real progress, tried restarting, frozen pink screen of death that wouldn't go away no matter what I did.

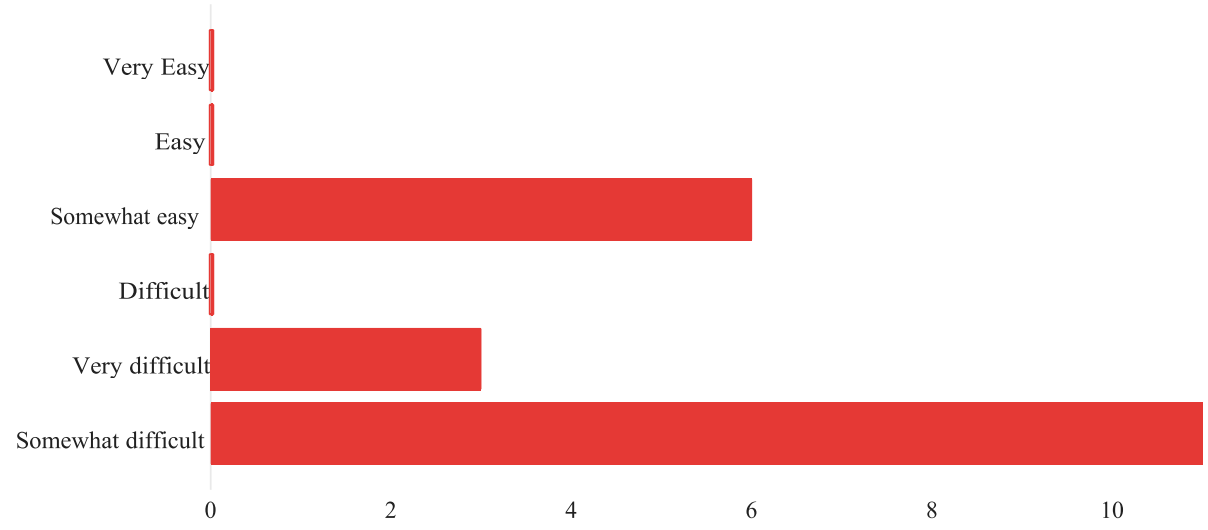
I don't know what the puzzle refers to. There was a jump I couldn't make so I couldn't continue in the game

Would be better if the success color for the puzzle was green instead of orange, since orange correlated with one of the puzzle pieces

That was really cool! My only issue with it is that it took me a long time to figure out what I had to do because I missed the dialogue explaining what to do and there were no reminders, so I was left to figure out what to do on my own. However, that system that you created to guide the player through the puzzle was really good! I was able to figure out what to do all on my own.

Not sure if I reached the puzzle, but in regards to exiting the door, the mouse sensitivity made it hard to go from platform to platform

How difficult did you find the puzzle to complete?



Field	Choice Count
Very Easy	0
Easy	0
Somewhat easy	6
Difficult	0
Very difficult	3
Somewhat difficult	11
Total	20

(Optional) Comments:

see 13

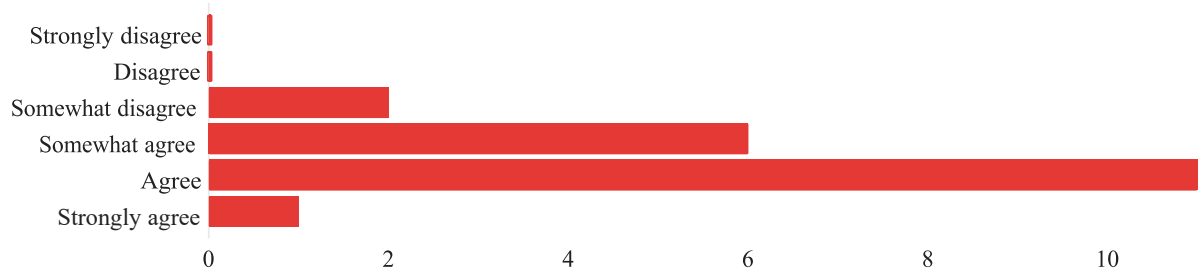
I think this is a good starter puzzle. I could tell it was based around audio pretty easily, and I didn't read the README beforehand (don't worry, I did sign the informed consent form beforehand). I had some trouble figuring out what each of the power cores did to the sound, though. I understood that they changed the pitch, and I was able to figure out that I was looking for octaves, but it was still trial and error after figuring that out because I didn't know what the cores actually did individually.
Also, I think it's worth noting that I am a musician, and I am quite familiar with music theory, so it was pretty clear to me that I was looking for consonant intervals. It was still a little difficult to hear, though, probably because the two notes were separated by an extra octave I think. Which means this puzzle may be too difficult for someone who hasn't gone through some sort of basic ear training through a music theory course or just by writing/playing a lot of music.

If the puzzle refers to the required platforming, the platforms were just slightly too far apart to be easily jumpable, especially for what I presume is the beginning of the game

It was a lot of trial and error even with a musical ear but it was very engaging so the monotony of constantly replacing geo blocks wasn't that bad.

Same as previous

Would you agree with the following statement: "The sound effects and dialogue contributed to my enjoyment of this game."



Field	Choice Count
Strongly disagree	0
Disagree	0
Somewhat disagree	2
Somewhat agree	6
Agree	11
Strongly agree	1
Total	20

(Optional) Comments:

The sound effects are great. However, the dialogues are too long, and some prevent me from moving forward. It would be greater for me if there can be a feature of canceling some part of the narrative or the narrative is shorter.

The dialogue felt off at times (felt a bit flat).

Well, the puzzle did require sound effects for me to know what was going on with it, and they definitely did a good job of that. The rest of the sound effects were good too. The voice acting on the dialogue was just okay, but the actual content of the dialogue got me very interested in the world that this game is set in.

I liked Seru's voice a lot. However, the fact that the dialogue repeats whenever you die got grating.

Iris was only coming out of my left ear, even when I moved the camera, so that was a bit frustrating at times, but using the audio for the musical puzzle at the end was very nice and contributed to my enjoyment a lot. Additionally, the dialogue setting up the mysterious story of the game contributed to my enjoyment and I am interested in knowing how that story develops.

Would you agree with the following statement: "The music contributed to my enjoyment of this game."

Field	Choice Count
Strongly disagree	0
Disagree	0
Somewhat disagree	1
Somewhat agree	9
Agree	10
Strongly agree	0
Total	20

(Optional) Comments:

Having the music that was in the game was definitely better than having no music, but it wasn't anything remarkable. It was nice that the music increased in intensity during the last fight before the puzzle.

I did not notice there was any music until I got to the second battle of the game, after riding on the tram. It was nice there, but didn't add that much.

I turned the sound off to concentrate

the music was pretty good

(Optional) How would you describe this game to a friend?

A 3D hack-n-slash platformer with an interesting setting.

3D platformer whereby the protagonist has been awoken by a mysterious floating friend? Your goal is to escape your homeland that has fallen into ruin and reach the surface.

It's a hack and slash game where you control a robot and their friend and explore an empty city and fight whoever doesn't want you snooping around.

A game where you wake up as an old robot, and have to find your way out of an abandoned facility

A post-apocalyptic sci-fi game where you're a robot who got woken up after some disaster and is trying to escape a facility for some reason.

(Optional) Do you have any other feedback or comments you would like to share?

It was frustrating starting over from the beginning when I would die. Especially falling off of platforms and respawning all the way at the beginning sucked.

Some parts are missing collision, such as the boxes in the area where you first see the droids.
Droids sometimes jitter and teleport when fighting them.

screen was too dark to make any real progress in the game, tried restarting, frozen pink screen of death that wouldn't go away no matter what I did including reopening the game.

I couldn't play the game because it was too dark, tried relaunching and only got a non-moving pink screen. ;/

Lighting in the beginning areas could help. Maybe some discreet lights to show the player where the path leads, since the scene often blends walls and other stuff together in darkness.