Shotoku's Defense: Physics Based Combat Using Motion Controls in Virtual Reality

Jose Antonio Li Quiel, Lisa Liao, William Lucca, Evan Plevinsky

October 11, 2019

A Major Qualifying Project Report
submitted to the Faculty of

WORCESTER POLYTECHNIC INSTITUTE

in collaboration with
RITSUMEIKAN UNIVERSITY

for partial fulfillment of the requirements for the
Degree of Bachelor of Interactive Media and Game Development
and Degree of Bachelor of Computer Science

Project Advisers:
Professor Ralph Sutter
Professor Joshua Cuneo

Abstract

Shotoku's Defense is a virtual reality physics-based action game developed for the HTC Vive. The player must survive against enemies across five different locations in a stylized and abstract traditional Japanese temple. Using motion controls, players must harness the power of earth-based elemental attacks to create various rock formations. The combat system is based on physics, causing faster movement to do greater damage. Development occurred across three months at Ritsumeikan University's Biwako-Kusatsu Campus in Japan. This paper details the design and implementation of the game.

Acknowledgements

We would like to thank Professor Thawonmas Ruck and Professor Hideyuki Takada for generously welcoming us into their labs and providing us with equipment necessary for this project. We also would like to thank Professor Haruo Noma for his hospitality during our stay in Japan. Additionally, we would like to thank our advisers Professor Ralph Sutter and Professor Joshua Cuneo for their guidance. Finally, we are thankful to Worcester Polytechnic Institute for sending us to Japan to work on this project.

Our Team

Shotoku's Defense was developed by four senior students from Worcester Polytechnic Institute. The team consists of Jose Antonio Li Quiel (Computer Science), Lisa Liao (Interactive Media and Game Development), William Lucca (Computer Science and Interactive Media and Game Development), and Evan Plevinsky (Computer Science). Jose programmed the game's enemy pathing and menus. Lisa designed and created assets. Will programmed the enemy behavior and created audio assets. Evan programmed the player mechanics and user interface.

Table of Contents

Abstract 2
Acknowledgements 3
Our Team 4
Table of Contents 5
List of Figures 8
1. Introduction 10
2. Background and Inspiration 12
  2.1 History 12
  2.2 Inspiration 13
    2.2.1 Historical Reference 13
    2.2.2 Video Games and Other Media 15
3. Technology 17
  3.1 Lab Environment 17
  3.2 Hardware 17
  3.3 Software 17
    3.3.1 Engine 18
    3.3.2 Source-Code Editors 18
    3.3.3 Art 19
    3.3.4 Audio 19
  3.4 Project Management 20
    3.4.1 Files 20
    3.4.2 Communication 20
    3.4.3 Version Control 20
4. Art 22
  4.1 Vision for Arena 22
  4.2 Arena 23
    4.2.1 Iterations 23
    4.2.2 Architecture 25
    4.2.3 Foliage and Pathways 27
  4.3 Lighting 28
    4.3.1 Time of Day 29
    4.3.2 Point Lighting and Fireflies 30
  4.4 Particle Effects 31
  4.5 Enemies 33
    4.5.1 Heavy Enemy 33
    4.5.2 Light Enemy 34
    4.5.3 Medium Enemy 35
  4.6 Animation 36
    4.6.1 Heavy Enemy Animation 36
    4.6.2 Light Enemy Animation 37
    4.6.3 Medium Enemy Animation 37
  4.7 Technical Challenges 37
    4.7.1 Animating Characters with Minimal Geometry 38
    4.7.2 Arena Ground Lighting 39

5. Design 40
5.1 Health, Energy, and User Interfacing 40
5.2 Player Controls and Abilities 41
5.2.1 Ability Creation 42
5.2.2 Ability Power-Ups 44
5.2.3 Healing 45
5.3 Enemy Behavior 46
5.3.1 Heavy Enemy 46
5.3.2 Light Enemy 48
5.3.3 Medium Enemy 49
5.3.4 Enemy Groups 51
5.4 Wave Spawning System 51
5.5 Level Design 52
5.5.1 Arena Decorations 52
5.5.2 Tutorial 52
5.5.3 Locations 53
6. Programming 58
6.1 Abilities 58
6.1.1 Ability Arc 58
6.1.2 Activating Abilities 59
6.1.3 Using Abilities 60
6.1.4 Controlling Power-ups 62
6.2 Enemy Implementation 63
6.2.1 Movement 63
6.2.2 Finite State Machines 65
6.2.3 Melee 66
6.2.4 Climbing 68
6.2.5 Strafing and shooting 69
6.2.6 Ragdoll, Get Up Animations, and Death Particles 70
6.3 Wave System 71
6.4 Tutorial 72
7. Sound 74
7.1 Audio Design 74
7.2 Sound Asset Creation 75
7.3 Ambience and Music 76
7.4 Audio Implementation in Game 77
7.4.1 Audio Sources 77
7.4.2 Enemy Spawn Audio Cues 78
8. Playtesting 80
8.1 Pre-Alpha 80
8.1.1 Testing Setup 80
8.1.2 Results and Conclusions 81
8.2 Alpha Testing 82
8.2.1 Testing Setup 82
8.2.2 Results and Conclusions 83
8.3 Beta Testing 84

8.3.1 Testing Setup 84
8.3.2 Results and Conclusions 85
9. Post Mortem 87
9.1 What went right 87
9.2 What went wrong 88
9.3 Future Development 88
10. Conclusion 91
References 92
Appendix A: Asset lists 94
Appendix B: Wave Spawns 102
Appendix C: Playtesting Screening Questions 104
Appendix D: Playtesting Surveys 105
Pre-Alpha 105
Alpha 108
Beta 112
Appendix E: Playtesting Changes Priority Log 117
Appendix F: Ritsumeikan and WPI Collaboration 124
User Interface 124
Hardware 125
Software 126
Conclusion 127

List of Figures

Figure 1. Bishamonten statue at Todai-ji in Nara 12
Figure 2. Shitenno-ji (Google Maps) 13
Figure 3. Wall and garden at Nijo-jo. 14
Figure 4. Mononobe warrior armor (McBride, 2011) on the left and kabuto (Reading, 2012) on the right 14
Figure 5. Tekkō on the left (Shimbukan Association) and chokutō (Kakidai) on the right 15
Figure 6. Toph from Avatar: The Last Airbender using wide movements while earthbending ... 15
Figure 7. Middle Ages Mine (Laryushin, 2017) 16
Figure 8. Low poly samurai asset pack (Synty Studios) 16
Figure 9. Man depicted with various levels of detail (Pontypants, 2017) 16
Figure 10. Shitenno-ji buildings in our game (left) and reference photograph (Soramimi, 2014) (right) 22
Figure 11. Complementary color schemes (Belenko) 23
Figure 12. Aerial views of Shitenno-ji (Google Maps) (left) and the game arena (right) 24
Figure 13. First arena design 24
Figure 14. Second version of arena, twice the size of the first 25
Figure 15. Lightning rod on pagoda (left) and roof horns on temple (right) 26
Figure 16. Boardwalk above pond (Mueller, 2005) (left) and in-game boardwalk (right) 26
Figure 17. Stepping stones leading to an entrance 27
Figure 18. Trees around, but not in front of, player's view of entrance 27
Figure 19. Foliage (Rad-Coders) and rock (SnowFiend Studios) assets from the Unity asset store 28
Figure 20. Arrangement of trees, bushes, grass, rocks, flowers, and mushrooms 28
Figure 21. Trees where enemies can hide 29
Figure 22. Texture for top of skybox 29
Figure 23. From left to right: tōrō in Osaka and in-game tōrō 30
Figure 24. Fireflies in the dark areas in our map 31
Figure 25. Small rocks appearing in the ground as wall rises 31
Figure 26. Quicksand's particle effects before appearing 32
Figure 27. Tōrō's flame particle effects 32
Figure 28. Early designs of the three enemy types 33
Figure 29. From left to right: Mononobe warriors' armor (Mcbride, 2011) and heavy enemy 34
Figure 30. From left to right: Mononobe tekkō and heavy enemy tekkō (Ryukyu Kobudo Shimbukan, 2014) 34
Figure 31. Light enemy shooting in action 35
Figure 32. From left to right: Japanese chokutō (Kakidai, 2018) and medium enemy sword 35
Figure 33. Medium enemy with bow (left) and with sword (right) 36
Figure 34. First iteration of the enemy models (from left to right: heavy, medium, light) 38
Figure 35. Normal maps creating the illusion that a smooth base mesh is faceted 38
Figure 36. Arena ground box (left) and one of the mesh slices used for light baking (right) 39
Figure 37. The normal Minecraft UI (left) attached to the bottom of the screen versus a VR Minecraft UI (right) floating in the world (Minecraft, 2008) 40
Figure 38. The UI and power-up icons used in-game 41
Figure 39. The Player Ability Ring surrounding the player with one valid Ability Arc 42

Figure 40. The player holding a small rock and a fully resized rock 43
Figure 41. A single spike from a small area vs multiple spikes from a larger area 43
Figure 42. A pile of quicksand after finished being created 44
Figure 43. A wall as it is being created 44
Figure 44. A line of spikes created from the spike power-up 45
Figure 45. Rock bouncing off heavy enemy if not powerful enough 46
Figure 46. Heavy enemy finite state machine 47
Figure 47. Light enemy is out of player's range 48
Figure 48. Light enemy state machine 49
Figure 49. Medium enemy climbing player's rock wall 50
Figure 50. Medium enemy state machine 50
Figure 51. The tutorial area (left) and a single section in the rock tutorial (right) 53
Figure 52. Final arena layout blueprint 54
Figure 53. Location 1's view from gate 55
Figure 54. Location 2's view from above (left) and player's view (right) 55
Figure 55. Location 3's view from above (left) and player's view (right) 56
Figure 56. Location 4'side view (left) and player's view (right) 56
Figure 57. Location 5's above view (left) and player's view (right) 57
Figure 58. The three states of the ability arc, from left to right: valid, interactive, invalid 59
Figure 59. Location 4 different area weights 63
Figure 60. Unity editor's NavMesh Surface component 64
Figure 61. Unity editor's NavMesh Agent component 64
Figure 62. Enemies remove NavMesh area when attacking (top) for others to path around (bottom) 67
Figure 63. NavMesh Link connecting the top of the wall to the floor 68
Figure 64. Man playing Ōdaiko drum (Prasanth, 2005) 74
Figure 65. Hyōshigi (Miya, 2005) 74
Figure 66. Creation of the wall raise loop sound in Reaper 75
Figure 67. Four audio sources surrounding the player 76
Figure 68. One audio source per object (left) vs. one audio source per sound (right) 78
Figure 69. Component that sets drumlings to use when spawning waves 79
Figure 70. Original map used for playtesting where player is placed next to Target one 81
Figure 71. The images provided to the player prior to playing the game 83
Figure 72. User interface with five different heart scenarios 124
Figure 73. Final proposed design for heart simulation 125
Figure 74. Reference model with all variables for the heart simulation 127
Figure 75. Final heart physical model 127

1. Introduction

Shotoku's Defense is a virtual reality (VR) physics-based action game where the player must defeat multiple groups of enemies across five different locations in a stylized and abstract traditional Japanese temple. Using motion controllers, players must harness the power of earth-based elemental attacks to create boulders, spikes, quicksand, and walls to defeat the enemies.

The game takes place in the Japanese temple of Shitenno-ji around 600 AD, immediately following the Soga-Mononobe battle at Mount Shigi. Prince Shotoku from the Soga clan warriors prayed to Bishamonten, god of war and earth, to win the battle and defeat the Mononobe. To honor Bishamonten for this victory, Prince Shotoku built Shitenno-ji, further angering the Mononobe and causing another battle to erupt. To reward Prince Shotoku's loyalty, Bishamonten grants him the ability to manipulate the earth to fight back the returning Mononobe warriors. The player of our game plays as Prince Shotoku utilizing these new abilities to defeat groups of enemies that invade the temple.

The first goal for our game was to provide the player with an immersive experience that would make them feel that the actions in the game reflects the actions that they are performing in real life. This led us to the idea of manipulating the earth through an assortment of abilities differentiated by player movements. Another goal was to take advantage of our travels in Japan by pulling from real events and locations to create a setting that would align itself with traditional Japanese culture. This prompted us to research different temples around Kyoto and Osaka as well as various historical events that could fit our description for the game, ultimately leading us to Prince Shotoku's story.

Our game was created in the Unity game engine due to its extensive VR support and documentation. It is also easy to learn for programmers since the base language is C# as opposed to an engine-specific language. Both of these factors proved to be vital for completing development given the 12-week development time. We worked out of two labs at Ritsumeikan University where we were provided with the HTC Vive to develop for. The game was built using the Steam VR library as it was simple to implement, provided base functionality for many different hand and player interactions, and was easily extensible and customizable.

Throughout the development of our game, we held three different playtesting sessions, each marking a different milestone for our goals: pre-alpha, alpha, and beta. Pre-alpha was

focused on the earth manipulation abilities and ensuring that the player was provided with powers that felt natural and fun to use. Alpha testing was focused on the enemies and the combat between them and the player. Finally, Beta testing prioritized the cohesiveness of the game, how well the gameplay fits the setting, and how capable the game was at providing an enjoyable experience for the users.

2. Background and Inspiration

The story for our game is inspired by real locations and events that provide a cohesive setting for the game. Additionally, we referred to different forms of inspiration to develop our game with a solid foundation. This chapter provides an overview of the history, culture, and popular references that our game draws from.

2.1 History

In the years preceding the Asuka Period (538-710) in Japan, the country was heavily plagued with religious conflicts surrounding the introduction of Buddhism. This caused a power struggle between two noble families: the Soga clan and the Mononobe clan. The Soga were supporters of widespread Buddhism in Japan while the Mononobe believed worshipping Buddha was disrespectful to the Japanese deities they had worshipped for centuries. While the two clans clashed many times due to this disagreement, one specific encounter solidified the Soga clan's superiority.

Figure 1. Bishamonten statue at Todai-ji in Nara

In July of 587 AD, a decisive battle for power occurred at Mt. Shigi in Nara, approximately 30 kilometers from Osaka (Umehara, 1980). The story goes that the Soga were losing the battle, and Prince Shotoku prayed to Bishamonten (Figure 1) for help with the promise that he would build a temple in Bishamonten's name if they won. The Soga successfully overcame the Mononobe at the battle and subsequently mitigated Mononobe influence in Japan prompting Shitenno-ji, or "Temple of Shitenno", to be established in 593 AD. Shitenno refers to the four heavenly kings of Buddhism, of which Bishamonten is the most prominent.

The structures of the temple have burned down in several fires and natural disasters. Most of the current temple is a faithful reconstruction of the original architecture. Today, it houses records, historical artifacts, and grounds for ceremony and celebration.

2.2 Inspiration

We looked to Japanese culture and historical sites as inspiration for the design of our game to keep our game's audio and visuals culturally accurate. Aspects of popular American culture found their way into our game while we gathered inspiration from outside sources as well, both in the form of television shows and video games.

2.2.1 Historical Reference

Prior to the development of the in-game arena, we visited Shitenno-ji (Figure 2) in Osaka to take photographs and experience the scale of the temple in real life. This trip proved to be incredibly helpful for creating the arena and architecture for the game as it gave us a basic design to work off and make into our own. Shitenno-ji is comprised of a rectangular courtyard with two temples and a pagoda. There is an entrance on each of the East, West, and South sides. The North wall is occupied by the large temple which houses Buddha statues and paintings. The large temple in the middle of the courtyard houses statues of the four heavenly kings and historical scrolls. The five story pagoda to the south of the large temple is the most symbolic of structures at Shitenno-ji, housing small Buddhist statues and golden tags surrounding a winding staircase. The entire courtyard is covered in gravel and has an enclosed roofed walkway along the edges.

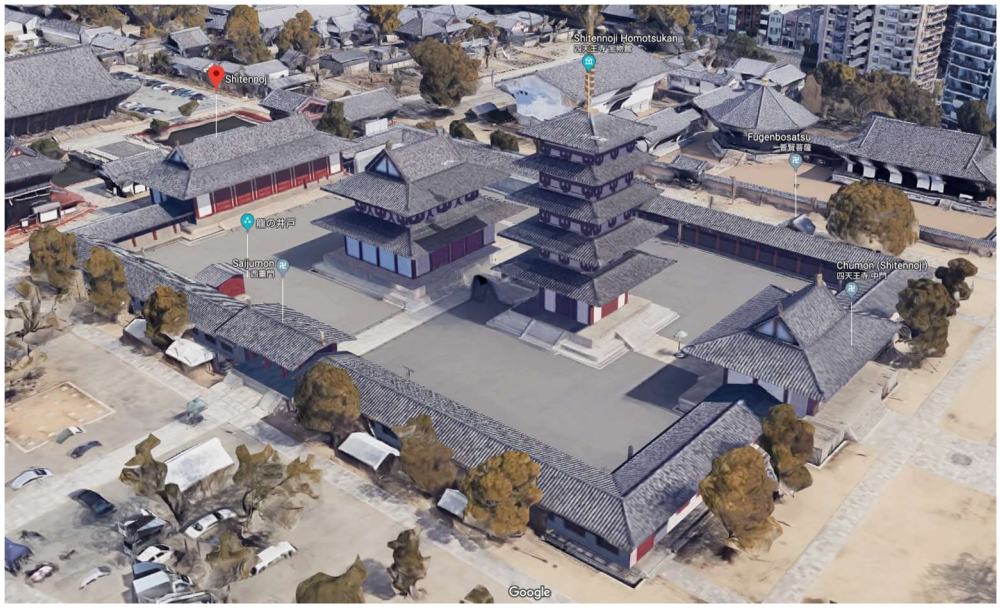

Figure 2. Shitenno-ji (Google Maps)

To gather additional architectural details and inspiration, we visited the Nijo Castle (Nijo-jo) in Kyoto. Nijo-jo was built in 1603 and served as housing for Tokugawa Ieyasu, the first shogun of the Edo period. The style of the castle's walls, gardens, and ponds (Figure 3) were exactly the traditional Japanese elements we were looking for, so we incorporated them in our game.

Figure 3. Wall and garden at Nijo-jo.

Since the basis of our game involves battling the Mononobe warriors, we sampled Mononobe's outfits for our enemy designs. The Mononobe were adept at battle and wore armor like the ones seen in Figure 4. We incorporate the *kabuto* (Figure 5)—a helmet with an antler-like decorative piece on the forehead (Bryant, 1991)—as a representation of an enemy's strength and boldness.

Figure 4. Mononobe warrior armor (McBride, 2011) on the left and kabuto (Reading, 2012) on the right

We sampled ancient Japanese weapons from our game's time period, but there is too little documentation to know that the Mononobe themselves used them. One of the weapons our enemies use is the *tekkō*, a semicircle that wraps around the knuckles and is used for punching (Figure 6). Another enemy uses a weapon called the *chokutō*, an ancient Japanese straight sword derived from similar swords found in China (Green, 2018) (Figure 7).

Figure 5. Tekkō on the left (Shimbukan Association) and chokutō (Kakidai) on the right

2.2.2 Video Games and Other Media

Humans having the inherent ability to manipulate the earth has been depicted in many different forms of media. One of the most popular occurrences of this is in the animated series *Avatar: The Last Airbender*. Characters in the show use the four elements of water, fire, earth, and air to engage in combat (Nickelodeon, 2008). While using these elemental abilities, a great deal of physical movement is used (Figure 8), and the action they perform changes the resulting attack. The movements in the show are much more complex than anything motion control tracking can mimic, but we took inspiration from the show's usage of wide, exaggerated movements in order to create different abilities.

Figure 6. Toph from Avatar: The Last Airbender using wide movements while earthbending

For our game's visual style, we took inspiration from artwork such as "Middle Ages Mine" by Vladislav Laryushin (Figure 7). We admired the level of simplicity that the artist was able to achieve whilst still conveying a story and background. We also liked the matte finish on the models that made the world look almost like it was made of paper.

Figure 7. Middle Ages Mine (Laryushin, 2017)

We wanted the characters to have exaggerated features and behaviors. The enemies each have a simple and recognizable silhouette as well as different colored clothing. The characters also take on the environment's abstract and faceted qualities. The enemies' abstraction comes in part from their lack of faces. The lack of facial expressions allows the character's body and actions to dictate personality. We decided to focus on those features because facial expressions require extensive fine-tuning to look believable and natural. We took inspiration from low poly art creators such as Synty Studios (Figure 8) and Pontypants (Figure 9).

Figure 8. Low poly samurai asset pack (Synty Studios)

Figure 9. Man depicted with various levels of detail (Pontypants, 2017)

3. Technology

Across our development time, we utilized as many resources as we could to simplify as many aspects of our project as possible. In this section, we describe our work environment as well as the different hardware and software that were utilized to create our game.

3.1 Lab Environment

Our team worked in the Creation Core building of Ritsumeikan University's Biwako-Kusatsu Campus. This building is primarily used for labs and offices for the College of Information Science and Engineering and houses a number of labs run by professors on campus. Our team was split between two independent labs where we were provided a workspace and, in one of the labs, desktop computers. The lab professors and assistant professors were available for us if we needed any equipment. We generally collaborated with each other in the larger lab space as needed.

3.2 Hardware

Across both of our labs on campus, we had access to two HTC Vive headsets, one of which being an early developer build version while the other was the first release version. The HTC Vive is a VR headset developed by Valve Corporation and HTC. Some of the headset specifications include a dual 1080p AMOLED display with a refresh rate of 90 Hz and a field of view of 110 degrees. Additionally, the system includes two base station cameras for tracking and two motion controllers ("VIVE Virtual Reality System", n.d.). Along with these, we had access to two laptops that were powerful enough to run our VR environment, one brought by a team member and one provided by a lab. Having access to this equipment was vital to our project's development as there were many features requiring frequent testing and tweaking through the player's input. The desktop computers provided by one of the labs were useful for intensive and lengthy lighting computations. One of the labs also provided us with a drawing tablet that allowed us to create more precise and original artwork.

3.3 Software

To develop our game, we needed to use an assortment of applications and frameworks that all worked together to structure our final product. This section discusses how we set up the

game engine, source-code editors, and asset creation software that we used for our game and why we used them.

3.3.1 Engine

The first major decision we needed to make was in regards to the game engine we would develop our game using. While our team knew of a few viable candidates, we ultimately decided on Unity 2019 version 1.11f since it has a reputation for being easy to develop for programmers who have never used a game engine before which was the case for all of our members (Unity, 2019). Additionally, Unity provides good support for VR development. Finally, coding for games developed using Unity is done in C# which some of our members were comfortable using.

When we first began work on our project in Unity, we set it up to use the Lightweight Render Pipeline (LWRP), because we needed to maximize performance and did not need high definition render quality. The LWRP is a render pipeline optimized for mobile and web browser performance. The LWRP allowed us to create a VR game without having to make big optimizations to maintain the necessary 90Hz framerate, allowing us to put more time into developing the game itself.

However, this decision, while ultimately necessary, did come with some unforeseen drawbacks. Unity implemented this render pipeline to allow for up to 16 lights on camera at any time. Our scene is full of torches called *tōrō* that both serve to set the scene and provide much needed illumination. Therefore, this method of rendering the scene caused some lights directly in the player's view to flicker on and off depending on which way the player was looking.

To simplify the development of the player system and the methods in which the player interacts with the world, we utilized the Steam VR library through the Unity asset store. This comes equipped with an extensive set of scripts that control the player and manage all of their interactions including picking up and throwing objects, tracking hand velocity, and structuring the collision area for the player. The scripts provided can be customized to add or remove functionality as needed making the library extremely extensible.

3.3.2 Source-Code Editors

We utilized two different source-code editing programs across the team: Visual Studio Code and JetBrains Rider. The choice between the two came down to the preferences of the

individual group members. In both cases, we were able to link the editor with Unity so they would automatically open when a script was selected in the Unity editor.

Setting up Visual Studio Code required the installation of a few extensions to the program. First, C# is not inherently supported by the software, so C# for Visual Studio Code was a necessary extension to include. To make the program ideal and easier to use, we installed additional extensions to add IntelliSense (code auto completion and auto correction), debugging, automatic commenting, quick formatting, and version control management. Finally, we installed Unity-specific extensions, which added Unity script recognition, IntelliSense features for the Unity scripts, and debugging for Unity programs as they run.

The other development environment we used, JetBrains Rider, comes packaged with Unity compatibility, eliminating the need for steps like the ones taken for setting up Visual Studio Code. JetBrains was made to work well with Git, C#, and Unity; it only falls short in its inability to edit and re-compile scripts as the game is running.

3.3.3 Art

For modeling and texturing game assets, we utilized Pixologic ZBrush and Autodesk 3dsMAX. These two programs complement each other to create assets that can be both freeform and calculated. Although our game does not feature high poly, detailed human models, ZBrush's sculpting tools were useful for creating early concept and prototype characters. We also used Mixamo's auto-rigging tool to create skeletons and download animations for our characters. We created 2D assets such as particle effect billboards, skybox textures, and background mountains in Adobe Photoshop, Adobe Illustrator, and Autodesk Sketchbook.

3.3.4 Audio

Finally, we created audio assets with the digital audio workstation (DAW) Cockos Reaper, and we found audio samples on *freesound.org*. We were previously familiar with Reaper, and the poor quality and selection of audio on an online soundbank required us to make many modifications, layered sounds, loops, and effects in the DAW to suit our game's needs. This workflow was used for all of our sounds (except for our music track, which we used as is) and proved simple and effective due to Reaper being a simple yet powerful program.

3.4 Project Management

Our team worked closely throughout the project and collaborated primarily in person on a day-to-day basis. This section discusses our methods of file keeping, communication, and version control.

3.4.1 Files

We maintained records of our work through a shared Google Drive folder. For keeping track of assets, we updated a spreadsheet of Art, Audio, and Tech asset lists throughout development (See Appendix A). We had additional subfolders where we kept daily progress logs, playtesting notes, priority checklists, and meeting agendas. To maintain development pace and budget time, we created a calendar in Google Sheets that we updated throughout development as well. Although the methods we used for keeping files was not sophisticated, they worked well for our purposes because the files are easily accessible from any of the computers we were using, and all of us were familiar with the resource beforehand.

3.4.2 Communication

Our primary methods of communication were the cloud-based collaboration tool Slack and email through a group alias. We had a Slack workspace set up for messaging and exchanging files between team members. This is another source for tracking work and files informally. Our advisers were members of the workspace as well and could access it at any point. However, our main communication and updates with advisers were through email. We had a video meeting via Google Hangouts and email update every week to provide updates, discuss changes, and share plans for future work throughout project development time.

3.4.3 Version Control

Having the teamwork across multiple different machines and many different tasks at a time made it necessary for us to utilize a version control program to keep track of our changes and assure everything works together. For this purpose, we chose to use Git. Additionally, we are more familiar with Git than other version control programs. Saving our changes to the remote

repository was achieved through both the git bash and GitHub Desktop applications varying by what each person was most comfortable with.

4. Art

Our vision for the game was to create a setting that accurately depicts Japan. We also wanted our characters to be distinguishable and stylized. We were able to accomplish this through online research, daily life in Japan, and in-person visits to cultural and historical sites. This chapter discusses the creation and implementation of architecture, foliage, enemies, and lighting of our game.

4.1 Vision for Arena

We wanted the environment of our game to be stylized and abstract. With this in mind, we decided to create objects with faceted surfaces and simple textures. Stylized and faceted geometry fits rock objects well, and the player's abilities all involve rock and sand objects. To be consistent with the theme of our game, we wanted the color scheme to lean toward reds and yellows that traditional Japanese architecture regularly utilize. The structures of Shitenno-ji are primarily red with purple-gray accents (Figure 10), which we maintained for the structures in our game.

Figure 10. Shitenno-ji buildings in our game (left) and reference photograph (Soramimi, 2014) (right)

The dirt and rocks occupy a large portion of the ground surface and are mostly yellow, brown, or gray. To break up the large portions of dull colors, we added contrast through the usage of bright green plants and decorated the scene with colorful flowers and mushrooms. The overall design of the game is consistent with the square color scheme shown in Figure 11.

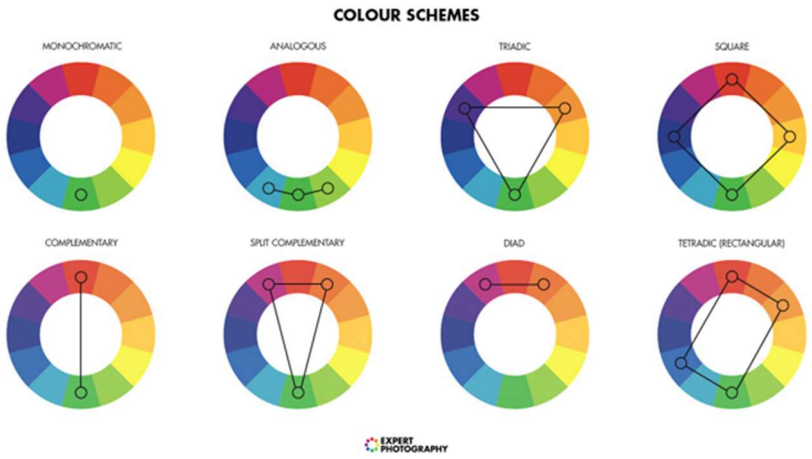

Figure 11. Complementary color schemes (Belenko)

4.2 Arena

Being in Japan for this project, we had the advantage of visiting cultural sights in person. Having seen some Japanese style architecture and design early on, we decided to create an environment that combines the aspects of a Japanese garden with that of a Japanese temple.

4.2.1 Iterations

After finding the story of Shotoku and Bishamonten, we decided that Shitenno-ji would be an ideal location to base our game off due to the simplicity of the temple space. A physical place in the world makes it easier to ground the game in reality compared to a generic Japanese garden. Shitenno-ji has two freestanding temples, three entrances, and one temple on the North side connected to the surrounding walls (Figure 12). We maintained the rectangular arena shape of Shitenno-ji surrounded by walls but replaced the large temple in the wall with a large entrance instead.

Figure 12. Aerial views of Shitenno-ji (Google Maps) (left) and the game arena (right)

As shown in Figure 13, the first design we had was an arena enclosed by walls that frame a temple on the North side. In this arena, there is a rock garden, a pond with a bridge, some sakura trees, and stone paths to guide the flow of enemies. At this phase, we did not fully understand the principles of virtual reality design, but we later learned that flat surfaces do not work well in a virtual world since the player is viewing the large platform at a relatively low angle, making the world appear flatter than it is. After visiting Shitenno-ji, we realized the actual temple was much larger than we expected. We expanded the arena to twice its original size, added fluctuations to the ground, and placed trees (Figure 14).

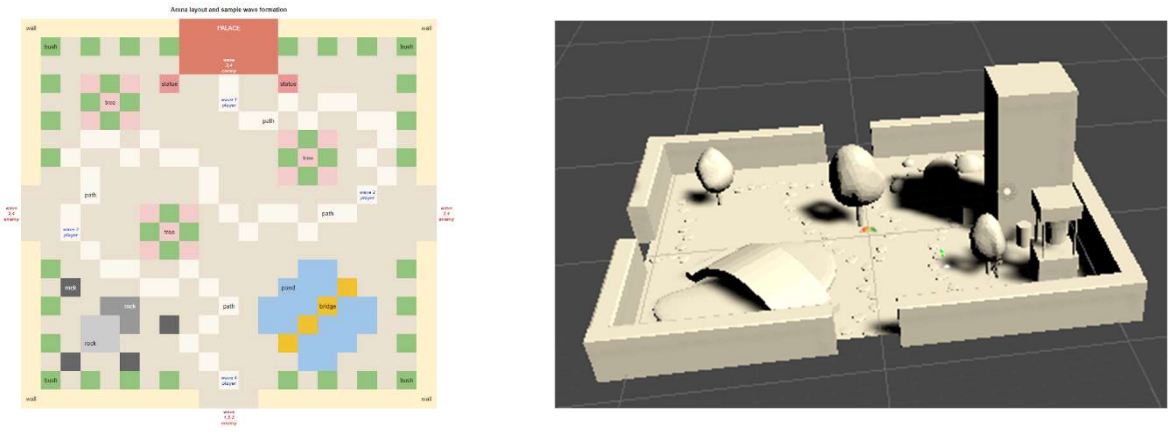

Figure 13. First arena design

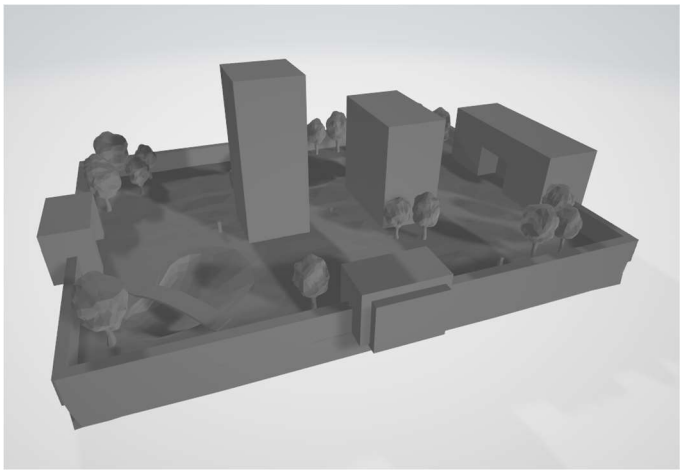

Figure 14. Second version of arena, twice the size of the first

The fluctuation we added to the ground only reduced the flatness minimally, so we decided to add tiers to the map, with each tier being 0.25 meters higher or lower than the one next to it. The arena, after several iterations, incorporates offset temples, tiered ground, and thriving foliage. The temples are offset for better viewing angles and the foliage creates natural pathways and divides the area into multiple regions.

We created the terrain based on each location's needs. The first location was designed to introduce combat, so the map is staged in a way that the player can focus on a single enemy entrance and not worry about any other sources of threat. Other locations with more complex combat called for changes such as wider spaces to allow enemies to surround the player or structures that would hinder abilities. In each location, the player stands on a raised platform of approximately three to five meters in radius to get a better perspective of the map and approaching enemies.

4.2.2 Architecture

The architecture of the game incorporates Japanese architecture from Shitenno-ji and Nijo-jo. The two buildings—the temple and pagoda—maintain the overall shape and colors of Shitenno-ji's large temple and pagoda. These two buildings fit our needs because they are distinct in both shape and design, which is what we aim for in our architecture and enemy designs. The lightning rod on the pagoda and roof horns on the temple are basic characteristics that make the buildings unique (Figure 15). The buildings both have wide bases with stairs that enemies can walk on for high ground and the multi-layered roofing that Shitenno-ji has. We

incorporated the Japanese Buddhist style curved roof (Figure #, same as above) in each of the buildings to break up straight geometry. This style of roof is believed to deter evil spirits that travel in straight lines. To further reduce structures that look too straight and artificial, architecture in the game is stylized in a manner that mimics aged material.

Figure 15. Lightning rod on pagoda (left) and roof horns on temple (right)

In addition to buildings and walls, we created a boardwalk, gazebo, and bell tower in the arena. The boardwalk occupies the pond, providing enemies with paths to reach the player. Traditional Japanese boardwalks (Figure 16) inspired this boardwalk. A gazebo was added in order to create dimensional interest to the relatively flat boardwalk and pond surface. The bell tower's main purpose was to fill an empty corner of the arena since bell towers are often seen near entrances to temples. The bell tower and walls of the temple were inspired by bells and walls found at Nijo castle.

Figure 16. Boardwalk above pond (Mueller, 2005) (left) and in-game boardwalk (right)

4.2.3 Foliage and Pathways

The foliage and pathways in each location were designed to lead player's eyes to entrances where enemies may emerge. Throughout the entire arena, there are stepping-stones leading from doors to the player's platform (Figure 17).

Figure 17. Stepping stones leading to an entrance

These are not necessarily the paths that enemies will take, but give the player a hint of the location of major walkways. Large trees aim to frame the player's view of entrances, divide barren wide angle views where desired, and fill in empty space around corners of the map where enemies are unlikely to enter (Figure 18).

Figure 18. Trees around, but not in front of, player's view of entrance

The trees themselves added some medium height geometry to our arena, but we needed bushes, grass, and plants to construct a more realistic ground surface. We used multiple versions of each type of foliage from the Unity Asset Store (Figure 19) to create diverse vegetation for our scene. We placed the foliage in a way that emulates real life. For instance, more grass and

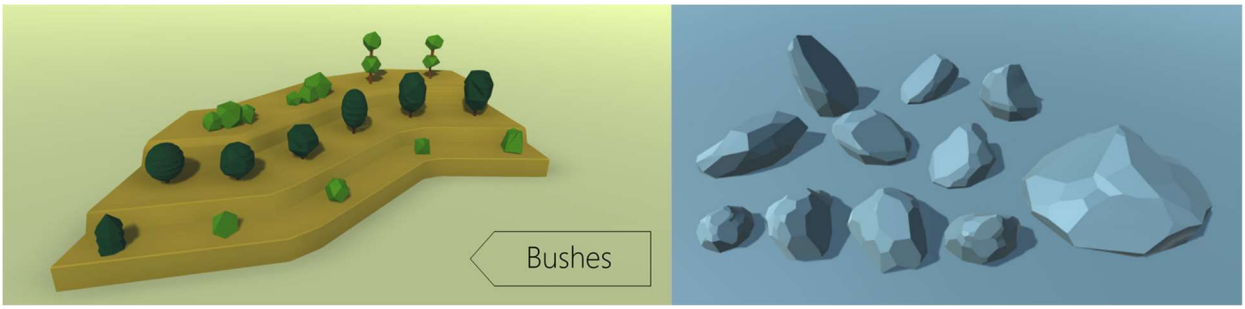

Figure 19. Foliage (Rad-Coders) and rock (SnowFiend Studios) assets from the Unity asset store

Figure 20. Arrangement of trees, bushes, grass, rocks, flowers, and mushrooms

rocks underneath trees and near bushes, mushrooms in corners and near bases of larger plants, and cattails and lilies around ponds (Figure 20).

Another benefit of low-level foliage is blending the changes in elevation where the lighting either creates a sharp edge or makes it difficult to locate the ledge. Lining buildings with bushes also helps to ease the 90-degree angle between the ground and walls. Grass and small pebbles serve to add depth to the flat ground and fill empty surfaces. The pruned bushes by entrances to the arena and temples serve to both decorate the flat walls and mimic the pruned bushes of Japanese gardens. We selected these bushes and other plants such as cattails because they appear often in Japanese scenery.

4.3 Lighting

Scene lighting strongly dictates the mood of the game and influences the environment colors. Our game utilizes nighttime, torchlight, and changing light intensity to create interesting lighting effects.

4.3.1 Time of Day

The overarching lighting theme in the game is deep night under a bright moon. Nighttime allows for darker regions of the map that would otherwise be too bright and empty in the daytime. However, since the entirety of the game's setting is outdoors, the moon allows enough lighting to see the enemies and abilities well. The lighting levels, tint, and direction were all fine-tuned throughout development to create ideal scenes in each location. An ideal scene would have well lit entrances and some dark regions in view. A well-lit entrance helps the player see the incoming enemy and better strategize and prepare for combat. Dark spaces under trees or next to buildings creates areas where the player cannot see the enemy's movement perfectly (Figure 21). The combination of well-lit entrances and separate dark regions adds difficulty to combat because the player can anticipate the enemy's overarching movement and combat styles but not always their moment-to-moment behavior.

Figure 21. Trees where enemies can hide

Initially, the game was set to occur during sunset. Sunset lighting typically provides more interest than regular daylight due to the low angle and warm tint of the light source. The decision to switch to nighttime was largely due to the color of the map. Most of the arena is occupied with warm colors and lights, so the warm sunset lighting further saturated those colors toward yellow. Tinting the light source to a cool blue helped balance the arena's colors and helped the red and purple temples stand out more because their darkness makes them less influenced by colored lighting. Nighttime also allowed for a starry sky and low poly moon, shown in Figure 22. To add

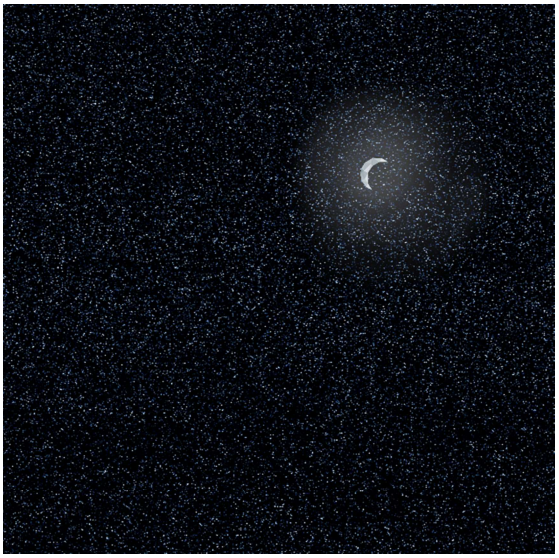

Figure 22. Texture for top of skybox

geometry to the sky and break up the large surface, we added clouds that move across the map at a slow and steady rate.

4.3.2 Point Lighting and Fireflies

In addition to the sky light, our game lights the arena using point lights—a type of light that emanates rays outward from a single point when illuminating objects. The light sources are designed to be standing *tōrō*, traditional Japanese lanterns commonly made with stone (Figure 23). *Tōrō* are used in Japan to line and illuminate pathways, allowing us to use them in this manner as well. The lanterns not only decorate the scene, but also illuminate enemies since they utilize real time lighting. We added interest to the *tōrō* through flickering lighting and flame particle effects. The point light pulses at a pleasant rate while blending between two flame colors, creating the illusion of a flame.

Figure 23. From left to right: tōrō in Osaka and in-game tōrō

To add variation to the shadows, we added fireflies and lanterns. Our fireflies gather in the dark shadows of dense trees. This simple addition makes still forests much more interesting and dynamic. The fireflies are non-interactive and do not emit any light, but they break up the darkness and stillness in some parts of the map (Figure 24). The lanterns line the edges of roofs

and add to the vertical geometry of the scene. Although they are only emissive and do not have point lights, they create artificial light in darker areas.

Figure 24. Fireflies in the dark areas in our map

4.4 Particle Effects

In an effort to add weight to our abilities, we created particle effects for the creation, destruction, and movement of assorted world objects. We felt that it was unrealistic for the ground to remain completely still and unturned while the player pulls out physical stone from it. The particles serve to mimic dust, dirt, and small rocks that are moved around as the earth is being manipulated. When the player picks up a rock, pebbles pop out of the ground to create the illusion that the ground has been broken. For both the spike and wall, cube particles vibrate at the base as the rock formation is being pulled out of the ground (Figure 25).

Figure 25. Small rocks appearing in the ground as wall rises

This mimics ground vibration and emerging dust. When rocks, walls, and spikes disappear, they break into particles of small rocks that fall outward, which look like the

formation is breaking up instead of disappearing into thin air. The quicksand evokes sandy particles that emerge from the earth toward the center of the quicksand to create the effect of gathering sand into a pile (Figure 26).

Figure 26. Quicksand's particle effects before appearing

In addition to the abilities, particle effects are applied on enemy death. The *tōrō* have particle effects for their flames as well to make them appear as if the light is coming from a real fire (Figure 27). Finally, we the fireflies in our scene were pre-made particle effects from the Unity asset store. All of these effects work together to make the world appear to be dynamic and natural.

Figure 27. Tōrō's flame particle effects

4.5 Enemies

The three warriors in our game are designed to be distinct but complementary in appearance. The heavy enemy is bulky, bold, and wears red like some Mononobe warriors; the medium enemy is fit, agile, and wears blue to contrast with the heavy enemy; and the light enemy is slim, cunning, and wears black to blend in with shadows. The concept art in Figure 28 shows early designs including weapons and enemy types. Some of these have since changed, but the overall appearance and behavior of enemies remain the same.

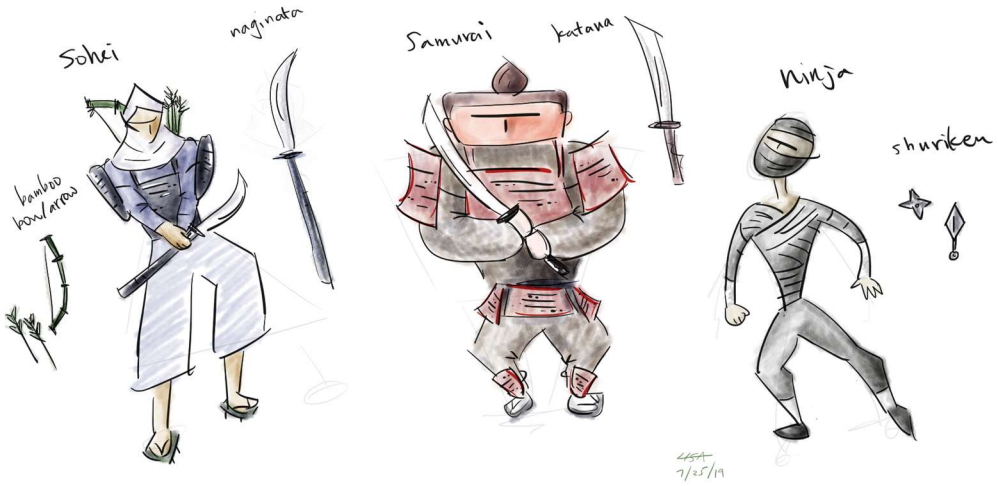

Figure 28. Early designs of the three enemy types

4.5.1 Heavy Enemy

The heavy enemy most closely resembles a Mononobe warrior (Figure 29) since they are wearing armor, boots, and a kabuto. The heavy enemy is designed to be a bold character that stands in their opponent's way. The large frame of the model helps convey the enemy's boldness, and since their body is "V" shaped—broad-shouldered with a narrow waist—they look larger up close. The body armor is red-brown in color and not incredibly metallic or sturdy. The skirt-like waist armor, an important characteristic of Mononobe warriors, behaves like a hard cloth so that it fits around the enemy's legs during motion.

Figure 29. From left to right: Mononobe warriors' armor (Mcbride, 2011) and heavy enemy

The heavy enemy wields the handheld tekkō (Figure 30) and goes straight for the player. The heavy enemy holds this short weapon to prevent adding any length to their already long arms. Since the weapons are small relative to the world, they are not detailed or heavily stylized. The heavy enemy holds the tekkō at all times, and the weapon acts as part of their fist.

Figure 30. From left to right: Mononobe tekkō and heavy enemy tekkō (Ryukyu Kobudo Shimbukan, 2014)

4.5.2 Light Enemy

The light enemy is the opposite of the heavy enemy. The light enemy is a cunning character that hunts enemies from the shadows. They are designed to be slim with long limbs in order to walk fast and crouch low, especially because they have long distances to cover. The

disproportionately long limbs also add to the character's odd persona. The light enemy wears primarily dark clothing to blend into dark shadows between trees and does not wear shoes in order to walk silently. The white cloth body wraps add depth to their clothing. The red headbands are a stylistic choice to add character (Figure 31).

Figure 31. Light enemy shooting in action

4.5.3 Medium Enemy

The medium enemy's build is between the heavy and light enemies' in terms of volume. The medium enemy is designed to be fit and agile, and able to adjust to situations quickly. The enemy is characterized by an "A" shape, with a small head and wide pants. The enemy's pants are dark blue to contrast with the reddish tones of the heavy enemy. The character's haircut seen in Figure # reflect Japanese apparel and style but not the Mononobe specifically. The appearance of the medium enemy, while done in reference to Mononobe warriors, was primarily designed in a stylistic fashion.

Figure 32. From left to right: Japanese chokutō (Kakidai, 2018) and medium enemy sword

The medium enemy's bow and arrow are the same as those of the light enemy. The medium enemy wields an additional weapon, the chokutō (Figure 32). Although the medium enemy uses the same bow as the light enemy, they wield the weapon differently. The medium enemy stands taller and has a wider range of motion compared to the light enemy (Figure 33).

Figure 33. Medium enemy with bow (left) and with sword (right)

4.6 Animation

Animations bring an incredible amount of characterization to a model. Creating or selecting animations becomes an important consideration in conveying the character's role in the game and interactions with the player. Our workflow consisted of modeling characters, rigging them automatically in Mixamo, making a list of what types of animations the enemy would need for in-game functionality, and selecting animations for that list based on the design of the enemy and their role in the game's combat. In the next three sections, we provide some examples of how this thought process affected the final animations of the game's characters.

4.6.1 Heavy Enemy Animation

The heavy enemy was made to be large and stocky: a big target with minimal maneuverability. They were also modeled with exaggerated proportions, so that they have a large, intimidating upper body and short legs. Because of this, the gait of the walking animation we selected appears sturdy and top-heavy with a short stride. We wanted this enemy to look strong and powerful while still being clumsy enough to be completely swept off their feet when hit with rocks.

The combat animations carry a similar characterization. They have two punch animations that carry all of their body weight in the swings of their fists. From these animations, it is readily apparent that this enemy deals plenty of damage, without even needing to be hit once.

4.6.2 Light Enemy Animation

The light enemy is the polar opposite of the heavy enemy in form, function, and animation. We selected their sidestepping animations to reflect their careful, calculated movements. They stay in a crouched position at all times, sneaking around the player and making themselves hard to hit. While sturdy, this enemy does not exude the same intimidating qualities as the heavy, so this animation does not give the impression that they would stand their ground if approached. This suits their function in the game, as they always remain in the background to shoot the player from a safe distance.

4.6.3 Medium Enemy Animation

Despite attacking from afar like the light enemy, the medium enemy's animations do not give off the same stealthy quality. The medium enemy is more mobile and confident in their attacks, so we selected a sidestepping animation where they stand upright, making the character appear prepared to take a hit and advance to the player.

Their sword swinging animations are consistent with this portrayal of their character. Their movements include quick footwork that moves them toward the player with swift slashes. These smooth and dynamic motions express their ease in performing both ranged and sword attacks, making their versatility on the battlefield readily apparent.

Finally, their climbing animations were selected to further express their high maneuverability. They quickly enter the climbing animation that takes them up an entire wall in a few steps. They then leap off the wall and resume sidestepping in one bound. Their swift and smooth motions here portray their ease in nullifying the player's walls when the player attempts to block them, adding to their character's urgent and overzealous nature.

4.7 Technical Challenges

During our process of developing assets, we encountered some technical issues in the game engine. Limitations in Unity's LWRP and Mixamo's auto-rigging system required us to find some creative solutions as a team when implementing some 3D models in our game.

4.7.1 Animating Characters with Minimal Geometry

The first iteration of enemies was our first attempt at stylized human models. We tried to create enemies that resemble rocks, which resulted in the enemies shown in Figure 34.

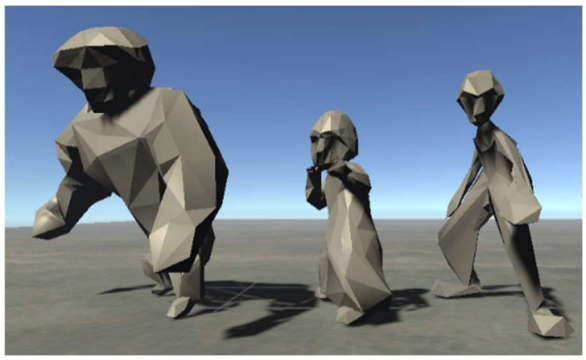

Figure 34. First iteration of the enemy models (from left to right: heavy, medium, light)

These enemies did not have enough polygons to support animated joints, so their geometry overlapped during animations causing their body to appear malformed. To maintain the desired style, we experimented with dense base meshes that used normal maps created from stylized models. Conventionally, normal maps are used on optimized models — models with as few polygons as possible — to create the illusion that it has more detail than is actually there. Our attempt to mimic simple geometry while maintaining fluid joints worked, but our models looked neither realistic nor stylized (Figure 35).

Figure 35. Normal maps creating the illusion that a smooth base mesh is faceted

After extensive experimentation with surface geometry, our final designs have few enough polygons to meet our expectations but enough polygons for proper animation.

4.7.2 Arena Ground Lighting

The arena ground was initially created as one large mesh shown in Figure #. However, we had to change this when we began experimenting with lighting. The Unity render pipeline we used can only accommodate four real time lights per mesh, which is too few for the entire arena. Our solution was to slice the arena ground mesh into sections of one square meter each, with each section similar to the one shown in Figure 36. This allowed each section of the ground to be sufficiently lit during gameplay.

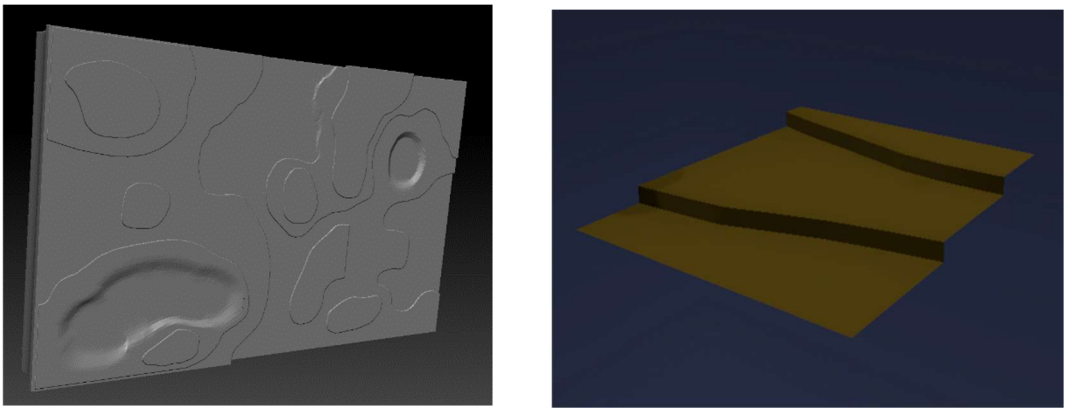

Figure 36. Arena ground box (left) and one of the mesh slices used for light baking (right)

Before the mesh is ready to be put in the game, it has to be unwrapped -- the process of mapping a 3D object to a 2D surface for texturing purposes. Slicing the mesh into squares was a detriment to the unwrapping and lighting process since it was difficult to hide the obvious seams along the squares. To account for this naturally, we adjusted the lights in a way that minimized color changes along edges and covered seams with foliage as needed.

For the ground collider, we decided to apply a mesh collider to the unsliced version of the ground. A single mesh collider is better than a couple hundred of meshes, but the lighting did not allow for the single mesh. Our result was using sliced planes for the render mesh and an invisible mesh with the same surface deformations right underneath the plane for the collider. The surface mesh was chamfered along the edges to ease the right angles for better lighting.

5. Design

Being an arcade style game, the combat had to be fast-paced and satisfying, the enemies had to be simple yet threatening, and the art had to be easy to look at and captivating. Therefore, we put a great deal of forethought into each of these areas to give the player the best experience possible. This chapter goes in depth regarding why the design decisions made for the game were the best for the player's experience.

5.1 Health, Energy, and User Interfacing

Before any form of combat was put into place, we needed to decide on the main goal and rules of our game. The goal would be to survive attacks, and limitations would involve a tradeoff for using abilities. This prompted us to implement a health and energy bar based system, where the player has a maximum amount of health and energy that drain as they take damage and use abilities, respectively. To regain health, we made a healing skill that costs energy in exchange for health. We wanted the player to be able to heal at any time they wanted while incurring a small penalty to prevent the game from being too easy. To regain energy, the player must not use their abilities for a short time, providing the player with a tradeoff for rapidly using their abilities.

Since managing the health and energy levels are necessary for the player's success, we were presented with the challenge of creating a comprehensive user interface (UI). In VR, it is not possible to attach images and text to the player's screen like you can for all other forms of gaming (Pan, 2017). Doing so causes the perspective of the player to constantly switch between

Figure 37. The normal Minecraft UI (left) attached to the bottom of the screen versus a VR Minecraft UI (right) floating in the world (Minecraft, 2008)

viewing the close objects on their screen and the far objects in the world, often causing the player's eyes to hurt. To avoid this issue, in-world UI objects are created and attached to physical objects. An example of this difference can be seen with Minecraft, as seen in figure 37. While there are many ways that developers for VR attempt to accomplish this, our team decided to create a UI that is attached to the back of the player's hands, allowing the player to refer to them at any point during the game. Additionally, we have icons that indicate the power-up bars for our abilities on both sides of the energy bar (Figure 38).

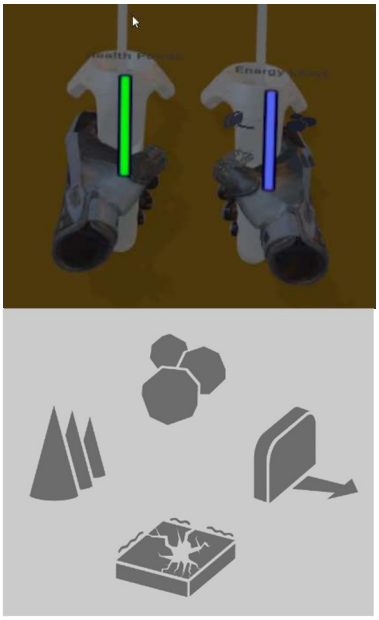

Figure 38. The UI and power-up icons used in-game

In traditional video games, menus are also part of the UI that get attached to the player's screen. For the reason described above, VR does not allow for this method of interaction. Instead, when the player pauses the game, we create a physical world object a set distance from the player that can be interacted with through a laser pointer coming from the player's right hand. This menu is automatically oriented toward the direction the player is looking to ensure that it is in the player's line of sight.

5.2 Player Controls and Abilities

Given VR's focus on physical player interaction with the world combined with the low number of input methods provided by the Vive controllers, the controls and abilities needed to utilize motion controls as much as possible. Furthermore, we wanted the player to feel as if the movements they were doing affected the earth as if they actually had these powers in the real world. As a result, our team implemented a system that minimizes the amount of button inputs and maximizes the player's physical control.

From a gameplay perspective, we wanted to have a diverse set of moves for the player to utilize. This gives them the power to figure out unique strategies for fighting off enemies and prevents the game from becoming stale. To accomplish this, our team developed four abilities for the player to utilize at any point during the game: rocks, spikes, quicksand, and walls. Additionally, the player is provided with a healing ability separate from these four core abilities.

5.2.1 Ability Creation

The player interacts with the world through an arc coming from their hand called the "Ability Arc". This arc curves toward the ground ending in a ring on the surface of the ground that identifies the ability's creation area. Both of the player's hands have an Ability Arc so that they can use their abilities simultaneously. Due to the player's powers being limited to interacting with the earth, abilities can only be created on ground that has a clear path for usage. Valid ability creation areas are marked with a green circle while invalid ones are marked with a red circle with an 'X' through it. Other objects in the world that interact with the Ability Arc show a yellow highlight around the object instead of a ring.

Each of the abilities drain the player's energy as they are created with varying energy costs based on the type of ability that was used and its size. The player's energy is replenished gradually after not using abilities for a short time. Abilities can only be created if the player has sufficient energy to do so. This functionality was put in place to challenge the player by preventing them from always using the most powerful abilities, ultimately adding to their experience.

In alignment with our minimum input philosophy, each of the abilities rely on the player using the controller's trigger button. This presented the challenge of differentiating between each ability. Additionally, spikes, quicksand, and walls are physical world objects that the player should not be able to walk through which cannot be prevented if the player creates a spike directly under themselves. To avoid both of these issues, we implemented a visible ring around the player that we refer to as the "Player Ability Ring." Within this ring, the player can create rocks that emerge from the ground and fly into their hand. Outside of the ring, the player can

Figure 39. The Player Ability Ring surrounding the player with one valid Ability Arc

create a resizable ability outline that creates spikes when they move their hand upwards and quicksand when they push their hands downwards. In both cases, the player only needs to press and hold the trigger to activate the ability and let go of the trigger to use it, with the rest of the interaction being through motion controls. The wall is also usable outside of the Player Ability Ring but requires one additional input. The Ability Arc and Player Ability Ring can be seen in figure 39.

When rocks are created, they are small and deal little damage to enemies. To increase damage from rocks, the player is able to resize them by holding a rock in one hand and pulling the rock outward with the other hand. Both the smallest and largest rocks can be seen in Figure 40. The player uses an increasing amount of energy, as the rocks get larger, with the smallest rock not costing energy and the largest rock costing 20% of their energy to create a tradeoff for utilizing this action. To incentivize movement by the player, the rocks deal increasing damage the faster they are thrown at an enemy. This is the cheapest of the four abilities and the primary source of damage. They can also be used to punch with by holding the rock and swinging at nearby enemies, also dealing higher damage at greater speeds.

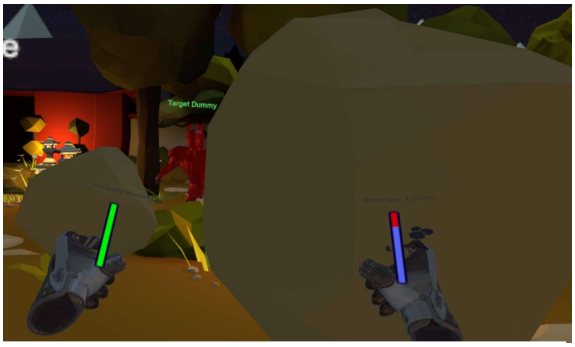

Figure 40. The player holding a small rock and a fully resized rock

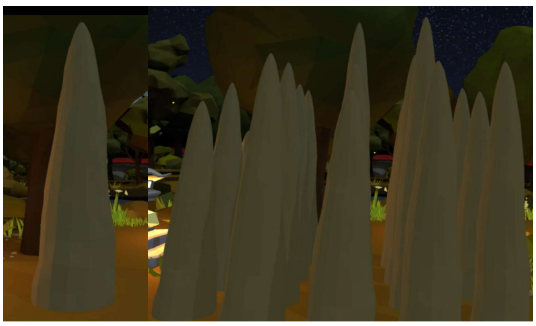

Figure 41. A single spike from a small area vs multiple spikes from a larger area

Similar to the rocks, the size of the area where spikes are created and the energy it uses increases as the player moves their hand upwards. As the size of the area increases, the number of spikes within the area increases as well (Figure 41). Upon releasing the trigger, spikes raise from the ground at a speed based on the player's hand speed. The spikes deal damage in relation to the speed of the spike, further incentivizing faster movement. While the spike can be an expensive way to spend energy, it does devastating damage to even the strongest of enemies. Moreover, spikes cannot have abilities created inside of them while they are active and are only active for a few seconds before being destroyed.

Unlike the previous two abilities, the quicksand (Figure 42) does no damage but instead slows the enemy by a significant amount. The size of the quicksand's creation area increases as the player pushes their hand down and at twice the rate of the spike's area for the same cost. This gives the player a reason to strategically place their quicksand in a place that allows for maximum growth with the most enemies affected. As to not obstruct the player and provide a greater benefit for using the quicksand, all abilities can be used from within its area.

Figure 42. A pile of quicksand after finished being created

Figure 43. A wall as it is being created

The primary uses for the wall are to block the paths of enemies and prevent ranged attacks from hitting you. This is the only ability that requires both hands to use, and for this reason, the team decided to add an additional, one-time input. To enter the mode where the player can draw their wall, the player must press both trackpads simultaneously. During this time, an outline of the wall is created which can be moved, resized, and rotated based on the location of the player's Ability Arcs. Upon pressing both triggers, the outline disappears and the player can raise their hands to pull the real wall out of the ground (Figure 43) ending at any height the player decides below a displayed maximum size. The energy drained by the wall is proportional to the length of the base and height of the wall. Walls that are obstructive to the player can be deleted at any time by pressing either grip button while aiming the Ability Arc at the wall.

5.2.2 Ability Power-Ups

While the abilities give many options for the player to work with, we wanted to give them a sense of progression with their abilities as well as additional temporary mechanics for them to take advantage of. For this reason, we created stronger versions of each of the four abilities. We also wanted to avoid overcomplicating the controls of the game, and as a result,

each of the power-ups are used in the same way as their base ability aside from a small addition to the wall. None of the abilities cost any additional energy to use.

The rock power-up creates four additional rocks when it is thrown, with each slightly veering off from the original in a random direction. These rocks mimic the properties of the original in both size and speed.

With the spike power-up, the ability creation area increases in number instead of size, which adds a line of additional ability creation areas with its length proportional to the distance the player raised their hand. Using the spikes causes each player creation area to spawn a chain of spikes that continue in the direction opposite the player until they hit something, with each chain acting independently from each other (Figure 44).

Figure 44. A line of spikes created from the spike power-up

The quicksand power-up creates an earthquake within the area of the quicksand that causes enemies to fall over. Any enemy outside the radius of the quicksand suffers a movement penalty proportional to how close they are to the quicksand for a short time.

The wall power-up pushes the wall in a given direction that, as stated prior, adds an additional action for the player to perform. Before completing the wall creation, the player pushes their hands in any direction they chose, causing the wall to move in that direction at a speed proportional to the speed of the player's hand at the time they release the trigger. The wall stops moving when it hits a stationary object.

5.2.3 Healing

To counteract the damage that the player would be taking from enemies throughout the game, we added a healing skill that regenerates health in exchange for energy. To use this skill, the player moves their hands close together without using any abilities and holds both grip buttons simultaneously. The team decided to require the hands being close together to force the player to focus on healing, providing an extra challenge that they need to account for. Both grip buttons were utilized, as it was a unique input that the player would be able to associate with

healing as well. Finally, players cannot heal while using abilities to force the player to plan their healing between attacks and create a strategy that allows for that period of vulnerability. The health they gain while healing is equal to the amount of energy they spend and occurs over time.

5.3 Enemy Behavior

After determining the abilities that our players would have at their disposal, we needed different enemy types to encourage them to use all of the abilities in combat. The different enemy types each have strengths and weaknesses in their statistics and behavioral patterns. These properties work together to create a variety of complex situations that force the player to think about how to defeat them effectively without being overwhelmed.

5.3.1 Heavy Enemy

The heavy enemies are highly aggressive and utilize only melee attacks (close-quarter combat), wielding spiked metal tekkō, ancient Japanese weapons akin to brass knuckles (Rich, 2015). We created this enemy in such a way that made the spike and quicksand abilities very effective, to encourage their usage when players are mostly using walls. Walls do not interfere with the heavy enemies for very long as they can easily walk around them, and rocks often do not deal enough damage to knock the enemies over (Figure 45). Quicksand and spikes, however, are easy to cast on nearby enemies, since they must approach the player. Spikes can also deal exceptional amounts of damage when clustered together on the enemies' large body, as each spike deals damage.

Figure 45. Rock bouncing off heavy enemy if not powerful enough

The heavy enemies walk slowly and deal a large amount of damage in a single punch with their tekkō once in range. They also have the most health out of any enemy, making them difficult to eliminate quickly. This means that players have the opportunity to ignore them until they are close, but leaving them completely unchecked might result in the player's quick death.

We designed the heavy enemies in this way to give the player choices when planning their attacks and to pressure them to make said choices quickly. This leads to more engaging gameplay and strategies.

The game uses finite state machines to control all of the enemies. The heavy enemy's behavior is dictated according to the following machine (Figure 46):

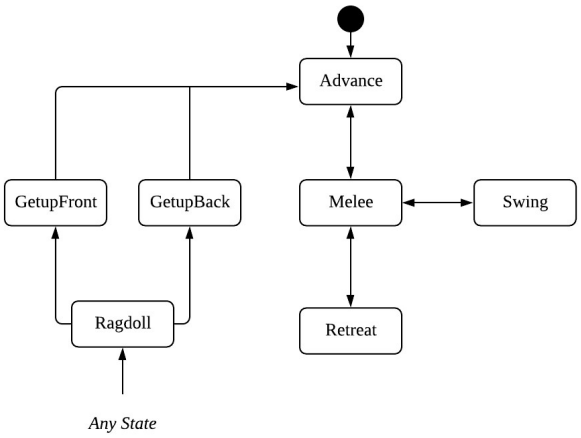

Figure 46. Heavy enemy finite state machine

In the Advance and Retreat states, the heavy enemy paths directly towards or away from the player, respectively. Within a certain radius, they stop walking and enter the melee state where they attack with their fists at a regular interval. This allows the player to move in the real world and have the enemies dynamically reposition themselves for attacking by advancing and retreating. When attacking, they enter the Swing state, in which they can damage the player with their tekkō. Their Ragdoll state disables their animations and lets the physics simulation control their limbs, causing them to go limp and flow freely until they finally settle on the ground. This leads directly into the `GetupFront` and `GetupBack` states, depending on how their bodies are resting when they try to get up.

5.3.2 Light Enemy

While the heavy enemies are very aggressive, the light enemies are the most defensive. The light enemy type is physically the weakest, but they compensate for it with fast, ranged attacks while keeping a safe distance from the player. We designed these enemies to stay just outside the radius where the player can create a spike or quicksand to force the player to resort to their other two abilities: rocks and walls (Figure 47). We decided to give them a small amount of health to compensate for the difficulty in attacking distant targets. This means that only one good hit from a rock is needed, creating a sensible level of difficulty.

Figure 47. Light enemy is out of player's range

These enemies' attack pattern is also ineffective against the player's walls as their only attack uses a projectile that is easily deflected. Our initial plan was to make them more mobile by having them climb the walls created by the player. Eventually, after seeing their behavior in the game, we decided it would not be useful to them. The player's walls act as a perfect defense against light enemies when they attack from afar, and the player is not able to create walls far enough away. This means that the light enemies would never have the opportunity to climb. However, even if they did, this ability would change what techniques work well against the enemy, unbalancing their strengths and weaknesses. This kind of complication would naturally be frustrating for players developing strategies to fight the light enemies.

A light enemy's behavior is dictated by the following finite state machine (Figure 48):

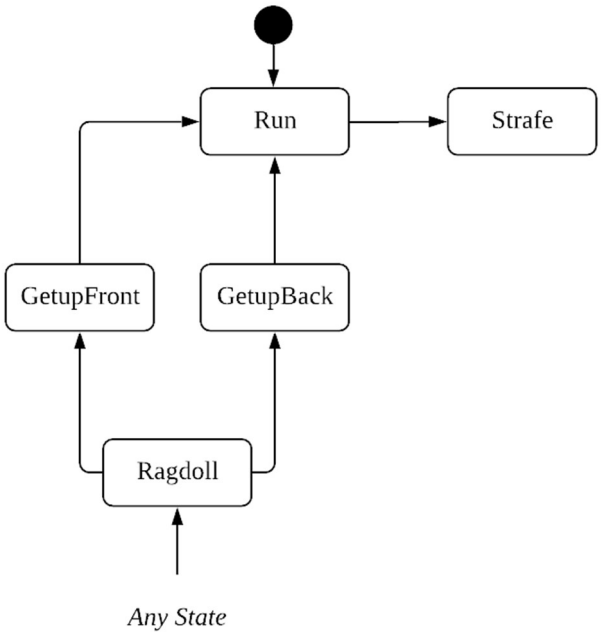

Figure 48. Light enemy state machine

In the Run state, the enemies follow the best path to the player before switching to the Strafe state once they are within a set radius. The Strafe state is defined as when enemies move sideways while facing the player. In the Strafe state, they shoot their bows at regular intervals and at a high velocity. This gives the player less time to react to incoming arrows, making the enemies more engaging and difficult to counter during battle. Lastly, like the heavy enemy, the Ragdoll state lets the physics simulation control their limp body before transitioning into whichever Getup state is more appropriate, finishing by returning to the Run state once again.

5.3.3 Medium Enemy

The medium enemies find a balance between the heavy and light enemies by wielding a bow for ranged attacks and a chokutō for close combat. They possess superior mobility among the three types of enemies, as they can walk side-to-side, approach the player, and even climb the player's walls (Figure 49). We moved this ability to the medium enemies because their steadily approaching movement pattern was more suited to it then the light enemy. In addition, the combination of strafing and approaching made the medium enemies just as susceptible to quicksand and spike attacks as the heavy enemy, only without the health and armor to hold their

ground, so they needed additional power to prevent them from becoming trivial. This additional pathfinding option affords new routes to traverse and requires more complex decision-making for the player when it comes to placing walls to block enemies physically.

Figure 49. Medium enemy climbing player's rock wall

To distinguish the medium and light enemies even further, we made their shooting arc behave slightly differently. To circumvent the player's walls yet again, the medium enemy shoots arrows in a large arc that will soar over most walls. This means that players will have more time to react to these arrows but will have to dodge or punch them, rather than block them with walls. While executed differently, all of these motions are easy to switch between, allowing players to get into a more interesting and challenging rhythm in the moment-to-moment gameplay.

The medium enemy's behavior is dictated by the following state machine (Figure 50):

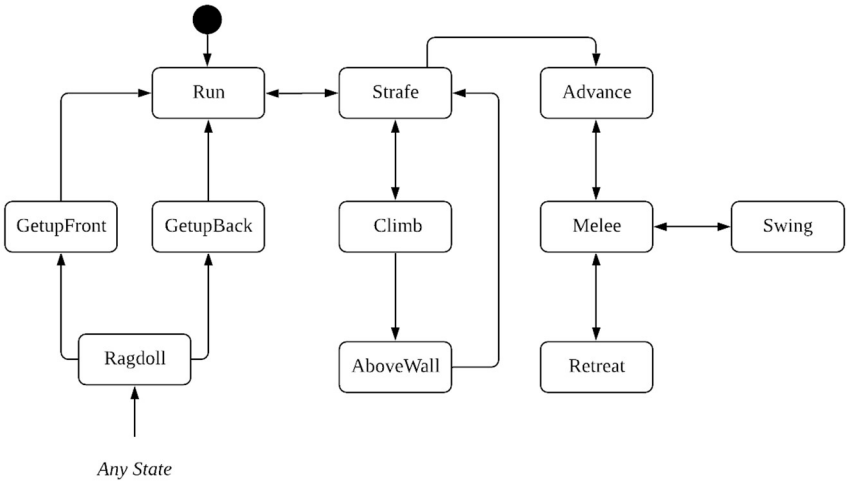

Figure 50. Medium enemy state machine

Run, Strafe, Advance, Retreat, and Melee all function the same as they do in the heavy and light enemies, only they approach the player as they strafe, taking them close enough to advance for melee attacks. Medium enemies can climb a wall when strafing, causing them to enter the Climb state, followed by the `AboveWall` state, and finally returning to the strafe state, allowing them to climb back down. Melee and Swing states also function just like their heavy counterparts. They recover from the Ragdoll state into a Getup state like any other enemy, finally transitioning it to the Run state to resume fighting.

5.3.4 Enemy Groups

Each type of enemy is easy to fight on its own by design, but together, they bring out much more intricate gameplay. The heavy and light enemies work well to complement each other's fighting styles, while the medium enemy exists to blur the line between them. The heavies are always marching forward like a tank, while the lights provide consistent damage from behind, and mediums add a little more confusion into the player's plans. The player must always figure out for himself or herself how to divide their attention between these different types of enemies to maximize their survivability. This pressure, together with the required snap decision-making, creates a gameplay loop in which players must utilize all their abilities in succession. This is why the enemies needed such distinct designs and attack patterns.

5.4 Wave Spawning System

Our goal during gameplay was for the player to get the feeling of being swarmed by an endless wave of enemies as they progress through the temple. To accomplish this goal, the team put a great deal of forethought into the design of our wave system, which is structured as follows.

First, each location around the map corresponds to individual information regarding the spawning patterns of enemies. Within a location, there can be any number of enemy waves. A wave can be defined as a grouping of enemies that spawn in succession without the player being given a break. During each wave, any number of enemies can be spawned from any of the four gates around the map based on the amount of time that has passed since the wave began. If the player eliminates all enemies on the map before the next group of enemies spawns during that wave, that group is spawned immediately. Once all enemies are defeated within a wave, the

player is provided with a short break before the next wave begins. Upon completion of all of the waves in a location, the player can take as much time as they need before teleporting to the next location and beginning another set of waves.

The wave system was designed in this way to allow for a highly extensible system for creating groups of enemies to fight at each location. The files that support this system are easy to update, allowing us to make quick and simple changes whenever there needs to be changes made to the spawning patterns of enemies.

5.5 Level Design

The goal for the game setting was to have a large arena with multiple locations where the player can fight. We used the same overall setting throughout the game to create cohesion while the change in locations allowed for more variety in both gameplay and visuals.

5.5.1 Arena Decorations

While laying out the map, we had to keep in mind that adding decorations such as foliage would minimize both where enemies can walk as well as where players can use their abilities. Since player abilities collide with large foliage, the foliage adds a level of difficulty by providing cover for enemies. However, they must not overly hinder player action or cause dead spots where the player has no way of attacking an enemy as this breaks the flow of the game. The decorations also take into account the player's wall ability. Certain walkways are narrow enough for the player to wall off temporarily, while other walkways are too wide to block entirely. In the latter case, walls are more useful for blocking projectiles than inhibiting enemy movement.

5.5.2 Tutorial

Due to the variety of abilities available to the player, we found it necessary to introduce each ability through a tutorial. At the beginning of the tutorial, the player starts with active Ability Arcs and a floating star that says, "Click me," teaching the user how to interact with world object interactives throughout the game. Next, the player gains abilities one at a time in the following order: rock, spike, quicksand, and finally walls. For each ability, there are a series of videos showing basic in-game footage of how to use the ability as well as instructional controller images showing the user which button to press to perform the action. Each ability tutorial

contains a section for power-ups as well, temporarily enabling the power-up for practice (Figure 51)

After all of the actions for a given ability have been taught to the player, they are presented with the options to do target practice, show the tutorial videos again, or start a wave to fight a single heavy enemy and practice their skills in real combat. At any point during the wave, the player can return to the tutorial area to review the instructions and restart the wave they are ready. Once the tutorial is complete, they are presented with another star that prompts the user to teleport, thus starting the game.

Figure 51. The tutorial area (left) and a single section in the rock tutorial (right)

5.5.3 Locations

The philosophy for the final arena was to create a sense of progression by starting the player at the front the arena at location one and ending on the other end at location five as seen in Figure 52.

Each location in *Shotoku's Defense* introduces new types of enemies and wave structures. Additionally, the game difficulty increases linearly as the player progresses from location to location. Each area was designed to be completed in around two to four minutes per location.

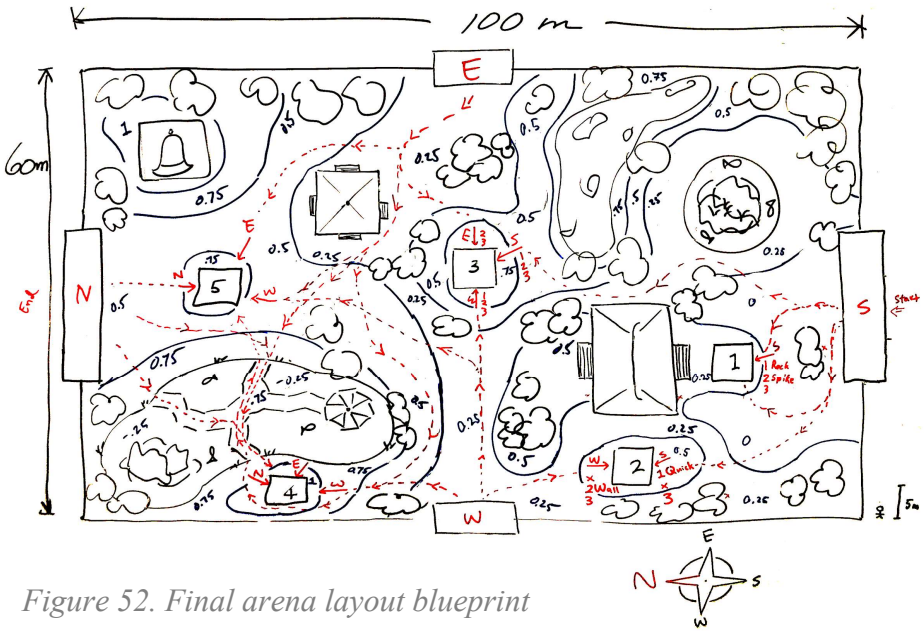

Figure 52. Final arena layout blueprint

Different locations encourage or discourage different abilities to challenge the player to strategize on how to use their different abilities.

Players will have to complete all the waves at each location to progress through the game. Each wave is subdivided into a number of rounds. Enemies spawn at different time intervals in each level. To promote fast-paced gameplay, if the player does not eliminate enemies in a reasonable amount of time, a new group of enemies will start spawning to overwhelm the player further.

The first location is primarily used for teaching the user how to use their abilities through the tutorial as described above. At this point, only heavy enemies are introduced to the player, as they are the slowest and easiest to deal with on their own. Additionally, the heavy enemies only spawn from a single location. Figure 53 shows aerial and perspective views of the first location.

Figure 53. Location 1's view from gate

Location two stations the player in a corridor between the arena walls and the South temple. Enemies will emerge from the South gate and East gate. This location introduces enemy spawning from multiple gates and attacking from multiple directions. The location starts with only one enemy spawning from the West gate to create a safe environment before easing the player in to paying attention to all gates and building the habit of checking surroundings.

The player also encounters the light enemy for the first time at this location. Light enemies strafe around the player in a circular path while shooting arrows at the player that can be blocked with walls and rocks. Light enemies combined with this location's small area aim to incite the player to use walls to block the enemies' line of sight, movement, and projectiles. Figure 54 shows aerial and perspective views of the second location.

Figure 54. Location 2's view from above (left) and player's view (right)

Location three places the player in an open space at the center of the arena. The player can be approached from every gate, making blocking enemy paths more difficult and forcing the player to look around frequently. The medium enemy appears for the first time in location three. The location's lack of obstacles combined with the climbing ability of medium enemies make walls weaker compared to location two to promote varied gameplay. Moreover, the team intended to use the shooting ability of medium and light enemies to help cover the weakness of the heavy enemy by distracting the player with projectiles while heavy enemies approach. Figure 55 shows aerial and perspective views of the third location.

Figure 55. Location 3's view from above (left) and player's view (right)

Location four is located next to the large pond between the North and West gate. Enemies spawn from the North, West, and East gates. The pond has a boardwalk on which enemies may travel to reach the player. The boardwalk limits the player's space to create walls and disallows the creation of spikes, walls, and quicksand on top of itself. This encourages the player to be more accurate with rocks, shield arrows with rocks, and evade attacks to maintain health. Players may knock enemies off the boardwalk with a rock and then follow up with other abilities. Figure 56 shows aerial and perspective views of the fourth location.

Figure 56. Location 4'side view (left) and player's view (right)

Location five is the final and most difficult level of the game. Enemies here spawn from the North, West, and East gate. The player is located close to the North gate, allowing enemies to reach the player faster. Similar to location three, the location is an open area where the player can be attacked from every direction. Timeouts between rounds are short to overwhelm the player by quickly spawning enemies one after each other. Figure 57 shows aerial and perspective views of the fifth location.

Figure 57. Location 5's above view (left) and player's view (right)

6. Programming

This chapter goes in depth on how the design of our game was ultimately implemented. The following sections detail the implementation of the player's abilities, the logic that the enemies follow, the enemy wave spawning system, and the tutorial at the beginning of the game.

6.1 Abilities

The player abilities serve as the primary form of interaction that a player has with the game. Development, creation, and use of the abilities changed as time progressed and as the desires of our players became clearer. This section details how we implemented the major ability mechanics, how we changed this over time, and why we made these changes.

6.1.1 Ability Arc

Before any of the abilities could be worked on, the player needed a way to interact with the world. The most natural implementation for world interactions in VR is through picking up and interacting with objects in the same way that you would in the physical world. However, this method was not applicable to our game since the actions the player would be performing are done from a distance. This lead us to finding other ways in which the player could accomplish that intractability in a way that still feels as natural as grabbing an object.

The next place we looked for a solution was the teleporting functionality that most VR games use to circumvent the limitations of small play spaces (Unity, 2019). In many cases, teleporting is done by creating an arc from the player's hand that lands on the ground showing a place where they are eligible to teleport to. This provides the functionality for interacting with the world from a distance that we were looking for, thus causing us to use similar techniques to activate our abilities.

The ability arc that we use in game is heavily derived from SteamVR's implementation of the teleportation arc. While the arc is active, it draws a dotted, curved line from the player's hand down to the ground, ending in a ring that turns green when valid and red when invalid. When the arc hits an object, a series of conditions are checked to assure that the object can be interacted with prompting the change in validity. From here, our implementation begins to differ with that of the teleportation arc. We started by changing the conditions to be that the object the arc hit was marked to be a valid ability usage area. This condition prevents the player from using

abilities where they should not be able to, such as on trees or under the bridge. At first, this was the only requirement that needed to be met.

Further into development, we needed to incorporate other ways to interact with the world, including picking up previously created rocks, destroying walls, and activating UI interactives. This caused us to expand on the conditions for a valid arc, as well as add a third state for the arc to show the player that they were interacting with an interactive object. The three states of the ability arc can be seen in Figure 58. Additionally, we needed extra information that a teleport feature had no use for, such as getting the end point's distance from the player, further prompting expansion of the system which gets utilized by the ability system.

Figure 58. The three states of the ability arc, from left to right: valid, interactive, invalid

6.1.2 Activating Abilities

The creation and manipulation of abilities is managed through the `PlayerAbility` script. This script is applied to both of the player's hands to allow them to use each hand independently. For each update loop of the game, the script checks for input from the user before performing any actions. If the trigger is pressed, the script attempts to use a new ability. If both trackpads are pressed, the script attempts to activate or deactivate the wall drawing mode. Finally, if the grip button is pressed, one of the following may occur: if the player's ability arc is touching a wall, they destroy the wall; if the player has an active ability, they cancel it; if the player is pressing both grip buttons, they heal.

On trigger press, the script checks for whether the ability arc is interacting with a UI interactive or attempting to create an ability. For the former, the ability creation scripts are skipped and the action dictated by the interactive is followed instead.

When attempting to create abilities, aside from checking for the movement criteria from the motion control, the distance of the ability arc to the player is taken into account. Within a certain range, rocks are the only abilities that can be created to prevent the player from entering

world objects as stated in previous chapters. This area of effect is marked with a white ring surrounding the player.

6.1.3 Using Abilities

Each of the abilities are separated into their own singleton scripts, with spikes and quicksand sharing a script due to the nature of their similar motion controls, making the abilities simple to manage and update iteratively. Each of these scripts interact with the `PlayerAbility` script in three major ways: creating, updating, and using the ability.

The rock ability can be created in one of two ways: by pulling the trigger while pointing the ability arc to the ground nearest to you or by picking up a previously existing rock. For newly created rocks, they either are pulled from a pre-populated list of rock game objects or are newly instantiated if that list is empty. Rapidly instantiating game objects at runtime can cause significant lag. By having a pre-populated list of game objects, we can avoid this issue entirely. The new rock gets attached to the player's hand through the Steam VR library and updates with the hand as it moves allowing for minor offsets from its original position before snapping back to it, thus giving the player the ability to punch close enemies. The location of object attachment to the hand that we chose is slightly in front of the player's knuckles. While this may not seem like the proper position for throwing, the way that location interacts with the pivot point on the player's wrists makes the rock fly in a much more natural way while throwing.

Once the rock is created, no regular updates occur until the player chooses to resize it, where it scales its energy cost and mass for damage calculations based on the size of the rock. When the rock gets thrown, the velocity gets scaled up to allow the player to throw farther and harder, and the game begins a five second countdown before the rock gets removed from the world and returned to the list of rock game objects. While the rock power-up is active, four additional rocks equal to the original are created with random directional velocities applied to them, splitting them apart at slightly different angles.

Activating the spike and quicksand abilities create a transparent blue ring showing the area of effect for those abilities. The size of this area of effect is based on the distance your hand travels while it is active: the greater the distance, the larger the size, but also the greater the energy cost. This adds a tradeoff to prevent the player from only using massive spike and quicksand areas. At the time of the trigger's release, spikes are created if there was an upward

movement in the hand, and quicksand is created with a downward movement. This combination not only allows the player to use two abilities with one action, it also gives them the opportunity to change tactics quickly when needed.

The formation of the spikes is based on the size of the ring the player selects and a predefined minimum size for each spike. In a majority of cases, a recursive function gets called to generate a list of positions in a hexagonal shape around a given point while checking that the point is both within the area of the ring and not already marked for the creation of a spike. Given a ring of a small enough size that is also larger than a single spike, a preset triangle of spikes is created instead. In both cases, any additional, unused space in the ring is used to resize the spikes to fill the entire radius. This not only adds visual interest to a more powerful spike attack, it also increases the power of the attack by increasing the number of spikes hitting an enemy at once.

While the spike power-up is active, these algorithms are replaced by a series of co-routines that create spikes in a line. These lines adjust to the height of the ground and stop creating spikes when one of a few conditions are met: the line collides with an object, the number of spikes in the chain exceeds the maximum amount, or if the spike chain cannot locate the position of the ground. In all cases, the spikes are taken from a list of pre-populated game objects the same way that rocks are, further minimizing performance issues. The new spike begins by moving directly beneath the ground, and it then moves toward a position two meters above it at a speed based on your hand's movement speed. Two seconds after reaching its final position, the spike disappears and returns to the list of spike game objects. This makes spikes short lived, heavy hitting attacks that should be used sparingly due to their high-energy cost.

Quicksand starts under the ground on creation as well but takes a fixed amount of time to rise allowing its particle effect to play properly and look more natural. At any point during its lifetime, objects that collide with the quicksand get their velocity drastically reduced. When enemies leave the quicksand area, their speed is reset to what it was before entering the quicksand, resulting in a smooth transition out. While the earthquake power-up is enabled, enemies caught within the quicksand are immediately sent into a ragdoll state, while all enemies in a preset area around the quicksand get their movement speed reduced. For both normal quicksand and the earthquake, the slowing effect is applied to enemies in the area for up to thirty seconds before the quicksand is destroyed. This prevents the player from covering the entire arena in quicksand rendering every enemy immobile.

Unlike the other three abilities, the wall is equipped with two sets of create, update, and use functions: one for the drawing of the wall and one for the physical wall. The player first enters the wall drawing mode by pressing both trackpads simultaneously. This causes a transparent wall to be created that has its center position as the midpoint of the player's ability arcs and a height correction to assure it begins on the ground. The wall also follows the rotation of the ability arcs, allowing the user to customize the position, size, and orientation of the wall.

Pressing the triggers prompts the set of actions for the physical wall to begin, starting with the creation function, which removes the outline of the wall and replaces it with a solid version below the ground. During the update of the wall, the height is increased proportionally with the height of the player's hands, reaching maximum height after raising their hands by one meter. Once pulled out of the ground, the player cannot return it to the ground as their energy was already used to perform the action. Additionally, any enemies that interact with it immediately begin to ragdoll. The wall is a static object that enemies will avoid unless the wall power-up is enabled. During this time, the wall will move in the direction of the player's hands at the time of release, ultimately stopping on collision with an object.

The wall in both cases lasts for 30 seconds before being automatically destroyed, preventing the player from permanently blocking enemies. The player can also press either grip button while the ability arc is pointing at the wall to destroy it manually. This was a highly requested action from our first round of playtesting.

For the spike, quicksand, and wall outline areas that are shown to the user during the creation state of the ability, the color of the rings dynamically changes, showing updates to the type of ability being used and the validity of that ability's area for usage. By default, the spike outline is shown in green, the quicksand outline is shown in yellow, and the wall outline is shown in blue. At any point, if the ability usage area collides with illegal items, the area will turn red. Some of these invalid areas include trees, the temple, tōrō, and the arena walls. Additionally, the ability area will turn red when overlapping the white ring that surrounds the player, disallowing the user from resizing abilities to within the play area.

6.1.4 Controlling Power-ups

The player entity in our game contains a `PowerupController` script that handles all activation and deactivation of power-ups. In any script of our program, the

`PowerupController` can be called to increment a counter for each power-up by any amount. As the counter increases, a bar on the player's right hand that represents each power-up increases to fill a representative image for the ability. Once that value surpasses a preset threshold, the power-up activates for a given amount of time and the power-up image emanates a cyan color. The power-up is then handled by a coroutine that decreases the fill of the bar over time, causing it to flash when it is nearly drained, and ultimately resetting the counter and bar once time expires. With its current implementation, the rock and spike counters increase based on damage while the quicksand and wall counters increase based on energy usage.

6.2 Enemy Implementation

The enemy's main abilities consist of movement, melee, climb, strafe, and shoot. Some of the technical challenges with animation include shooting while strafing and ragdolling. Moreover, this section describes the general implementation of the finite state machines.

6.2.1 Movement

To implement movement for enemies, the team decided to use Unity's built-in navigation system, the NavMesh Surface Component, due to its ease of implementation and extensive documentation. According to Unity's documentation website, a NavMesh describes the walkable surfaces of the world, allowing an agent to path from one location to another. This component is built automatically based on the level geometry (Unity, 2019). An important feature of the NavMesh component is the ability to assign different weights or movement costs to areas on the map. Through the weighting of areas, the team was able to encourage enemies to walk in particular ways, such as across the bridge at location four (Figure 59), on top of building platforms, and around pond areas.

Figure 59. Location 4 different area weights

In addition to area costs, foliage serves as a means to define enemy paths. Since obstacles dictate enemy movement, larger foliage such as trees and bushes are strategically placed in a way that creates interesting walking behaviors as opposed to straight paths.

Since the game has three different types of enemies with different sizes and movement patterns, the team implemented three different NavMesh Surfaces (Figure 60). Each NavMesh Surface needs a specific NavMesh Agent to traverse it. Therefore, we also designed one NavMesh Agent per enemy type.

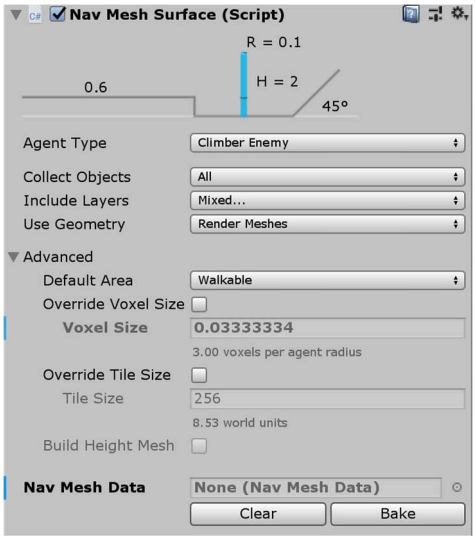

Figure 60. Unity editor's NavMesh Surface component

The NavMesh Agent is a built-in component that allows enemies to reach their destination through the NavMesh while avoiding obstacles and other enemies in their path. This agent component allowed the team to create varied enemy types easily by modifying parameters such as movement speed, and player avoidance radius (Figure 61). The paths agents can follow on the NavMesh can be different by setting different parameters on each agent.

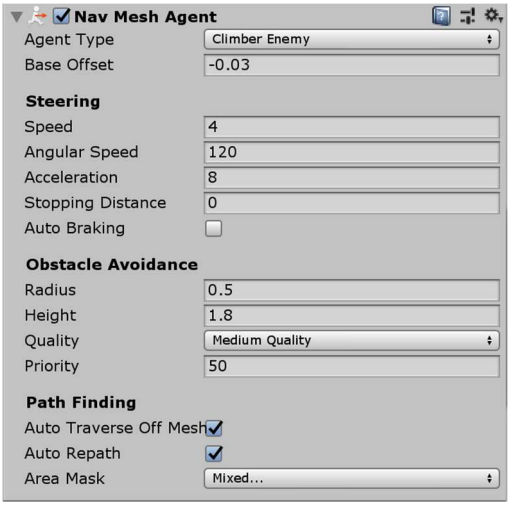

Figure 61. Unity editor's NavMesh Agent component

Unity also contains a NavMesh Obstacle component that describes objects that the NavMesh Agents should avoid while navigating the world. This component allowed the enemies to avoid obstacles in the map such as walls, tōrō, and stationary enemies in melee state.

6.2.2 Finite State Machines

Artificial intelligence for games can be handled with a variety of solutions, but for the simple behaviors of our three classes of enemies, we decided to use a finite state machine implementation. This way, we could create behaviors that change and respond to stimuli by changing states that could be written in a modular fashion. Our general structure for each enemy machine contains four methods:

- `Action()`: This runs every frame and performs the specified action described by the state (Run, Strafe, Melee, etc.)
- `Transition()`: This runs immediately after Action and checks if the machine can transition to another state
- `Enter()/Exit()`: These methods are called immediately after switching into/out of the given state to run setup functions for the state

Every enemy prefab contains a derived class of the `EnemyProperties` class, which builds a finite state machine with the states and properties necessary to define the behaviors of the enemies. For instance, the light and medium enemies both have a Strafe state that gets constructed on creation, but the `LightEnemyProperties` class defines its strafe speed as 1.6 meters per second (m/s), while the `MediumEnemyProperties` class define its strafe speed as 2.5 m/s. This method reuses code for common states such as strafe, melee, and ragdoll while still permitting us to tweak the properties for the enemies from the single classes that define each. Since each state contains a reference to the "Properties" class of its enemy, it can access functions shared by all enemies, which are defined in the parent `EnemyProperties` class. Some of these include a function that rotates the enemy to face the player when in melee or strafe state and another function to calculate square distance (a common helper function used to determine distance).

6.2.3 Melee

The melee combat abilities of the medium and heavy enemies required four special states in their state machines: Advance, Retreat, Melee, and Swing. The enemies enter Advance and Retreat states if the enemy is too far or too close to the player, respectively. We included this functionality because the player is able to maneuver their avatar in the game world by moving their body in real life. Having enemies keep a specific distance from the player specified by us ensures that their attack animations are always at the right distance to hit players. The enemy performs advancing and retreating by setting pathfinding destinations at the player's location or directly away from the player so that they can still maneuver on the terrain in these states.

When in the correct distance margin for a melee attack, the team needed a way to allow more enemies to approach and attempt to melee attack. The NavMesh Agent prevents enemies from passing through each other, but it does not force other NavMesh Agents to find a path around each other. This caused groups of enemies to become stuck in front of the player when those in the back could not find a path around those in front. Our solution to this issue was to have enemies disable their NavMesh Agents when attacking with melee weapons and have them put up a NavMesh Obstacle that carves a hole in the NavMesh, ensuring that other enemies path around them (Figure 62). This way, enemies intelligently see one enemy take up a position to fight the player and then path around that enemy to find an opening on a different side of the player.

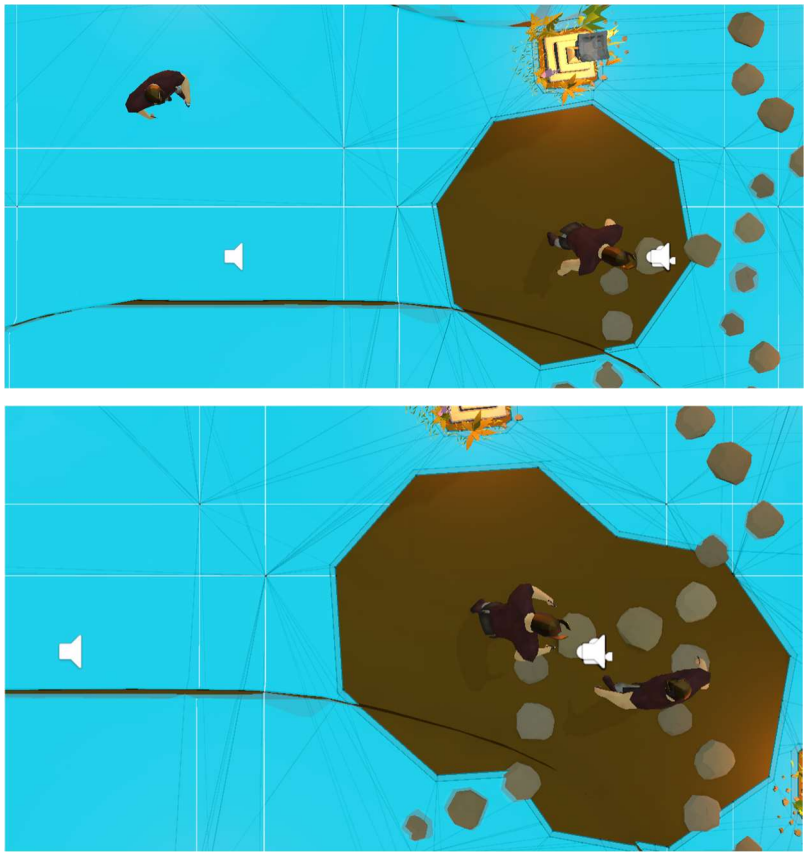

Figure 62. Enemies remove NavMesh area when attacking (top) for others to path around (bottom)

When in the Melee state, enemies attack at a regular interval, causing them to enter the Swing state. The enemy then plays an attack animation, and if any of their melee weapons collide with the player during this animation, the player takes damage. We encountered an issue with the player registering multiple hits if the enemy's weapon enters and exits the player's body multiple times in one swing. During testing, both our team and our testers found that this felt unpredictable and unfair. To solve this, we made each weapon only register one collision per attack animation. This meant that the heavy enemy, wielding two weapons, still has the chance to hit twice if the player is not careful. Therefore, this fix solves the issue of receiving melee damage while still maintaining the heavy enemy as a powerful melee fighter.

6.2.4 Climbing

To implement the climbing ability for the medium enemy, the team decided to use the NavMesh Link component. This built-in Unity component allows a NavMesh Agent to traverse between two different NavMeshes. Our plan was to create a wall, then re-bake the NavMesh to include the top of the wall. Baking the NavMesh means recalculating the walkable paths for the enemies. Afterward, we would instantiate the NavMesh Links to connect the NavMesh on top of the wall and the floor (Figure 63). This method prevents the team from having to manually make the agent jump or climb using physics.

Figure 63. NavMesh Link connecting the top of the wall to the floor

While implementing this method, the team encountered two challenges. First, when the player finishes instantiating a wall, the wall would re-bake the whole NavMesh at runtime causing efficiency issues. Second, NavMesh Links are usually baked before run time; therefore, we had to figure out how to instantiate the links on runtime at the correct coordinates from the wall. Furthermore, wall height needed to be calculated since it varies depending on arena height.

To address these problems, we first made the three enemy NavMesh surfaces ignore walls and then created a separate NavMesh Surface that only bakes on top of walls. Recalculating only this NavMesh at runtime is fast and does not slow down the program. The instantiated wall also has a NavMesh Obstacle component to indicate the enemies to move around the wall without the need to re-bake the other three NavMesh surfaces for the heavy, medium, and light enemies. Second, to gather the coordinates to instantiate the NavMesh Links, we used the wall's transform coordinates and raycasting. The wall coordinates, in addition to an offset, allowed us to get a position on the front, back, and top of the walls in which the links will start and end. By raycasting a ray downwards from the top of the wall, the link will know what the height of the wall is at any position in the arena.

Fortunately, NavMesh Agents have a method `isOnOffMeshLink()` which tells the program if an agent is traversing the link which facilitated the implementation of climbing

animations. One of the challenges the team faced was prohibiting the agent from climbing through NavMesh Links in quick succession. Climbing right away allowed enemies to move back and forth rapidly, which made the enemies appear to glitch across the wall. To solve this issue, we implemented a climbing state and an above wall state in the finite state machine. These two states keep track of how many times the agent has traversed a NavMesh link. Once the agent traverses two links, the agent is not allowed to use the links until two seconds later. The team also implemented a timer on the top of the wall that prevents the agent from moving while animations complete.

6.2.5 Strafing and shooting

Light and medium enemies strafe and shoot around the player to be harder targets to hit. The team decided to use the NavMesh agent component to facilitate their movement. A NavMesh agent can only take one destination at a time, so our challenge was to help the agent move in a circular manner. We decided to get the points of a circle around the player using the player's location as the center and a radius that we could determine programmatically. When the enemy reaches strafe distance, the enemy will calculate the points of a circle around the player and move towards the closest point. Once a point location is reached, the enemy recalculates a new set of circle points with a new reduced radius. In this way, the enemy can close their distance between the player while still moving in a circular manner. We made this radius as a public variable in our program for easy access through the Unity editor. Light enemies have a radius reduction of zero, meaning they always stay at maximum strafe distance. On the other hand, when spawned each medium enemy has a different radius reduction to make them harder to hit.

Light and medium enemies are capable of shooting arrows when at a distance. These enemies use a raycast that shoots a ray towards the player. If the ray registers a hit with the player collider, then the enemy is permitted to shoot. On the contrary, if the enemy's ray is blocked with walls or obstacles, they will not shoot the player.

To make projectiles easier to block and evade, arrows move in an arc motion. Arrows were designed to move slowly enough so that the player could react to the arrow. Using projectile motion formulas, the program calculates the necessary velocity in the y-axis of the

arrow to move at a given velocity in the x-axis. A white trail that follows behind the arrow was also created, making the arrow significantly easier for the player to spot.

The animations in this state proved to be particularly troublesome. The enemy is always facing the player when strafing and shooting, yet their behavior allows them to move in any direction. To solve this, the team created a two-dimensional blend of strafe animations in four cardinal directions, and then synchronized the feet up to the ground by making the playback rate of the animation proportional to the speed of the enemy itself. The enemies then needed to be able to shoot at the same time as this animation, necessitating an additional layer in the animation controllers. We assigned the light and medium enemies a layer that only blends into the upper body of their skeletons and, when they begin to fire an arrow, plays only the shooting animation. Finally, we noticed that this animation does not affect their hips, making them rotate slightly to the side of where they are trying to shoot. This is something we fixed by adding a simple offset to their rotation function as necessary.

6.2.6 Ragdoll, Get-Up Animations, and Death Particles

To give the player a sense of satisfaction, the enemy will ragdoll when hit with enough force. In order to implement ragdolling, the team first had to understand that ragdolling is not an animation and thus is not controlled by Unity's animation controller. Ragdolling is to leave the enemy's body fall to Unity's physics system without an animation. To follow Unity's physics system, each limb of the enemy has a rigid body and a capsule collider. Afterwards, our `ragdollController` sets each rigid body in the enemy to be kinematic, meaning they do not follow physics for movement and instead follow the animator controller. Once enemies are hit, the `ragdollController` disables the animator and then sets each limb to be not kinematic.

When enemies are ready to stop ragdolling, enemies will either die if their health reaches zero or enter the `getUpState`. When the enemy dies, the program first takes a snapshot of the enemy mesh and saves it to a variable. Subsequently, this mesh's transform, rotation, and shape are used to instantiate the death particles. These death particles mimic how the enemy looked when they were ragdolling, and then they disappear into the air. If the enemy does not die, the animator is reactivated and all the limbs are made kinematic again. The position of the hips is then checked to determine if the enemy should do a get up animation from the back or the front.

Once the animation is done, the enemy translates from the `getUpState` to the enemy's respective `resetState`.

6.3 Wave System

The wave system in our game depends on the game controller, enemy producer, and a series of queues for the locations. The team created an assortment of classes to organize the required information for each location.

First, time-based spawn information such as the types of enemies to spawn, quantity of each enemy, and location for spawning are organized together in a class called `SpawnInfo`. All the `SpawnInfo` are located in five different JSON files, each file representing a location of the arena. The JSON files for each location can be found in Appendix B. Each `SpawnInfo` is then stored in a `Wave` class which consists of a dictionary data structure. This dictionary uses a float representing the time for spawning as its key and a `SpawnInfo` as its value. The use of a dictionary allowed for quick retrieval of specific `SpawnInfo` based on time, as opposed to iterating through a list.

Next is our class `LocationWaves` which contains a queue of `Wave`. This queue represents the multiple waves of enemies at one location. Finally, our `gameController` script stores a queue of `LocationWaves`, each one representing a different location in the game.

The main logic loop of the game is handled by our game controller script. Upon launching the game, the queue of `LocationWaves` is initialized, then the first location and the first wave within that location are de-queued, and the wave timer is set to zero. The wave timer gets updated every frame, followed by a check to see if there is a wave available in the queue at that time. If there is an available wave, the program checks if there is a `SpawnInfo` at the current time. If a `SpawnInfo` matches the current time, the enemy producer spawns the corresponding new set of enemies. However, if there is no `SpawnInfo` at the current time, the loop restarts and updates the time.

The game controller also contains the function `OnDeathEnemyClear`, which is called every time an enemy is destroyed. If all enemies have been eliminated, the next `SpawnInfo` is spawned early. If there are no more `SpawnInfo`, this means the wave is finished, resulting in the start of a new wave in the current location. If there are no more waves left, the location is

completed and the player is prompted to teleport to a new location. If all locations are completed, the player has won the game, and they are provided with an option to return to the main menu.

The game controller also has a boolean variable called `pauseWaveSystem`. This variable cancels spawning from the update function. `TogglePauseWaveSystem` is a method that sets the value of `pauseWaveSystem` to the opposite of its current boolean value. This function is called inside an Invoke method with a timer to give the player a small break before starting each wave.

In addition to the game controller, there is an enemy producer script that handles all the enemy spawning with the help of a queue of `EnemyInfo`. `EnemyInfo` is a struct that contains a reference to the type of enemy to spawn and its spawn location. This queue is in charge of keeping track of enemies that could not be spawned in the game due to the scene already having the maximum number of enemies allowed.

6.4 Tutorial

The tutorial interweaves itself within the first location's wave system and is controlled by a singleton script called the `TutorialController`. At the beginning of each tutorial section, the `TutorialController` sets the wave system to be paused. This allows the player to view the tutorial as many times as they need before starting the real wave. Additionally, the player has a "Return to Tutorial" button, which resets the wave, returns to the previously opened tutorial, and pauses the wave system.

Our original design used buttons and a laser pointer to click through the tutorial. However, this was not a good way to learn for the user as found in our playtesting sessions. Therefore, the tutorial now appears the same as a normal UI menu but utilizes interactive world objects, thus allowing the player to keep their ability arc active and continue using their abilities. The activation of these interactive world objects begins within the `PlayerAbility` script. If this script detects that it is hovering over a UI object, control is transferred from the `PlayerAbility` script to another script that initiates the defined action.

The tutorial starts with only the rock ability active, preventing any of the other abilities from being created. As the player completes the tutorial waves, they are introduced to each ability one at a time. Once all abilities have been learned, the player moves to location two to

begin the real waves. The activation and deactivation of abilities is handled through the `PlayerAbility` script as well. This script contains a static variable for each ability that indicates the activation status of that ability, assuring that each instance of the `PlayerAbility` script follow the same rules.

During each tutorial section, there is also a portion that teaches about each ability's power-up. At this time, the `TutorialController` manually activates the power-up temporarily while also changing the color of the power-up icon to show it as enabled.

7. Sound

We began work on audio elements and asset creation during the final weeks of development and continued implementing it right up until the end of our work on the game. This chapter addresses our methods for developing and implementing sounds. It goes into detail on a few example sound effects and examines challenges we faced with unique cases and differences in VR.

7.1 Audio Design

Our main sound design goal was to emphasize the physical weight of our game objects and reflect real life actions in the game. This served to enhance the feeling of power that we hoped to achieve when designing the rock ability and motion control systems. The best solution to make the player feel powerful while using rock abilities was to create rumbles and crashes that emulate real-world stone. This meant that abstract sound cues, which have no simple real world sound that we could mimic, such as UI interaction and power-up timers, needed a realistic aspect to it as well in order to not sound out of place. We decided on using samples from Japanese instruments like *Ōdaiko*—a big drum with wooden beaters (Figure 64)—and *Hyōshigi*—wooden clappers like claves (Figure 65)—to reinforce the Japanese style of the game world.

Figure 64. Man playing Ōdaiko drum (Prasanth, 2005)

Figure 65. Hyōshigi (Miya, 2005)

7.2 Sound Asset Creation

Every sound in our game was found for free online, most often from *freesound.org*, then modified or tweaked in some way in our audio editor, Reaper. Our workflow for a single sound effect usually included discovering one or more audio files online that approximated a sound we were trying to create. For example, when the player pulls a wall up from the ground, the sound that plays contains three layered sounds originating from users on *freesound.org*. These tracks were modified with EQ — the process of changing the balance of different frequency components in an audio signal (Hahn, 2019) — to make each occupy different frequency ranges in the resulting waveform. This assures that each track is distinct and that the result is not busy-sounding.

First, a rumbling "earthquake" sound effect gives the wall a sense of weight and size as it slides, matching the massive stone that the player is controlling. On top of this are the sounds of sand settling in the highest register and pebbles falling in mid- and high-range frequencies. These effects mimic the fluid motion of the rock particles that the player sees emerging from the base as it breaks out of the earth. Finally, we made the tracks into a loop so that we can play it indefinitely (Figure 66).

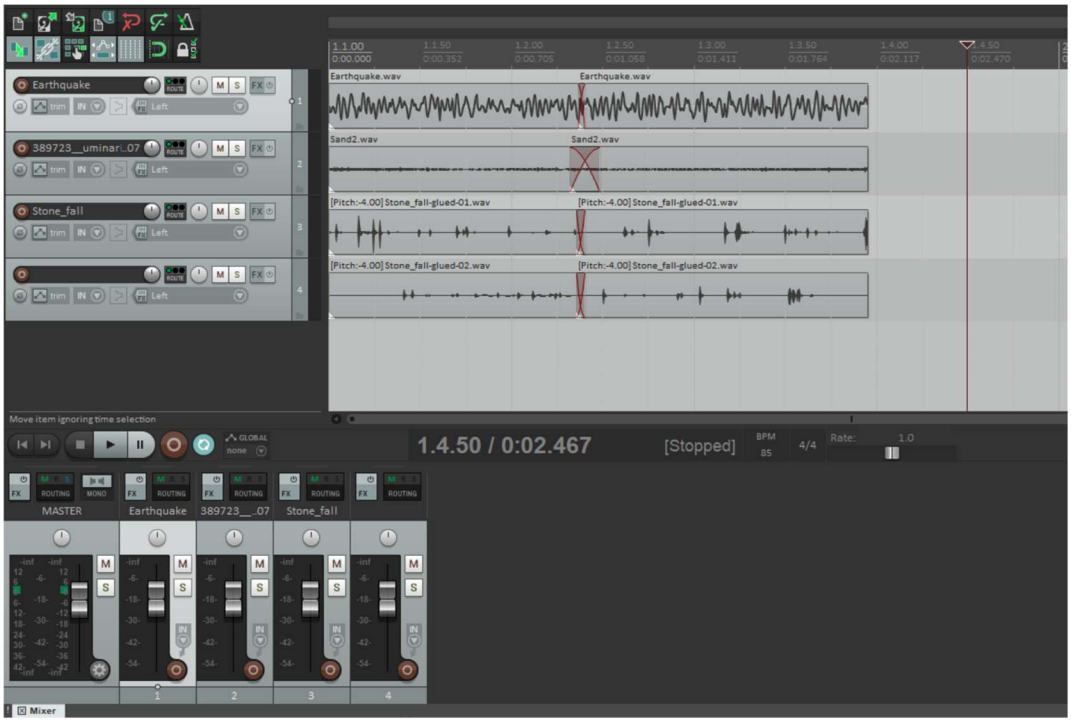

Figure 66. Creation of the wall raise loop sound in Reaper

Aside from the looping aspect, this process was repeated for nearly every sound in the game, then exported at 44.1kHz and 16-bit depth. We exported sounds that would exist in 3D space in mono so that Unity's audio engine could perform audio localization calculations itself. Conversely, we exported UI sounds, background music, and other sounds that should not exist in the world space in stereo to be fed directly to the player's ears without additional processing by the engine.

7.3 Ambience and Music

Virtual reality games have the unique property of requiring players to rotate their physical head and body in the real world. Games that do not use VR typically implement ambient soundscapes created in stereo and played constantly in the background as the game runs. Background sounds like music are non-diegetic—they do not come from a physical source in the game world—so this method is acceptable in VR as well. However, this does not work for diegetic ambient loops like crickets and cicadas in the grass, because they should realistically be emanating from a real 3-dimensional point in the game world (ustwo Games, 2015).

Our solution to this was to use four audio sources that constantly follow the player's head while remaining oriented in the same cardinal directions in the world (Figure 67). Each plays a different ambient soundscape in mono, and the engine makes them sound like they are originating from real locations within the world.

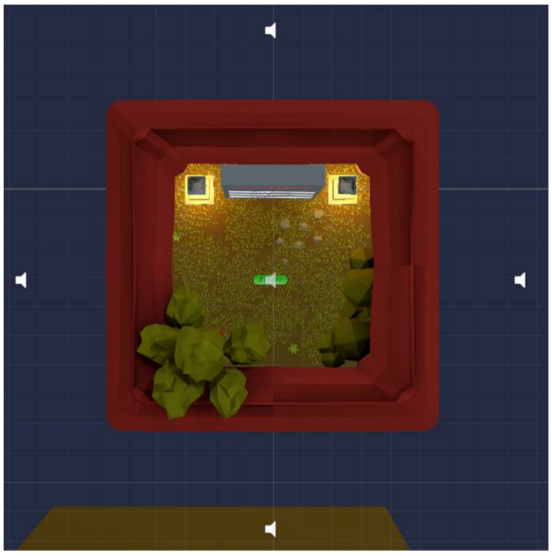

Figure 67. Four audio sources surrounding the player

The soundscapes themselves are composed of layered sounds of various wildlife that we have all heard on summer nights during our time spent in Japan. Surrounding the player are the sounds of crickets and the cawing of crows. The sound of *min-min-zemi*, a species of cicada whose sound is iconic of summertime in Japan, stand out particularly well in the soundscape.

Finally, our music, being non-diegetic, is played directly in the player's headphones with no directional audio. It was selected purely based on its style and instrumentation, which sound undoubtedly Japanese, in line with our visual style.

7.4 Audio Implementation in Game

The process of how audio was implemented into the game itself had changed over the course of its development. Initially, we found it simplest to use one audio source per object that makes sound. Then, the scripts operating within this object could call for any one shot audio clip to play from the audio source. We later discovered that this method had multiple limitations. When playing a one shot from an audio source, the sound uses the settings from the audio source but cannot be controlled in the same way as one playing with the Play function. It can neither change its pitch and volume nor play and stop playing programmatically. This is important, especially for sounds with effects like the pitching up and down of rocks of different sizes.

7.4.1 Audio Sources

Our solution to this going forward was to give each audio clip we had its own source prefab. In other words, if a spike has four sounds it needs to play, we give the object four audio sources that can each be controlled individually to affect the sound that they are playing. This proved to be more clean and modular. With each sound as its own prefab, the volume and pitch of this sound for the whole project could be adjusted in one place, while individual instances can still have total control over starting, stopping, pausing, pitch-shifting, and more in the code, which the one shot audio clips simply could not (Figure 68). This allowed us to master every sound together in the editor using the audio source options in each prefab.

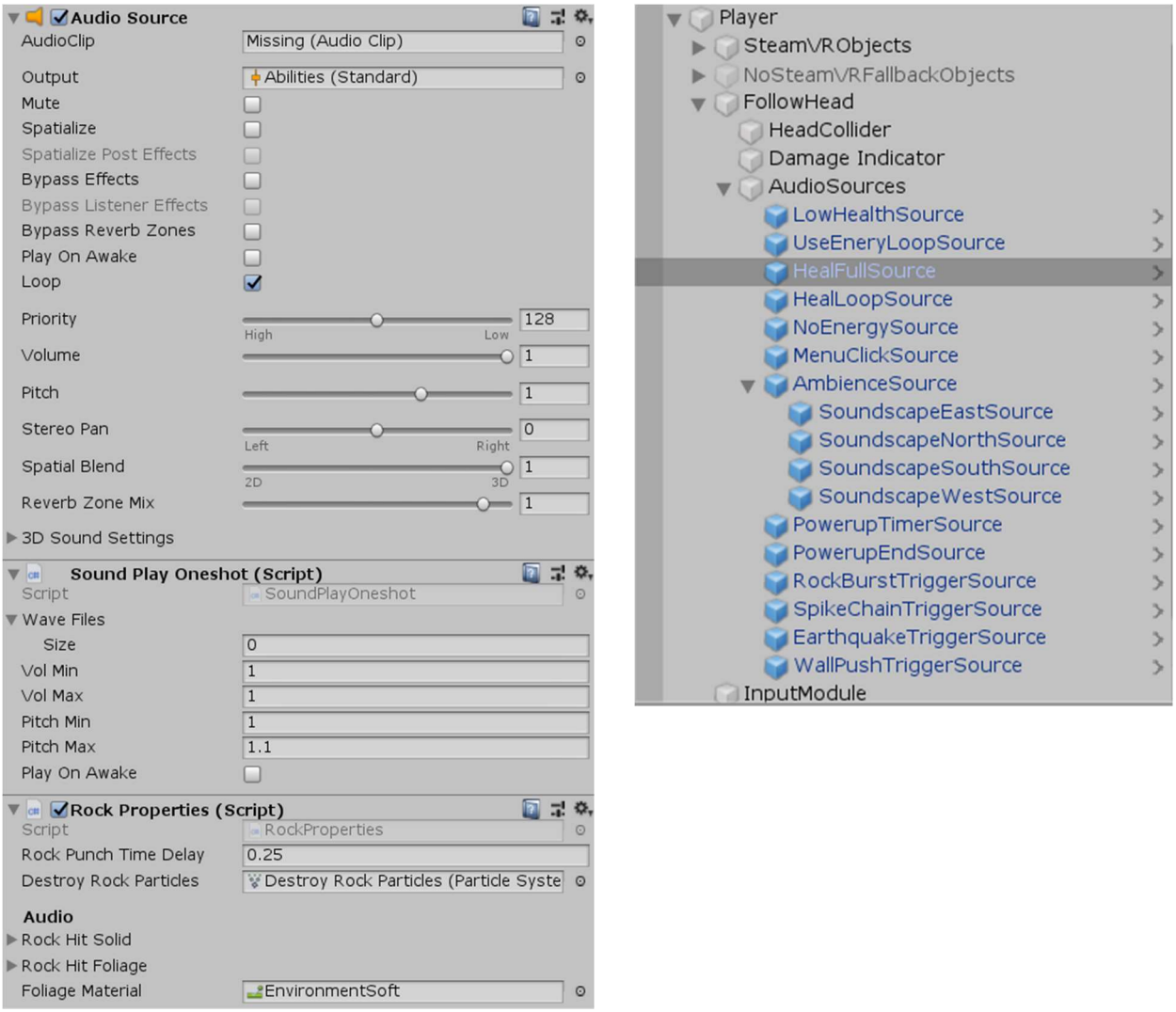

Figure 68. One audio source per object (left) vs. one audio source per sound (right)

7.4.2 Enemy Spawn Audio Cues

We knew ahead of time that we would need a sound effect of some sort to play when enemies spawn so the player can have their attention directed towards them without looking. How this should be done became a topic of discussion for the whole team. We knew we wanted the sound of drums to reverberate from the enemies' location, as this would give the impression of an intimidating army. However, enemies can spawn rapidly, slowly, continuously, and with big breaks in between, depending on what wave the player is fighting. This meant that our initial plan of playing one sound from the gates at the beginning of the wave would not be enough.

After recording two bars of drumming on *Ōdaiko*, we split it into smaller chunks to control the duration of the track. This created eight one-beat-long clips we have called

drumlings. With these tracks, we were able to create a script that plays a given number of drumlings in a random sequence (Figure 69). This way, when an enemy spawns, we can play a drum cue for a very specific duration that is created on-the-fly to sound different every time, without one cue running over or interrupting another.

Figure 69. Component that sets drumlings to use when spawning waves

8. Playtesting

Across our 12 weeks in Japan, we performed three separate playtesting sessions: pre-alpha, alpha, and beta. When performing playtesting, we had the opportunity to test both in our hosting lab as well as in a dedicated room for VR usage. Both places provided ample space for the player to move around safely, and the room we chose during each playtest was strictly based on whichever was unoccupied for our allotted time slot.

Our goal was to iteratively receive feedback from potential players of our game to determine what they were looking for and align those ideas with our own vision for the game. For each session, we took notes as the tester played the game, and we asked the tester to complete an anonymous survey following the session to gather data for later analysis. This section details the purpose of each playtesting session and the information we gleaned from them. Our IRB approved screening questions and surveys can be seen in Appendix C and D, respectively, while the list of changes from each session can be seen in Appendix E.

8.1 Pre-Alpha

Our testing for pre-alpha primarily focused on the abilities that the player would be using throughout the game to fight off enemies. These abilities were the basic rock, spike, quicksand, and wall abilities, not including power-ups as they had not been implemented at the time. With the motion controls being the mechanic that the player uses to interact with the world, it was important that they were made to be enjoyable and feel comfortable to perform. Therefore, one of our goals for this playtest was to get the player's feedback on how it felt to use the motion controls for each of the abilities. Another goal was to gauge how far away the enemies could be from the player to make them difficult to hit, but not impossible.

8.1.1 Testing Setup

We acquired five people to participate in this round of testing. This group consisted of students that attend Ritsumeikan University as well as members of another MQP team. Two of the participants had never used VR equipment before while the other three were proficient with it, providing our testing with a variety of the types of users that might play our game.

We created a scene in Unity specifically for this playtest that consisted of a large platform with four prototype enemies at varying distances that we used as targets. One enemy

was within punching distance, one was just out of punching distance, one was at the edge of the range in which players could create abilities, and the last was far past the edge for creating abilities. Before playing, we provided verbal instruction to the player along with a reference to the controller to teach them how to use the abilities before helping them put on the headset and take the controllers.

For each tester, we first let them experiment with the different abilities without any instruction to see what they would figure out on their own and how they would interpret the different abilities. We then instructed them to use any abilities that they had not used on their own after a few minutes had passed. Finally, we told the tester to throw rocks of varying sizes at each of the targets, starting with the closest target and ending with the farthest to see how difficult it was for players to hit enemies as they got farther away. The layout of the playtest map can be seen in figure 70.

Figure 70. Original map used for playtesting where player is placed next to Target one

8.1.2 Results and Conclusions

For most of the abilities, each of the testers said that the motion controls were comfortable and made sense for the action being performed. The major piece of feedback we received regarding the controls concerned the creation of walls. Testers consistently reported that there were too many inputs occurring simultaneously: both trackpads and triggers had to be held to create the wall while also performing the motion control. The solution was to toggle the wall drawing mode by clicking the trackpads as opposed to holding them, reducing the number of inputs at any given time.

Regarding hitting distant targets, players were satisfied with how it felt to throw the rocks. Some reported that it felt as if they were throwing an object in real life, while others who had used VR before said that the throwing functionality felt more natural than most other games they have played. This led each tester to have an easier time hitting each of the targets, with most of them only having difficulty on the final target. This showed us that enemies could be far from the player while still being able to eliminate them, and we used this information when calculating the movement paths for the ranged enemies.

Aside from information gathered about our playtesting goals, we were also able to acquire additional information about the abilities. One of the most common pieces of feedback that we received was that the spikes were too fat and short, causing them to feel as if they were not threatening. This led the player to use the creation of walls in ways that we intended the spike to be used for. Another important piece of feedback we got came from two different testers, both saying that they were hoping for some way to destroy walls manually as they could become obstructive.

8.2 Alpha Testing

At this point of development, most of the basic functionality was in the game: the abilities and power-ups were complete with the changes made from pre-alpha, the basic arena was set up with proper lighting and simple color on objects, and the 3 enemies could spawn and attack the player in their own ways. Since we had a basic gameplay loop, the focus of this playtesting session was to get a feel for what players thought about the combat system, both in regards to the effectiveness of the abilities and the quality of the enemies.

8.2.1 Testing Setup

We acquired eight people to participate in this round of testing. Five of these individuals participated in the first round of testing while three of the participants were new and had never used VR equipment before.

The scene we used for this session was the complete map until that point with the five different locations. Each location had a set of predetermined waves of enemies for the players to defeat before they could move on to the next area. Rather than verbally explaining how to use the controls as we had done during the first playtest, we drew basic images depicting the actions the

player needed to perform to use their abilities that can be seen in figure 70. The purpose of this was to prepare for the implementation of our tutorial by using a different method to see what the player responded more positively towards.

For each tester, we gave them a minute to play with their abilities before the waves began and they had to fight real enemies. After that, they had up to ten minutes to make it as far as they could through the map. If the tester had any questions about using the abilities, we would address them and help them figure it out, but otherwise left them on their own during the waves. Upon completing a location, we instructed them to click a button in the pause menu to teleport to the next location.

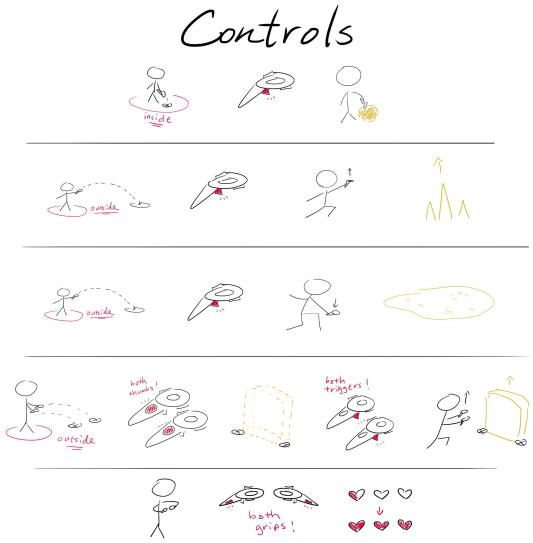

Figure 71. The images provided to the player prior to playing the game

8.2.2 Results and Conclusions

Even though we were testing for the combat system, the most valuable takeaways from this playtest were regarding the necessity of a tutorial and audio. The players struggled to remember the visuals we had provided them during combat, and it took them a while to fully grasp how the combat system worked which was something we didn't think to account for. Additionally, some of our testers spoke very little English and could not understand our instructions. This showed us that we needed a comprehensive tutorial for the user to learn their abilities through that utilized as little text as possible. These findings ultimately resulted in the tutorial implemented in the final version of our game.

Even once the player figured out how to use their abilities, they struggled to determine where enemies were coming from, how many enemies were around them, how full their energy was, and when they were low on health. This caused progressing through the game to be extremely difficult for most players, pushing us to put an immediate focus on the audio system in an attempt to avoid this issue for the next play session.

Similar to an audio cue for being low on health, players suggested that there be a visual indication for receiving damage as well. Unless the player was looking at their health bar on their

hand, it was impossible to tell when the player took damage. This meant that the testers were losing without ever knowing why or from where they were taking damage.

In terms of gameplay, one of the most consistent responses was that quicksand was not useful or unenjoyable to use. At the time, no abilities could be activated from inside a pile of quicksand, which players reported took away from any benefit that the quicksand might be able to provide. Another major concern by the testers was that the heavy enemy moved too quickly for the amount of damage they dealt to the player with every attack. If these enemies are to be as powerful as they are intended to be, the player needs to be given sufficient time to deal with them.

8.3 Beta Testing

This final round of testing included every feature that we had planned for the game in addition to a majority of the changes and bug fixes implemented since the previous playtesting sessions. We had three major goals for this round of playtesting to assure that our game was in a completed state. First, the tutorial had to be easy to understand and make it quick for the user to learn the mechanics of the game so there was no confusion while fighting enemies. Next, enemy spawning needed to be reduced enough to prevent enemies from overwhelming the player while still providing a challenge. Finally, the art and sound had to be cohesive and add to the game in a way that did not detract from the gameplay experience.

8.3.1 Testing Setup

For this final round of playtesting, we were able to acquire 20 testers, with the eight participants from alpha testing returning. This time, the playtest occurred across three different days to allow each of our participants to find a time that worked for them.

The map for this playtest was the same as the one from the previous playtest complete with all of the decorations, proper lighting, and mesh textures. The five locations remained the same from the previous test, but the waves were rebalanced to provide a better progression for the user. This was achieved by introducing each enemy type one at a time, increasing the number of enemies the player faces in each wave as the game progresses, and having a theme for each wave with respect to the frequency of each type of enemy that appears. Once the player was

situated, the game started with the tutorial. The team answered any questions the tester had but provided as little instruction as possible to see if the game was self-explanatory.

Each tester received thirty minutes to play the game. This time included making it through the tutorial and completing as many locations as possible. If the tester took too long on any given tutorial section, we instructed them to continue through the game so they could see as many locations as possible.

8.3.2 Results and Conclusions

The main takeaway that we had from this round of playtesting was that the tutorial was not intuitive for the players. People had difficulty understanding the level of importance for motion control, making abilities such as the spikes ineffective. Additionally, the tutorial videos were not always obvious about the actions the player needed to perform causing a great deal of confusion for using the abilities. A common piece of feedback we received was that the controller inputs were not stressed enough. While the in-game video did help the user, the testers reported that a more in-depth video showing how to use the controller would have greatly benefitted the learning experience.

Another common criticism we got was that there was not enough text during the tutorial. Our team created the tutorial with the intention of keeping the word count small to avoid any translation issues for our non-English speaking players. However, most players expressed that they would have liked more text to add further clarification the abilities. Our final point of criticism was that there was too much information given to the user at once. The testers agreed that splitting up the tutorial to not all occur at once would have helped them to remember the abilities better as they played.

In terms of the wave system and enemy spawning, our testers were happy that the tutorial only included the heavy enemies as their slow speed made them easy to deal with on their own. For the first time a light enemy spawned in location two, many testers stated that they were caught off guard and would have liked there to be an explicit way to indicate to the player that a new type of enemy was spawning. While most testers did not have time to progress further than location two, those that did reported that the difficulty increases too rapidly. While introducing the player to new enemy types, the difficulty of those waves should not increase by very much to give the player time to adapt to the new challenge.

The art and audio from our game received the highest amount of praise. Every tester stated that they enjoyed the atmosphere and that the art and audio worked together to make them feel as if they were really in Japan. For how it affected gameplay, most players said that the decorations did not inhibit their ability to play and added a perfect amount of difficulty. However, many people wished that there were more sounds and indications for when enemies were approaching them and from which direction. Additionally, the UI was easy for people to understand, but most people expressed that they would have liked to have seen their health and energy bars at all times.

9. Post Mortem

In this chapter, we reflect on the development of our game across 12 weeks in Japan. We primarily focus on what we did most successfully, what we could have improved on, and what we would like to add in the future.

9.1 What went right

Following the completion of the development of our game, our team was very happy with what we were able to produce. This was a game that we not only had fun playing but enjoyed creating as well. We were especially happy with the reactions of our testers. According to our testers, the world felt immersive and consistent, making the player "feel" like they were in Japan. A tester mentioned that although it was an American team, the game expressed the Japanese style well. Additionally, several players reported that they would have loved to play the game again if we had additional time allowing for it.

The team also believed that the game mechanics worked well together from a design perspective. Abilities felt powerful and rewarding, especially when the player obtains a power up. Each ability can be used in various ways allowing for creativity on how to approach each round. For example, rocks can be used as projectiles as well as shields. However, a rock shield does not allow the player to regenerate energy, so a different player could simply opt to move to evade projectiles or use cost-effective walls.

The different types of enemies also work well together. For example, heavy enemies are weak by themselves due to their slow movement. However, combining the heavy enemies with an enemy with projectiles significantly increases their effectiveness. This creates challenging gameplay that the player has to adapt to constantly. Moreover, the implementation of the wave system and queue of enemies makes our game feel fast-paced and exciting.

From a developing perspective, the team made it a priority to have a realistic and achievable scope for what we could accomplish across 12 weeks. Furthermore, the game's design and goals stayed consistent throughout development. For these reasons, the game's production went without any severe drawbacks or need to discard completed work. We also believe that the work we planned to complete was properly distributed amongst the members of our team. Everyone had tasks to complete throughout the entirety of development that suited the

strengths of that person. This led to quick and efficient development for most of the functionality included in the final game.

Finally, we excelled at our ability to work together and communicate with each other effectively. At the beginning of each day, we would all quickly discuss what it was we would work on throughout the day and periodically provide updates throughout and at the end of the day. If anyone needed assistance for difficult implementations, we went to each other and worked together to get past the problems. Following each playtesting session, our team got together to discuss the quality of the playtest, determine what needed to be fixed as a result of watching others play our game, and come up with a plan to get us to our next deadline.

9.2 What went wrong

While there were many places where our team excelled, there were two main areas our team identified where we could have improved: better prioritizations before playtesting sessions and better planning for functionality that was saved for later in the development time.

Prior to each round of playtesting, our team continued development until hours before the first tester was scheduled. This led us to have an insufficient amount of time for testing the game, and, in one case, resulted in game-breaking bugs being in the playtesting build that required hotfixes between testers. If we had the opportunity to change this, we would create our playtesting build of the game the day prior to playtesting. This would give us plenty of time to test for and fix any obvious bugs ahead of time, giving our players a better experience.

The other major area for our team that needed improvement was with the planning for functionality that was started later into development such as the tutorial and audio implementations. For both of these areas, we underestimated the amount of time and effort that would be required to complete and polish them. While the tutorial and audio were functionally complete, our testers' main complaints had to do with both the tutorial and audio needing additional work. Knowing what we do now, we would have started working on the implementation of these components sooner into development while saving other less important aspects for closer to the deadline.

9.3 Future Development

While we were able to fulfill all of our goals during our time in Japan, we still have many other ideas for improving the game that we would like to implement after our project term. Of

the existing features, the tutorial needs the most work, as it currently relies too heavily on the player watching rather than interacting. Our testers most often struggled with how they should move their arms and hands to make the ideal motions, as the videos could only show the hands from another person's point of view. An improved version would use animated objects that exist in the virtual world to guide the player's hands in the entire range of motion we want them to use. For example, the spike input is more effective when the player's arm swings in a large fast arc. We would create a translucent Vive controller, placed on a long translucent rail, animating from bottom to top and forming a translucent spike in front of it. This would quickly and nonverbally show the player not only how to move but how the attack should appear when executed correctly.

In addition to revamping the tutorial, we compiled our own hopes and tester suggestions to guide our future development.

While these features were never planned or in scope originally, they each could improve the game in their own way. The score system is a highly requested feature since testers could not identify a clear goal to the game other than eliminating enemies. Rewards for speed, amount of damage, types of attacks, etc. would provide a concrete goal and invite repeated improvement. A story mode or boss fight would also contribute to the game's storyline and progression. Both options would add gameplay time and make completing the game more rewarding.

The current power-up system, while working as intended, appears nebulous to testers and even to more experienced players. The power-up UI elements are not plainly visible and detract from gameplay since the player must look at their hands. Collectible power-ups that appear as rewards for defeating enemies would be simpler for players to understand and could even have additional functions, such as infinite energy or health. This would remove the need for a UI element, thus simplifying the power-up system.

Additionally, many testers had a hard time getting past the tutorial or first location, while others found the last levels only mildly challenging, so the option to select difficulty would greatly expand our target audience. To further appeal to all skill levels in our target audience, we would add an endless mode where the game scales in difficulty endlessly as they progress through waves until the player dies.

Even with varying experience, all testers agreed that melee abilities and more types of enemies would make the game more interesting. We particularly want to improve close range

combat as all of our current abilities are geared toward preventing enemies from getting close to the player, so the player is far less powerful against close enemies.

During early planning stages, we also conceptualized a randomly generated arena with varying terrain and obstacles each game. Obstacle randomization in the future would increase replay value, since the arena, play locations, and waves are currently fixed.

Finally, we hope to rig and animate the characters ourselves in the future. Auto rigging programs are usually not as precise as manual work, so our enemy movements and physical deformations are not ideal in their current state.

10. Conclusion

Our team members unanimously enjoyed working on *Shotoku's Defense*, as we have been successful overall in turning our simple vision into something our players have found entertaining. When we first began this project, no members of the team had worked with Unity, and some had not even worked on a game project before. Based on our progress, we believe that implementing all of the changes we proposed could bring this game to a professional level. We plan to build on what we have learned during the 12 weeks we spent on development to bring the consistency, polish, and additional features the game needs to create the best possible gameplay experience.

References

Belenko, D. (n.d.). *How to Use Complementary Colors in Photography*. Retrieved from https://expertphotography.com/complementary-colors-photography/

Bryant, A. J. (1991). *Early Samurai 200-1500 A.D*. Oxford: Osprey.

Green, A. (2018, February 19). *Sword Spotlight: The Chokutō*. Retrieved from https://www.martialartswords.com/blogs/articles/sword-spotlight-the-chokuto.

Google (2019). Shitennoji. Retrieved from https://goo.gl/maps/GXC2CymeS7R8xguEA

Hahn, M. (2019, February 22). *EQ 101: Everything Musicians Need to Know About EQ*. Retrieved from https://blog.landr.com/eq-basics-everything-musicians-need-know-eq/.

Kakidai. (2018, September). File:Chokuto Sword - Suiryu ken.jpg. Retrieved from https://upload.wikimedia.org/wikipedia/commons/4/49/Chokuto_Sword_-_Suiryu_ken.jpg

Laryushin, V. (2017, October 20). *Middle Ages Mine*. Retrieved from https://sketchfab.com/3d-models/middle-ages-mine-0c74a90dd8674835b2a13a855663d0b3

Mcbride A. (2011, Jun 1). *Historical Warrior Illustration Series Part Xll*. Retrieved from https://thelosttreasurechest.wordpress.com/2011/06/01/historical-warrior-illustration-series-part-xll/#jp-carousel-1637

Minecraft [Computer Software]. (2011). Mojang.

Miya (Photographer). (2005). *Hyoshigi01* [Digital Image]. Retrieved from https://commons.wikimedia.org/wiki/File:Hyoshigi01.jpg.

Movement in VR. (n.d.). Retrieved from https://unity3d.com/es/learn/tutorials/topics/virtual-reality/movement-vr.

Mueller, J.B (Photographer). (2005, October 12). Pondbridge [Digital Image]. Retrieved from https://www.flickr.com/photos/johnmueller/52617869

Nickelodeon. (2005). Avatar: The Last Airbender.

Pan, S. X., Gillies, M. (n.d.) *Challenges in VR Interaction and User Interfaces in VR* [Video file]. coursera. Retrieved from https://www.coursera.org/lecture/3d-interaction-design-virtual-reality/introduction-to-graphical-user-interfaces-in-vr-bPoWX.

Pontypants. (2018, August 18 Published). How to Make Low Poly Look Good. Retrieved from URL https://sundaysundae.co/how-to-make-low-poly-look-good/.

Pontypants. (2019, March 24). Low Poly Character Design. Retrieved from https://sundaysundae.co/how-to-make-low-poly-characters/.

Prasanth (Photographer). (2005, December 5). Odaiko - A Japanese Drum [Digital Image]. Retrieved from http://piccy.blogspot.com/2005/12/odaiko-japanese-drum.html.

Programming in Unity: For programmers new to Unity. (2019). Retrieved from https://unity3d.com/programming-in-unity.

Rad-Coders. (2017, September 16). Low Poly: Foliage. Retrieved from https://assetstore.unity.com/packages/3d/vegetation/low-poly-foliage-66638.

Reading, N. (2012, March 19). *Moustache……* [Digital Image]. Retrieved from https://www.flickr.com/photos/nathanreading/6851599794/.

Rich. (2015, August 7). The Unique Weapons of Ancient Japan. Retrieved from http://www.tofugu.com/japan/ancient-japanese-weapons/.

Ryukyu Kobudo Shimbukan. (n.d.). Tekko/Horseshoe. Retrieved from http://ryukyu-kobudo.com/tekko/

SnowFiend Studios. (2019, Jan 28). LowPoly Rocks. Retrieved from https://assetstore.unity.com/packages/3d/environments/lowpoly-rocks-137970

Soramimi (Photographer) (2014, March). Retrieved from https://commons.wikimedia.org/wiki/File:Kondo_and_Gojunoto_Tower_of_Shitennoji_Temple_2.JPG

Synty Studios. (n.d.). POLYGON - Samurai Pack. Retrieved from https://syntystore.com/products/polygon-samurai-pack.

Unity Technologies (n.d.). Navigation System in Unity. Retrieved from https://docs.unity3d.com/Manual/nav-NavigationSystem.html.

Umehara, T. (1980). 仏教の勝利. Tokyo, Japan: Shogakkan, 291-292.

ustwo Games. (2015, December 22). Designing Sound for Virtual Reality. Retrieved from https://medium.com/@ustwogames/designing-sound-for-virtual-reality-a37a40e80463.

VIVE Virtual Reality System. (n.d.). Retrieved from https://www.vive.com/us/product/vive-virtual-reality-system/.

Appendix A: Asset lists

Art Category	Type	Item	(Expected) Completion Date
3D	Earth	Quicksand	Complete 8/19
		Rocks	
		Spikes	
		Wall	
	Enemies	Light Mononobe	Complete 9/5
		Medium Mononobe	Complete 9/3
		Heavy Mononobe	Complete 9/5
	Weapons	Bow and Arrow (Light, Medium)	Complete 9/20
		Sword (Medium)	Complete 9/5
		Tekko: Brass knuckle like weapon (Heavy)	Complete 9/5
	Arena	Temple 1 (North, tall)	Complete 9/23
		Temple 2 (South, wide)	Complete 9/23
		Entrances x4	Complete 9/24
		Tileable straight arena walls	Complete 9/23
		Tileable corner arena walls	Complete 9/23
		Area Base with pond cutout	Complete 8/20
		Gazebo (pond)	Complete 9/6
		Bell Tower	Complete 9/24
		Trees	Complete 9/24
		Foliage	Complete 9/22
		Boardwalk	Complete 9/27
		Toro	Complete 9/6
		Lanterns	Complete 10/5
		Arena Graybox	Complete 8/7
		Mountain and tree parallax	Completed 9/24
		Skybox	Completed 9/26

		Final Arena	Complete 10/1
		Lighting	Complete 10/9
2D	Power-up icons	Earthquake	Complete 8/27
		Multiple rocks	Complete 8/27
		Spike chain	Complete 8/27
		Wall push	Complete 8/27
	Particles	Sand	Complete 9/5
		Fire	Complete 9/5
		Hole	Complete 9/5
	Misc.	Skybox top with stars	Complete 9/26
Animation	Heavy	T pose	Complete 9/5
		run forward	Complete 9/5
		walk forward	Complete 9/5
		idle	Complete 9/5
		punch 1	Complete 9/5
		punch 2	Complete 9/5
		getup stomach	Complete 9/5
		getup back	Complete 9/5
	Medium	T pose	Complete 9/5
		Run forward	Complete 9/5
		strafe forward	Complete 9/5
		strafe back	Complete 9/5
		strafe right	Complete 9/5
		strafe left	Complete 9/5
		idle	Complete 9/5
		sword swing 1	Complete 9/5
		sword swing 2	Complete 9/5
		sheath	Complete 9/5
		unsheath	Complete 9/5

		bow1	Complete 9/5
		bow2	Complete 9/5
		climb	Complete 9/5
		jump in place	Complete 9/5
		getup stomach	Complete 9/5
		getup back	Complete 9/5
	Light	T pose	Complete 9/6
		run forward	Complete 9/6
		strafe forward	Complete 9/6
		strafe back	Complete 9/6
		strafe right	Complete 9/6
		strafe left	Complete 9/6
		idle	Complete 9/6
		bow draw & release	Complete 9/6
		getup stomach	Complete 9/6
		getup back	Complete 9/6

Tech Category	Type	Item	(Expected) Completion Date
Player	Rock throw	Spawn rock	Complete 8/23
		Throwable physics object	Complete 7/26
	Ground cursor	Find world position with hand tracking cursor	Complete 7/30
		Arc UI	Complete 7/30
	Spike	Spawn spike with velocity based on player's hand	Complete 8/23
	Quicksand	Motion controls (swipe down)	Complete 7/31
		Slow enemies	Complete 8/26
	Wall	Motion controls (pick location, drag out, drag up)	Complete 8/23
		Launch enemies	Complete 8/7
	Heal	Motion Controls	Complete 7/28

	Health and Energy Bars		Complete 7/27
	Player Ability Area	Show a ring around the player to show where you can use rocks vs everything else	Complete 8/25
Pathfinding	NavMesh generation	Generate with static props	Complete 7/29
		Generate dynamically as player creates walls, spikes, quicksand	Complete 8/14
	Climbing / Jumping	Generate nav links at runtime	Complete 8/20
		Basic Functionality Done / light enemies are able to climb walls	Complete 8/26
		NavMesh link on corners of wall	Complete 9/6
Enemy Heavy	Animation	Walking	Complete 8/13
		Slashing	Complete 9/5
		Getup	Complete 10/3
		Controller	Complete 8/14
	Attacking	Attack range	Complete 8/14
		FSM	Complete 8/29
	Ragdoll	Ragdoll	Complete 8/9
Medium Enemy	Animation	Walking	Complete 9/2
		Shooting	Complete 9/5
		Melee	Complete 9/2
		Climbing	Complete 9/26
		Getup	Complete 10/3
		Controller	Complete 9/2
	Attacking	Melee or ranged	Complete 9/1
		Visibility check	Complete 9/2
		FSM	Complete 9/3
	Ragdoll	Ragdoll	Complete 9/5
Light Enemy	Animation	Walking	Complete 9/16
		Shooting	Complete 9/16
		Getup	Complete 10/3

		Controller	Complete 9/16
		FSM	Complete 9/6
		Shooting Ability	Complete 9/6
	Ragdoll	Ragdoll	Complete 9/6
Powerups	Energy	Unlimited Energy	Complete 8/19
	Abilities	Chain Spike	Complete 9/20
		Burst Rock	Complete 8/20
		Wall Push	Complete 8/23
		Earthquake	Complete 8/26
Enemy waves	Pre-made	Location 1	Complete 9/25
		Location 2	Complete 9/25
		Location 3	Complete 9/25
		Location 4	Complete 9/25
		Location 5	Complete 9/25
	Enemy spawners	Spawner prefab	Complete 7/26
UI	Laser Pointer	Laser Pointer Interaction Script?	Complete 8/13
	Healthbars	Tracking enemy	Complete 8/12
		Rotate to player	Complete 9/17
	Pause Menu	Laser Pointer	Complete 8/15
		Pause game logic	
		Instantiation of menu	
		Button logic	
		Game over + pause menu logic	Complete 8/29
	Tutorial	Videos the player can watch to learn how to play the game	Complete 9/25
	Game Over Screen	Script Logic	Complete 8/22
		Testing	Complete 8/29
Particles	Abilities	Rock creation (swirl)	Complete 9/3
		Rock creation (pull sand)	Complete 9/3
		Rock destruction	Complete 9/3

		Spike creation (rocks)	Complete 9/1
		Spike creation (moving earth)	Complete 9/3
		Spike destruction	Complete 9/2
		Quicksand creation	Complete 9/1
		Quicksand destruction	Complete 9/2
		Wall creation (moving earth)	Complete 9/1
		Wall destruction	Complete 9/2
	Enemies	Death particles	Complete 9/5
	Ambient	Flames	Complete 9/5
		Fireflies	Complete 9/22

Category	Type	Item	Components	Completion Date
Enemy	Light	Bow	Strong Draw	Complete 10/9
			Strong Release	Complete 10/9
		Hurt	Small	Complete 10/9
			Medium	Complete 10/9
			Large	Complete 10/9
		Death	Death	Complete 10/9
	Medium	Bow	Weak Draw	Complete 10/9
			Weak Release	Complete 10/9
		Footstep	Ground (x5)	Complete 10/9
			Quicksand (x4)	Complete 10/9
		Hurt	Small	Complete 10/9
			Medium	Complete 10/9
			Large	Complete 10/9
		Sword	Hit (x2)	Complete 10/9
		Death	Death (x2)	Complete 10/9
	Heavy	Footstep	Ground (x5)	Complete 10/9
			Quicksand (x4)	Complete 10/9
		Hurt	Small	Complete 10/9
			Medium	Complete 10/9
			Large	Complete 10/9
		Tekko	Hit (x2)	Complete 10/9
		Death	Death	Complete 10/9
	Arrow	Fly	Loop	Complete 10/3

		Hit	Player	Complete 10/3
			Solid (x3)	Complete 10/3
			Foliage (x3)	Complete 10/3
Environment	Wave Cues	-	Taiko Beat (x8)	Complete 9/24
	Ambient	Soundscape	North	Complete 9/27
			West	Complete 9/27
			South	Complete 9/27
			East	Complete 9/27
			Ears	Complete 9/23
		Tourou	Loop	Complete 10/2
Player	Abilities	Heal	Loop	Complete 10/2
			Full	Complete 10/2
		Effectiveness	Nice	Complete 10/9
			Excellent	Complete 10/9
		Rock	Emerge (x3)	Complete 10/9
			Grow Start	Complete 10/9
			Grow Loop	Complete 10/9
			Grow End	Complete 10/9
			Throw Small	Complete 10/9
			Throw Medium	Complete 10/9
			Throw Large	Complete 10/9
			Hit Solid (x5)	Complete 10/9
			Hit Foliage (x3)	Complete 10/9
		Spike	Select Start	Complete 10/9
			Rise Start	Complete 10/9
			Rise End	Complete 10/9
		Quicksand	Rise	Complete 10/9
		Wall	Toggle On	Complete 10/9
			Toggle Off	Complete 10/9
			Drag Loop	Complete 10/9
	Health and Energy	Health	Low	Complete 9/26
		Energy	Usage Loop (Pitches up)	Complete 9/28
			No Energy	Complete 9/28
	Powerups	Rock Burst	Trigger	Complete 9/28
			Rock Split	Complete 10/9
		Chain Spike	Trigger	Complete 9/28
		Wall Push	Trigger	Complete 9/28
			Release	Complete 10/9
		Earthquake	Trigger	Complete 9/28

			Quake	Complete 10/9
		Timer	Loop	Complete 9/25
			End	Complete 9/27
UI	Menu / tutorial	Menu	Show	Complete 9/22
			Hide	Complete 9/22
			Click	Complete 9/22
			Misclick	Complete 9/22
		-	Start tutorial wave	Complete 9/22
Music	Background	Instrumental	-	Complete 9/24

Appendix B: Wave Spawns

Location #1

Time	Location	Light	Medium	Heavy
0	South	0	0	1
0	South	0	0	1
0	South	0	0	1
0	South	0	0	1

Location #2

Time	Location	Light	Medium	Heavy
0	West	1	0	0
30	South	2	0	0
50	West	1	0	1
60	South	1	0	0

Location #3

Time	Location	Light	Medium	Heavy
0	East	1	0	0
15	West	0	1	1
30	South	2	0	1
45	West	0	1	0
46	East	1	1	1

Location #4

Time	Location	Light	Medium	Heavy
0	North	2	0	0
15	West	0	2	0
20	West	0	0	1
25	East	1	1	1
35	North	2	0	1
70	North	0	2	2
80	East	2	0	0
86	West	1	2	2

Location #5

Time	Location	Light	Medium	Heavy
0	North	2	0	1
10	East	1	2	1
25	West	2	0	0
30	North	0	2	0
55	North	0	0	2
60	East	1	0	0
65	West	0	1	0
68	West	0	0	1

Appendix C: Playtesting Screening Questions

Wellness

- Are you prone to getting motion sick?
- Do you feel dizzy, lightheaded, or nauseous?
- Are you feeling excessively tired?
- Are you feeling sick?
- Have you had excessive coffee or energy drinks today?
- Are you prone to having seizures?
- Have you had migraines, headaches or earaches recently?
- Do you have a history of low blood pressure or fainting?
- Do you have a history of vertigo?
- Are your shoes comfortable?

Do not use the headset when you are: tired, under stress, suffering from cold, flu, headaches, migraines, or earaches, as this can increase your susceptibility to adverse symptoms.

Virtual Reality

- Have you used Virtual Reality (VR) equipment before?
- Are you comfortable with VR equipment?

Appendix D: Playtesting Surveys

Pre-Alpha

1. How natural did it feel to throw the rocks?

	1	2	3	4	
It was nothing like throwing a real object	◯	◯	◯	◯	It felt like I was really throwing an object

2. How hard was it to hit target 2?

	1	2	3	4	
Extremely Difficult	◯	◯	◯	◯	Extremely Easy

3. How hard was it to hit target 3?

	1	2	3	4	
Extremely Difficult	◯	◯	◯	◯	Extremely Easy

4. How hard was it to hit target 4?

	1	2	3	4	
Extremely Difficult	◯	◯	◯	◯	Extremely Easy

5. How easy was it to use the motion controls to regrow the rocks in your hand?

	1	2	3	4	
Extremely Difficult	◯	◯	◯	◯	Extremely Easy

6. Did the controls for regrowing the rocks make sense?

◯ Yes
◯ No

7. If you have any additional feedback or disliked any of the functionality about the rocks, please share below.

8. Please check all that you agree with:

	Agree	Disagree
The spikes are too tall	◯	◯
The spikes are too short	◯	◯
The spikes are too wide	◯	◯
The spikes are too skinny	◯	◯

9. How easy was it to use the motion controls to create spikes?

	1	2	3	4	
Extremely Difficult	◯	◯	◯	◯	Extremely Easy

10. Did the motion controls for creating spikes make sense?

◯ Yes

◯ No

11. If you have any additional feedback or disliked any of the functionality about the spikes, please share below.

12. How easy was it to use the motion controls to create quicksand?

	1	2	3	4	
Extremely Difficult	◯	◯	◯	◯	Extremely Easy

13. Did the motion controls for creating quicksand make sense?

◯ Yes

◯ No

14. If you have any additional feedback or disliked any of the functionality about the quicksand, please share below.

15. How easy was it to draw the base of the wall by pressing both track pads?

	1	2	3	4	
Extremely Difficult	◯	◯	◯	◯	Extremely Easy

16. Did the controls for drawing the base of the wall make sense?

◯ Yes
◯ No

17. How easy was it to use the motion controls to create walls by pressing and releasing the triggers?

	1	2	3	4	
Extremely Difficult	◯	◯	◯	◯	Extremely Easy

18. Did the motion controls for creating walls make sense?

◯ Yes
◯ No

19. If you have any additional feedback or disliked any of the functionality about the walls, please share below.

20. Please rate the abilities by how much you enjoyed using them

	Strongly Disliked	Disliked	Liked	Strongly Liked
Rocks	◯	◯	◯	◯
Spikes	◯	◯	◯	◯
Quicksand	◯	◯	◯	◯
Walls	◯	◯	◯	◯

21. Please rate the abilities by how frequently you think you would use them in a combat situation

	Very infrequently	Infrequently	Frequently	Very frequently
Rocks	◯	◯	◯	◯
Spikes	◯	◯	◯	◯
Quicksand	◯	◯	◯	◯
Walls	◯	◯	◯	◯

22. If there is an ability you wish you had while playing the game, please describe what this ability might look like.

23. If you have any additional comments, please share below.

Alpha

1. **Which ability was your favorite to use?**
   - ◯ Rocks
   - ◯ Spikes
   - ◯ Walls
   - ◯ Quicksand

2. **Which ability was your least favorite to use?**
   - ◯ Rocks
   - ◯ Spikes
   - ◯ Walls
   - ◯ Quicksand

3 **How often did you use the following abilities?**

	Very infrequently	Infrequently	Frequently	Very Frequently
Rock of default size	◯	◯	◯	◯
Resized rock, not maximum size	◯	◯	◯	◯
Maximum sized rock	◯	◯	◯	◯
Single spike	◯	◯	◯	◯
Multiple spikes	◯	◯	◯	◯
Small quicksand	◯	◯	◯	◯
Large quicksand	◯	◯	◯	◯
Narrow wall	◯	◯	◯	◯
Wide wall	◯	◯	◯	◯

4. **How fast do you feel the energy meter regenerated?**

	1	2	3	4	5	
Way too Slow	◯	◯	◯	◯	◯	Way too Fast

5. **How long did it take for the energy meter to start regenerating energy?**

	1	2	3	4	5	
Way too Long	◯	◯	◯	◯	◯	Way too Short

6. **How much energy did it cost to use each of the following abilities?**

	Way too much	Slightly too much	Just enough	Slightly too little	Way too little	I didn't use this ability
Rocks	◯	◯	◯	◯	◯	◯
Spikes	◯	◯	◯	◯	◯	◯
Walls	◯	◯	◯	◯	◯	◯
Quicksand	◯	◯	◯	◯	◯	◯
Healing	◯	◯	◯	◯	◯	◯

7. **How much damage did each of the following abilities do?**

	Way too much	Slightly too much	Just enough	Slightly too little	Way too little	I didn't use this ability
Default sized rocks	◯	◯	◯	◯	◯	◯
Resized rock, not maximum size	◯	◯	◯	◯	◯	◯
Maximum sized rock	◯	◯	◯	◯	◯	◯
Single spike	◯	◯	◯	◯	◯	◯
Multiple spikes	◯	◯	◯	◯	◯	◯

8 **How effective were the following abilities at slowing or stopping enemies?**

	Way too effective	Slightly too effective	Perfectly effective	Slightly not effective	Not effective	I didn't use this ability
Small quicksand	◯	◯	◯	◯	◯	◯
Large quicksand	◯	◯	◯	◯	◯	◯
Narrow wall	◯	◯	◯	◯	◯	◯
Wide wall	◯	◯	◯	◯	◯	◯

9. **How effective were the following power-ups in comparison to their original ability**

	Extremely more effective	Slightly more effective	Just as effective	Slightly less effective	Extremely less effective
Cluster of rocks	◯	◯	◯	◯	◯
Chain of spikes	◯	◯	◯	◯	◯
Moving rock wall	◯	◯	◯	◯	◯
Quicksand earthquake	◯	◯	◯	◯	◯

10. **How powerful were the following power-ups?**

	Way too powerful	Slightly over-powerful	Perfectly powerful	Slightly under-powered	Extremely under-powered
Cluster of rocks	◯	◯	◯	◯	◯
Chain of spikes	◯	◯	◯	◯	◯
Moving rock wall	◯	◯	◯	◯	◯
Quicksand earthquake	◯	◯	◯	◯	◯

11. **If there is an ability you wish you had while playing the game, please describe what this ability might look like.**

12 **If you have any additional comments on the abilities or power-ups, please share below.**

13. **How much damage did each of the following enemies take from rocks?**

	Way too much damage	Slightly too much damage	Perfect amount of damage	Slightly too little damage	Way too little damage
Light enemies (Archers)	◯	◯	◯	◯	◯
Medium enemies (Short Swordsman)	◯	◯	◯	◯	◯
Heavy enemies (Broad Swordsman)	◯	◯	◯	◯	◯

14. **How much damage did each of the following enemies take from spikes?**

	Way too much damage	Slightly too much damage	Perfect amount of damage	Slightly too little damage	Way too little damage
Light enemies (Archers)	◯	◯	◯	◯	◯
Medium enemies (Short Swordsman)	◯	◯	◯	◯	◯
Heavy enemies (Broad Swordsman)	◯	◯	◯	◯	◯

15. **How many enemies did you feel were around you?**

	1	2	3	4	5	
Way too many enemies	◯	◯	◯	◯	◯	Way too little enemies

16. **How difficult was it to see the incoming arrows shot by the light and medium enemies?**

	1	2	3	4	
Extremely difficult	◯	◯	◯	◯	Extremely easy

17 **How difficult was it to dodge or deflect the incoming arrows shot by the light and medium enemies?**

	1	2	3	4	
Extremely difficult	◯	◯	◯	◯	Extremely easy

18. **How difficult was it to attack light enemies at a distance?**

	1	2	3	4	
Extremely Difficult	◯	◯	◯	◯	Extremely Easy

19. **How quickly did the medium enemies approach you?**

	1	2	3	4	5	
Way too quickly	◯	◯	◯	◯	◯	Way too slowly

20. **How quickly did the heavy enemies approach you?**

	1	2	3	4	5	
Way too quickly	◯	◯	◯	◯	◯	Way too slowly

21. **If you have any additional comments on the enemies, please share below.**

Beta

1. **Overall, how helpful was the tutorial in learning how to use the abilities?**
   - ◯ Not helpful at all
   - ◯ Only a little helpful Moderately
   - ◯ helpful
   - ◯ Extremely helpful

2. **Which ability was the easiest to learn how to use?**
   - ◯ Rocks
   - ◯ Spikes
   - ◯ Walls
   - ◯ Quicksand

3 **What made this ability the easiest to learn?**

4. **Which ability was the most difficult to learn how to use?**
   - ◯ Rocks
   - ◯ Spikes
   - ◯ Walls
   - ◯ Quicksand

5. **What made this ability the most difficult to learn?**

6. **How helpful were the practice waves in learning how to use the abilities?**
   - ◯ Not helpful at all
   - ◯ Only a little helpful Moderately
   - ◯ helpful
   - ◯ Extremely helpful

7. **If there is anything else you would like to say about the tutorial, please share below.**

8. **How difficult was it to complete each of the following groupings of enemies?**

	Way too difficult	Slightly too difficult	Perfectly difficult	Slightly too easy	Way too easy	I didn't make it to this group of enemies
Enemy group after rock tutorial	◯	◯	◯	◯	◯	◯
Enemy group after spike tutorial	◯	◯	◯	◯	◯	◯
Enemy group after quicksand tutorial	◯	◯	◯	◯	◯	◯
Enemy group after wall tutorial wave	◯	◯	◯	◯	◯	◯
Enemy groups following the tutorials in front of the gate	◯	◯	◯	◯	◯	◯
Enemy groups between the wall and the building	◯	◯	◯	◯	◯	◯
Enemy groups at the middle of the arena	◯	◯	◯	◯	◯	◯
Enemy groups next to the pond	◯	◯	◯	◯	◯	◯
Enemy groups in front of the large gate	◯	◯	◯	◯	◯	◯

9. **If you answered "Way too difficult" to any of the above groupings, what made it difficult?**

10 **How many enemies did you feel there were in each of the following groupings of enemies?**

	Way too many enemies	Slightly too many enemies	A perfect number of enemies	Slightly too little enemies	Way too little enemies	I didn't make it to this group of enemies
Enemy groups following the tutorials in front of the gate	◯	◯	◯	◯	◯	◯
Enemy groups between the wall and the building	◯	◯	◯	◯	◯	◯
Enemy groups at the middle of the arena	◯	◯	◯	◯	◯	◯
Enemy groups next to the pond	◯	◯	◯	◯	◯	◯
Enemy groups in front of the large gate	◯	◯	◯	◯	◯	◯

11. **Overall, how do you feel about the increase in difficulty between each grouping of enemies?**

- ◯ The difficulty increased way too quickly
- ◯ The difficulty increased slightly too quickly
- ◯ The difficulty increased at the perfect rate
- ◯ The difficulty increased slightly too slowly
- ◯ The difficulty increased way too slowly

12. **If there is anything else you would like to say about the difficulty of the enemy groupings, please share below.**

13. **Was the design of the world enjoyable?**

- ◯ Yes
- ◯ No

14 **Was the design of the world consistent?**

- ◯ Yes
- ◯ No

15. **Was the setting of each location you were in diverse?**

- ◯ Yes
- ◯ No

16. **If no, which locations were not diverse and what made them not diverse.**

17. **How did you feel about the brightness of the game?**
    - ◯ Way too dark
    - ◯ Slightly too dark
    - ◯ Perfectly dark
    - ◯ Slightly too light
    - ◯ Way too light

18. **How do you feel about the amount of decorations in the level? (bushes, trees, lights, etc.)**
    - ◯ Way too many decorations
    - ◯ Slightly too many decorations
    - ◯ Perfect amount of decoration
    - ◯ Slightly too little decoration
    - ◯ Way too little decoration

19. **How intrusive were the decorations in the level to the game play? (bushes, trees, lights, etc.)**
    - ◯ Way too intrusive: very negatively impact game play
    - ◯ Slightly too intrusive: negatively impact game play
    - ◯ Not intrusive: impacted game play perfectly
    - ◯ Slightly not intrusive enough: had very little impact on game play
    - ◯ Way too not instrusive: had absolutely no impact on game play

20 **Did the audio positively affect your game play experience?**
    - ◯ Yes
    - ◯ No

21. **Did the audio feel consistent throughout the game?**
    - ◯ Yes
    - ◯ No

22. **Were you able to tell where enemies were coming from based on the audio?**
    - ◯ Yes
    - ◯ No

23. **Were you able to tell when you were low on health without looking at your health bar?**
    - ◯ Yes
    - ◯ No

24. **Were you able to tell when you were low on energy without looking at your energy bar?**

- ◯ Yes
- ◯ No

25. **Was the User Interface easy to understand?**

- ◯ Yes
- ◯ No

26. **If no, what made the User Interface not easy to understand?**

27. **Were there any performance issues while you were playing the game?**

- ◯ Yes
- ◯ No

28 **If you have any additional comments about the art style and design of the game, please share below.**

29. **How enjoyable was your game play experience?**

Mark only one oval.

- ◯ Not enjoyable at all
- ◯ Slightly enjoyable
- ◯ Moderately enjoyable
- ◯ Extremely enjoyable

30. **If there are any changes you would like to see to the currently existing game, please share below.**

31. **If there are any additional features you would like to see in the future, please share below.**

32. **The current name of the game is "Shotoku's Defense". If you have any ideas for a better name, please share below.**

33 **If you have any final comments about the game, please share them below.**

Appendix E: Playtesting Changes Priority Log

Priority scale:
1: least impact on mechanics/playability (cosmetic things, unnecessary extra features, etc.) / might not be in the final game
2: Might not be done after beta playtesting but it might be in the final game
3: Done before beta playtesting
4: Do as soon as possible.
5: Do now. high impact on mechanics/playability (game changing mechanics, game breaking bugs, etc.)

Pre-Alpha Change/Feature	Reason	Priority (1-5)	Complete?
Spike chain should increase the number of chains by size		5	X
Spike chain should continue infinitely until it hits something		5	X
Energy usage has a bug on wall create	• Rock didn't fly away when it had a wall come up from below • Certain walls did not disappeared	5	X
Don't allow wall drawing while using another ability	• This just doesn't make sense	5	X
Change Wall controls	• Two trackpads at the same time to toggle draw mode • Two triggers to go up • No abilities allowed in draw mode • While in draw mode, you can exit with two trackpad press or with grip button	5	X
Be able to destroy walls		5	X
Wall should break when it collides with the player area collider		4	X
Parrying / Blocking attacks with melee		4	X
Overlapping outlines should mean no performable action is valid		4	X
Make walls have a constant max height		4	X
Increase and randomize height of spike		4	X

Width of quicksand should increase	● It's useless at the size it currently is	3	X
Walls should slide through quicksand		3	X
Make player head collider smaller		3	X
Grab rock from far away and apply hover		3	X
Enemies explode into polygons on death		3	X
Decrease wall movement speed		3	X
Add rumble to rock resizing / throwing		3	X
Keep track of spike velocity across last 5 frames	● Spikes don't always release how/when you want them too	3.5	X
Zero the velocity of rocks when dropping if not in hand yet	● When spawning rocks, didn't hold so rock just flew away ○ Time ease in and check	2.5	X
Visual feedback for overlapping rocks with hand	● Some people thought they grabbed the rock when they didn't ● When picking up and when resizing	1	
Fall Damage		1	
Cluster rocks vs rocks collision		1	

Alpha Changes	**Description**	**Priority**	**Complete?**
Tutorial		5	X
Restarting function should delete all objects		5	X
Game Over Lock bug	Cancel abilities and change pointers	5	X
Audio		5	X
Taking damage should have a visual cue	Edges of view flashes red	4	X
Increase wall movements threshold	Walls are getting created by accident (when players do not pull up and just trigger)	4	X
Increase spike and quicksand movement threshold	Prevents players from accidentally creating spikes or quicksand Minimum size quicksand is too small	4	X

	thanks to a bug		
Energy and Health bar color for player	It's white right now, and hard to visualize	4	X
Enemy Queue	Limit amount of enemies at once	4	X
Enemies need to stand on the ground	Enemies currently float	4	X
Enemies get stuck on buildings' rooftops and trees	- Make building frictionless - Rooftops should not be part of the NavMesh - When wake up of ragdoll, check if near NavMesh - Check that agent is near agent before leaving ragdoll state	4	X
Enemies disappear sometimes when they are hit		4	X
Climbing glitches out	Disable climbing in retreat state Not sure if this will fix it	4	X
Climbing Animations	Look passable enough! Needs little tweaking	4	X
Check arrow hitbox so it hits walls and trees and spikes		4	X
Chain spike does no damage	Chain Spike currently move enemies instead of hitting them	4	X
There is a red circle inside the player area		3	X
Replenish energy while drawing wall		3	X
Quicksand should not impede wall and spike creation		3	X
Make arrow glow / have a trail	Arrow should be more obvious	3	X
Increase default rock damage (increase default rock radius)	Default rocks do too little damage. Max rocks felt like they did too little damage too but can probably fix w tutorial	3	X
Don't allow ability usage if the arc area is invalid		3	X
Add Arrow mesh and check rotations		3	X
Turn up ragdoll sensitivity	If enemies are punched, ragdoll all the time	2	X
Fix highlight shader		1	X
Show player health visuals	Green and red drain animation	2	
Quicksand earthquake trips those in full		2	

radius of quake			
Power-up ready visual cue	Can't tell when power-ups are going to happen in place of regular ability	2	
Headband looks funny		2	
Add Fall Damage		2	
Wider temple base	Enemies don't fit well	1	
Walking through quicksand makes enemies sweat and animate faster but walk slower	Not obvious that quicksand slows enemies	1	
Red highlight for rocks that can't be picked up		1	
Merge the ability outline rings when close		1	
Medium enemy didn't aim with the sword (if people squat they don't get hit)		1	
Make particles explode more when enemies die		1	
Make a minimum size for walls		1	
Enemies take damage from stationary wall	remove this	1	
Directional thumb button to activate power-ups	Currently power-ups just happen when they are charged	1	
Climbing bottom of temple	So enemies don't have to walk all the way around	1	

Beta Changes	**Description**	**Priority**	**Complete?**
Master *all* sounds	Quieter bug sounds	5	X
Wave testing	Design what waves are we going to have	5	X
Rocks are considered part of the player and causes them to take damage when deflecting rocks		5	X
Remove the trees by location #2 and location #1	Obstructs the player's ability to eliminate enemies	5	X
NavMesh Testing	Go through locations and check behavior	5	X

Fix Pause Menu Buttons	Set better height for spawning	5	X
Enemies shouldn't double hit for damage		5	X
Enemies de-spawn when the player dies in tutorial		5	X
Enemies get stuck in the east gate	needs more playtesting	5	X
Disable NavMesh on roofs, trees, bushes		5	X
Caption text in tutorial slides		5	X
Fix arrow and shooter rotation		5	X
Water shader		4	X
Texture heavy enemy		4	X
Teleport star needs to be closer to player		4	X
Rock damage needs to be fixed	sometimes, hit won't damage at all	4	X
Put Quit Game in death menu		4	X
Medium guys don't take damage with rocks very often		4	X
Make the ability usage ring red when interacting with the player radius		4	X
Invisible trees and bushes	LOD materials are wrong	4	X
Grass culling		4	X
Get up animation	Climbing, shooting animations	4	X
Don't let rocks on the ground prevent doing abilities		4	X
Disable the trackpad during the tutorial before walls are introduced		4	X
Wind zone on rock particles needs to be turned off		3	X
Spike particles not showing up on destroy		3	X
Spawn areas need to be smaller (closer to gates)	Enemies take a while to leave the spawn area, long pause after drums	3	X
Reset temporary energy on death		3	X
Make tutorial buttons be world objects and not UI buttons	Players want to make abilities during tutorial slide	3	X

Make light enemy not walk through player	enemy is encouraged not to walk through player	3	X
Let the player know when they win and restart the game from the spawn room		3	X
In power-up tutorial slide, turn on power-up and show image of power-up from hand	End when player goes to next slide	3	X
Fix flickering grass	Turn off 2 sided	3	X
Temple Doors		2	X
Ragdoll on enough punch damage, less than normal ragdoll		2	X
Make bridge as wide as two heavies side by side	Shrink enemy avoidance radius	2	X
Increase performance of NavMesh agents		2	X
Clouds		2	X
Change "Game Over" to "You Died"		2	X
Water sound		2	
Polygonal particles instead of round dust bunnies		2	
Lanterns		2	
Headbands waggle around		2	
Environment soft is too hard	Set to very soft but things bounce off like hard	2	
Encourage enemies to take stepping stone paths		2	
Bow color	Not gray - maybe bamboo Not using rig so can replace with own mesh	2	
Arc color change		2	
Tutorial start pillar: no text, show arrow after timer		1	
Steam VR Action Boolean Bug		1	
Spawn enemy sound played during tutorial		1	
Rock melee tutorial slide after resize slide	Make obvious that rock is still in hand after hitting enemy	1	
Put tutorial in the pause menu		1	

Put Main Menu / Start from Beginning in death menu		1	
Pillars should fall into the ground on exit and David's stars dissipate into particles		1	
In spike tutorial, spike area shouldn't get bigger on descent		1	
Hurt circle more transparent		1	
Highlight rock when other hand is intersecting		1	
Higher gravity for arrows		1	
Fall damage		1	
Enemy in water physics / animation		1	
Directional damage marker	Preset 8 directions (4 sides 4 corners) images	1	
Create a non-tutorial location 1 wave(s)	And in skip tutorial, go to first wave after tutorial at loc1	1	
Audio cue for headshots		1	
Arrow UI for telling player to look at hands	Power and low health	1	

Appendix F: Ritsumeikan and WPI Collaboration

In addition to our game project, our host school in Japan, Ritsumeikan University, tasked our team with a project of their own. We, along with the other WPI students and some students from Ritsumeikan, collaborated on creating a physical model of a heart simulation created by a Ritsumeikan Lab. The existing program used a very technical graphical user interface (GUI), making it impossible for someone to read it without understanding the process behind it all. The goal of our model was to make the results of the simulation clear to a person without any knowledge of the underlying biological systems at play. This way, our abstraction of the heart simulation could be used for educational purposes.

User Interface

For the user interface, the user can select between five different scenarios which represent different conditions of the heart. Our scenarios are composed of:

- Rest
- Old
- Exercising
- Cold
- Warm

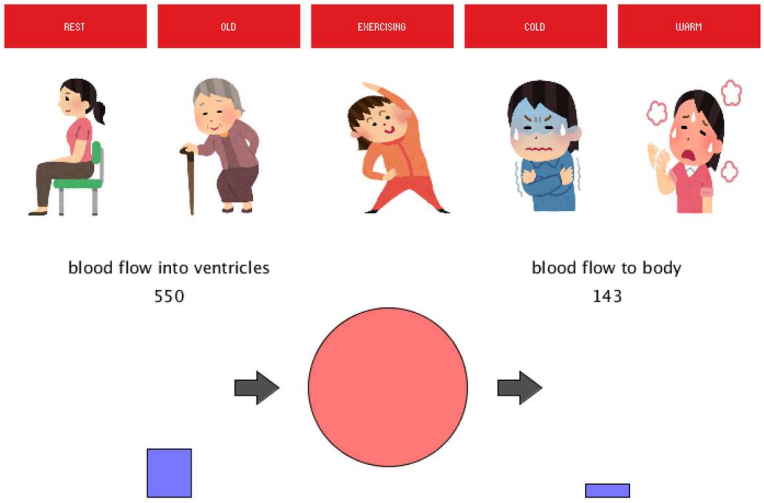

Figure 72. User interface with five different heart scenarios

Rest is our default scenario, where all the input values represent a healthy person in a resting state. Old scenario, representing the heart of an old person, means that compliance and resistance are higher, therefore aortic pressure is increased. Exercising refers to an increase in heart rate, as well as an increase in pressure from the lungs producing a higher aortic pressure. Cold scenario sets a high resistance that also causes high aortic pressure, because coldness constricts blood vessels. Warm scenario indicates a lower resistance that causes lower aortic pressure.

Additionally, the user interface has two bar graphs to represent blood flow into the ventricles and into the body. The red circle displays changes in the volume of the left ventricle. Figure 72 displays our final user interface design.

Hardware

We attached all of our hardware components to an Arduino Mega 2560 programmable microcontroller. This board runs all of our code and controls all of the electrical components. To show the blood flow and blood pressure, we utilized NeoPixel individually addressable light emitting diode (LED) strips. Each strip has the ability to set any of its lights to any color with 24-bit color depth. We attached two LED strips to our model—one representing the simulated blood flowing into the left ventricle and another representing that flowing into the body—to show the pressure and flow rate in the vessels we needed to visualize.

In addition to the LEDs, we used an air compressor with two solenoid valves—valves that can be switched open or closed using an electrical current—to inflate and deflate a balloon. We tied the balloon shut and dangled it inside a two-liter plastic bottle. When the air compressor feeds air into the sealed bottle at a high pressure, the balloon then deflates due to the increase in pressure outside of it. We then used the Arduino board and solenoid valves to toggle between filling the bottle with the air compressor and draining the built-up pressure back into the world.

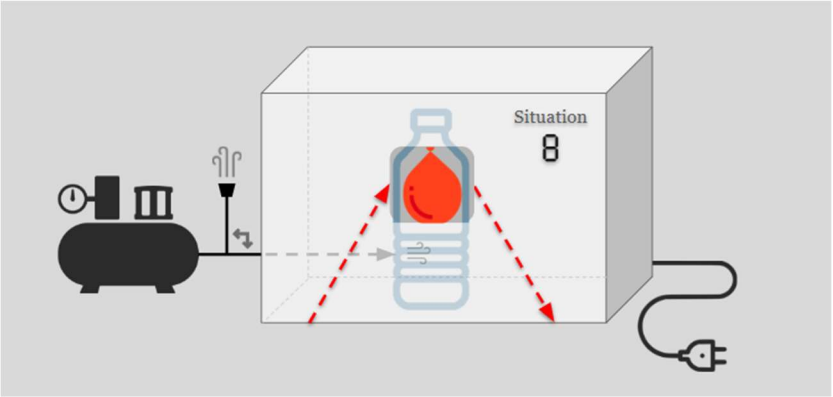

Figure 73. Final proposed design for heart simulation

Finally, we used a seven segment display and a buzzer to show additional information that is not calculated in the simulation. The number display is used for feedback from our GUI, showing what scenario the user has selected. The buzzer beeps once every heartbeat cycle after starting the simulation. While not corresponding to anything concrete about the state of the heart's pressure and volume, this beep provides a familiar, "electrocardiogram-like" sound to

inform the user of the heart rate. Figure 73, shows our final design for the hardware component of the project.

Software

The team's goal was to demonstrate the diastole and systole processes of the heart. For simplicity, the simulation only calculates information about one of the four chambers of the heart: the left ventricle. Diastole is when the heart fills with blood and systole is when the heart contracts sending the blood to the body. Through the simulation, the model is able to calculate the following variables that represent these two processes of the heart:

- Flow rate into the heart (f_v)
- Flow rate into the body (f_l)
- Pressure from lungs (P_v)
- Aortic Pressure (P_a)
- Pressure in the left ventricle (P_{lv})
- Volume in the left ventricle (V_{lv})

Pressure from lungs and aortic pressure are represented through the colors of the LED lights, blue meaning low pressure and red being high pressure. Flow rate into the heart and into the body, is represented through the speed the LED lights move. This functionality is handled by an animation function which displays a sine wave with the lights in the specified color. The offset of the sine wave is updated every step with a value directly proportional to the flow rate of the specific blood vessel in the simulation, making the crests appear to move. Also, a certain portion of the trough section of the sine wave sets the light intensity to zero, creating more separated segments of moving light.

Since pressure and volume in the left ventricle in our simulation are correlated, we decided to use volume in the left ventricle as the trigger for the air compressor. When volume in the left ventricle starts decreasing, the air compressor fills the bottle with more air, increasing its pressure and deflating the heart balloon. On the other hand, when it increases, excess air is released from the bottle, inflating the heart balloon. Figure 74, displays our variables and what they represent for the heart model.

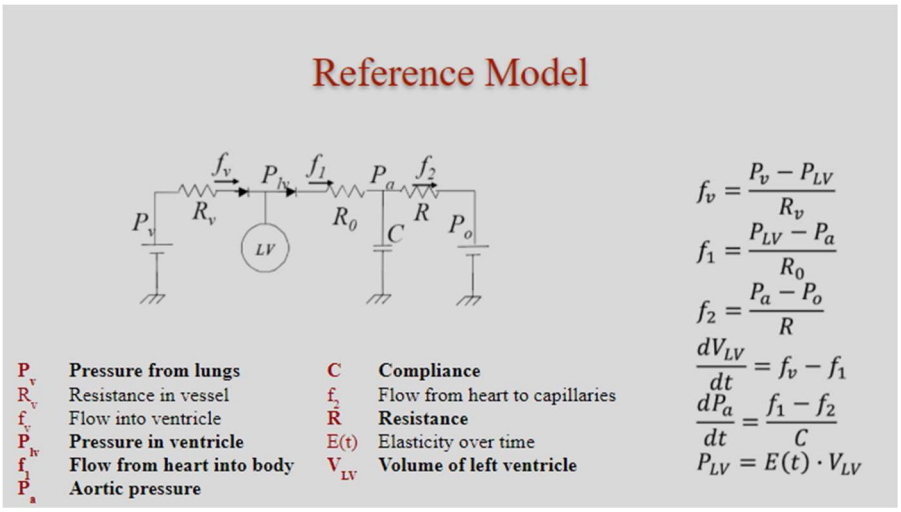

Figure 74. Reference model with all variables for the heart simulation

Conclusion

Through this project, our team was able to successfully complete a functional physical model of the heart (Figure 75). This model demonstrated volume change inside the heart through the use of air pressure and a balloon. Additionally, we approximated our input parameters to recreate five different heart scenarios. Finally, we developed a simulation program that calculates heart conditions and controls LEDs, valve switches, and a buzzer to replicate the results in an understandable, physical format.

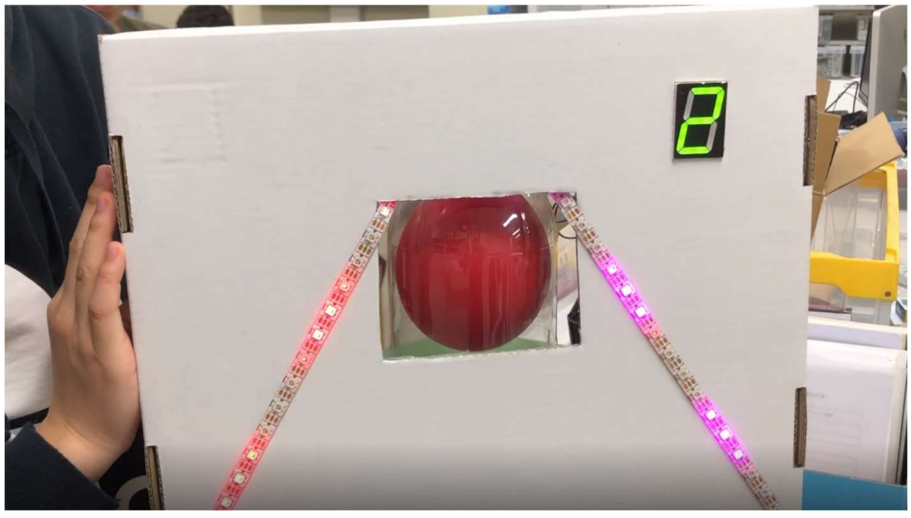

Figure 75. Final heart physical model