Hikari Hook: Game Development with the Oculus Rift and Handheld Motion Controllers

Eric Benson, John Breen, Chris Knapp, Sean Halloran, Andrew Han

November 8, 2015

A Major Qualifying Project Report:

Submitted to the faculty of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the Degree of Bachelor of Science

Approved by:

Professor Robert Lindeman, Advisor

Abstract

In seated virtual reality experiences, the player may feel disconnected from their character if walking is the primary method of traversal. The goal of this project was to explore alternative methods of character traversal to avoid this disconnect. *Hikari Hook* is a virtual reality game we developed in collaboration with Takemura Lab at Osaka University. In the game, players use a pair of grappling hooks to solve environmental puzzles and traverse the forest. Testing performed with students from Takemura Lab was used to improve the player experience.

Acknowledgements

This project was only possible thanks to all the support we received from many organizations and individuals. Either by providing guidance or providing the necessary equipment, these people helped us to complete a successful project.

The first organization we would like to thank is Worcester Polytechnic Institute (WPI). Without the support from WPI and the Interdisciplinary and Global Studies Division (IGSD), this project would never have been possible. We want to thank them for taking the time to organize and plan the logistics for the project in Japan. In addition, we would especially like to thank Professor Lindeman for taking the time to pull the late nights to remotely advise the project. The weekly meetings provided us with the encouragement and direction that allowed us to fully unlock the potential behind our ideas.

We would also like to thank Osaka University (OU) and specifically Takemura Lab for hosting us and providing us with the necessary equipment and workspace. Special thanks to Professor Kiyokawa for taking the time to provide us with any assistance we required during our time in Japan. Also to the members of Takemura Lab who provided assistance and feedback to the project during our numerous playtests, we would like to thank you for your time and help. We hope this is but the start of a lasting friendship.

Finally, we would like to thank all those not mentioned above who helped us with the project. No matter how small the contribution, the team would not have been able to achieve as much as it has without their support.

Table of Contents

List of Figures

1 Introduction

Hikari Hook is a virtual reality game developed for the new upcoming generation of virtual reality systems. It was created using Unity 5.2 in conjunction with the Oculus Rift. The game revolves around the player, who is attempting to navigate a forest canopy by solving various puzzles. In doing so, he will uncover the mysteries surrounding the forest and supersede his predecessor. In order to navigate the forest, the player must make use of the "Lighthook", an otherworldly device which allows them to tether to objects using beams of light. Using the ability to grapple and tether, the player swings from tree to tree to traverse the forest.

During the creation of *Hikari Hook*, many new virtual reality systems were starting to come to market for public consumers ("SteamVR"; "FOVE: The World's First Eye Tracking Virtual Reality Headset"). This presented a unique opportunity to create something targeted at a new, upcoming field in gaming. In the area of virtual reality games, there were very few complete games, because it was such a new and developing field. In many cases, the available content for the virtual reality systems lacked completeness and functioned as more of a demo. *Hikari Hook* attempts to surpass these games, providing players with immersive and comprehensive gameplay by utilizing motion controllers, camera rotation, the Oculus Rift, and many other techniques.

Virtual reality has the potential to create life-like scenarios and allows users to enjoy a real-world experience from the comfort of their homes. However, a virtual reality application is not inherently immersive or life-like; a lot of work has to be done in order to allow the user to interact with the game naturally and make it feel real. In order to establish the sense of realism and interaction required of an immersive experience, *Hikari Hook* employs movement in the form of swinging to avoid the disconnect typically caused by a walking in-game character and the player's stationary, physical body. Meanwhile, the use of motion controllers for aiming allows the player to physically control their movements and interact with the environment while constrained to their seat in the real world.

By combining different aspects from the fields of virtual reality and gaming, we hoped to further support the role that virtual reality can play in the future of gaming. *Hikari Hook* attempts to push the boundaries, redefining what is viable for an immersive virtual reality game. Part of our work involved breaking some specific conventions that are generally followed when creating virtual reality applications. By deliberately doing so, we hoped to create unique ways of immersing the player, improving the overall experience of the game.

The following background section provides information on the inspiration for the game, some of the designs that were employed, and general background about the systems used in the game. In the Methods section, the overall design of the game is explored as well as the implemented features. It provides detailed explanations on the mechanics of the game along with the algorithms behind them. Finally, the last section of the paper goes into detail on what could have been done differently in the game making processes, the final results of the game itself, and future work that could be done.

2 Background

Before the development of *Hikari Hook* started, there was a lot of preparation and brainstorming done. This work was critical to the development of *Hikari Hook*. Many of the ideas described in this section had a significant impact on the direction of the game and how it was developed, whether or not they were actually incorporated into the final project.

2.1 Rejected Game Ideas

In the months leading up to the project, several game ideas were iterated through before we settled on the grappling hook concept. These ideas were rejected for a variety of reasons, including hardware accessibility, the resources that were available to dedicate to the project, and other design decisions. This section lists the game ideas that were under consideration for some period of time, along with a brief explanation of why they were ultimately rejected.

2.1.1 Pro-bending Simulation Game

One of the earliest game ideas took inspiration from a fictional sport in the world of "Avatar: The Legend of Korra" called "Pro-bending". The virtual reality game would involve players making gestures mimicking martial arts in order to attack enemies with the elements of earth, water, and fire. It would also utilize tracking of the player's location in the room to allow the player to move around in order to dodge incoming attacks. The viability of this game idea was reliant on our ability to acquire an HTC Vive development kit. The HTC Vive would provide all the motion tracking functionality needed to make this game work in a single product that would soon be shipping to consumers. While we were aware of other hardware that could have provided the location tracking necessary for this game, we preferred to make a game reliant only on technology readily available to consumers. This would improve the size of our audience that could play the game. We applied for the development kit in Spring 2015 (

Appendix A: SteamVR HTC Vive Application), unfortunately with no response from SteamVR. As the start of our project got nearer, we decided to proceed to other ideas.

2.1.2 Desert Motorcycle Survival Game

When we decided that we would develop for the Oculus Rift, we started brainstorming game ideas that would make interesting seated experiences. A vehicular combat game, inspired in part by the films *Mad Max: Fury Road* and *Akira*, was proposed as an idea where the player could feel immersed despite staying seated while playing. The player would drive a vehicle of some sort in first person, while taking advantage of their ability to look around by using various weapons to fend off enemies. This is one of the ideas that went through the most iteration. We were planning on a procedurally generated terrain in order to more easily produce a world with the proper scale. Another idea we considered incorporating was the earth-manipulation mechanic from the Pro-bending game as an alternative way of interacting with the enemy riders. The player could create ramps and walls to leap through the desert and impede their enemies. While this was one of our more promising ideas, we abandoned it due to concerns about the resources we could put into the game and how complete we could make it feel after the three months we had to work on it. With the limited time frame, we felt as though we would not be able to create enough varied content to make the game interesting to play for extended periods of time. The nature of this game idea also meant that we would need a significant amount of character art for the enemies that the player would interact with. In order to lessen the work-load on our team's single artist, we decided to opt for an idea that required fewer characters to work.

2.1.3 Color Restoration Game

With our next game idea, we sought to reduce the scope of our project so that we could still deliver an interesting game experience despite our limited time frame. The core concept of the game was a world that had been stripped of its color. The protagonist would travel through the open world to defeat bosses guarding the colors. Defeating a boss would restore that color to the world and impart a new power to the player. We were considering powers that would be interesting to use in the Oculus Rift, such as teleporting, or shrinking and growing in size. Acquiring and learning how to use these powers would be the key to defeating the final boss

and restoring the world to its former glory. We stepped away from this idea due to a couple of reasons. Once again, we were slightly concerned about our limited resources for asset development. An idea like this would put a significant amount of pressure on our artist, since the art direction and quality could make or break the game. Furthermore, we felt that with the number of powers and abilities in this game, the gameplay would feel unfocused, like a mixed bag of powers that exist merely as a proof of concept. For our next step, we tried to hone in on a single mechanic that would make for an interesting game in the Oculus Rift.

2.1.4 Boss-centered Grappling Hook Game

Our first draft of a grappling hook-based game involved a large boss to overcome, much like the color restoration game. We envisioned using grappling hooks to climb a boss on the scale of the colossi from *Shadow of the Colossus*. We settled on grappling hooks as our main mechanic because it was a method of movement that granted the player a lot of freedom of motion. This meant the player could move around quickly without walking, avoiding the disconnect that exists between a walking in-game character and the stationary player. We eventually decided to design levels for the player to overcome, rather than a boss to defeat. This focused the challenge of the game into traversing the environment, which became the basis for *Hikari Hook*.

2.2 Abandoned Game Mechanics

During the design process of the game, many ideas were brought up and rejected due to time budgeting and our emphasis on fun gameplay. However, many of these ideas had a major influence on how several of the design choices concerning key aspects of *Hikari Hook* were made when moving forwards.

2.2.1 Monsters / Fighting:

One of the first ideas brought up during initial brainstorming was the idea to incorporate fighting into the game. Although fighting would undoubtedly add to the gameplay, it would also potentially add an unnecessary amount of complexity. Being able to produce monsters and an enticing fighting system would take a lot of time and focus on both the programming and art sides of the project. Due to the main focus of the game being on

maneuvering through the environment using grappling hooks, we decided to abandon the adversarial component of the game. In addition to adding to the complexity of the movement mechanics, adding an adversarial component to the game also could have detracted from the main concept. Because of this, we decided to focus solely on platforming, as it did not require the production of enemy assets and would allow us to refine the ability to grapple and move through swinging.

2.2.2 Open World

Another scrapped idea was an open world approach to the level design. It is far easier to direct a player in a non-open world environment, allowing us to design levels and appropriate challenges that cannot be easily skipped. In addition, making an open world game would also require a much larger space for the player to travel through. This was something that was not feasible given the constraints placed on the project. Having a huge open world would also greatly increase the strain on the graphics of the game. However, the idea was not completely abandoned. An initial open world area where players could practice grappling was planned to be implemented towards the end of the project. However, due to time constraints and the aforementioned reasons, the idea was later discarded.

2.2.3 Moving Anchor / Grapple Points

Moving or swinging grapple points were proposed as another dynamic environment challenge that the player would have to overcome. They would force the player to master the timing of their shots as well as their aim. However, this idea was later rejected due to time constraints and its similarity to other dynamic environments. The other dynamic environment challenges were chosen over moving anchors and grapple points due to their originality and a work to benefit analysis.

2.3 Virtual Reality Head Mounted Displays

We began brainstorming game ideas in the middle of 2015, right as companies like Oculus and Valve announced 2016 plans to release consumer editions of their virtual reality head mounted displays. The two we found most notable were the Oculus Rift and the HTC Vive.

Figure 1: Oculus Constellation tracking camera, Middle, Oculus Rift HMD, Right, bundled wireless Xbox One controller (Rift)

As seen in Figure 1, the Oculus Rift is scheduled to launch in Q1 2016 (Rift). Although not all of the official specifications have been released, the headset, seen in the center of Figure 1, will have a high resolution display, integrated headphones, and both head position and rotation tracking with the Oculus Constellation vision tracking system, seen on the left in Figure 1. We had been anticipating the release of this headset for years before the announcement of the 2016 date, so we were very excited to make a game that could be played on it not too long after the completion of our project.

Figure 2: The HTC Vive Developer Edition HMD (Nickinson)

The HTC Vive (Figure 2) developed by HTC and Valve Software is a head mounted display with similar specifications and capabilities to the Oculus Rift. It also has position and rotation tracking, however with a different proprietary tracking system called Lighthouse (Buckley).

Due to the upcoming availability of these two similar headsets, we set out to support both with our game.

2.4 Motion Controls

Along with the incorporation of a head mounted display, another thing we decided on early was that we wanted to have motion controls incorporated into our game. When done well, we believe motion controls add an extra layer of player interaction and immersion into a game. We felt that this, in conjunction with the immersion that the Oculus Rift brings, would create the kind of experience we wanted. In order to accomplish this, we considered a few different motion controllers.

2.4.1 HTC Vive

The first of the motion controllers we considered was the HTC Vive (Figure 3). The pair of controllers and corresponding HMD are tracked extremely precisely using multiple beacons that keep track of the location of all peripherals at all times through the use of photosensors. The controllers also track their orientation and the direction in which they are pointing with accelerometers and gyroscopes.

Figure 3: HTC Vive Motion controllers

2.4.2 PlayStation Move

Another motion controller considered was the PlayStation Move (Figure 4). The controller uses a camera mounted in front of the player to track a lit ball on the tip of the controller in order to measure its current position. The orientation of the controller is tracked by an accelerometer and a rate sensor, which measures the angles of the controller, similar in function to a gyroscope. The second controller, the navigation controller, is not tracked however. Instead, the navigation controller only has a button and joystick inputs without any motion tracking component.

Figure 4: PlayStation Move controller with motion tracked ball on end

2.4.3 Razer Hydra

The last controller we considered, and the one we ended up incorporating into our game, was the Razer Hydra (Figure 5). The Hydra consists of a pair of controllers with a joystick, trigger, bumper, and five buttons each, with the position and orientation tracked through electromagnetic sensors by a base placed in front of the player.

Figure 5: Razer Hydra controllers and attached electromagnetic base

3 Methods

Development of *Hikari Hook* was performed in Japan at Takemura lab, an international lab at Osaka University that specializes in Virtual and Augmented Reality, among other fields. *Hikari Hook* was developed using Unity 5.2 along with the Oculus Rift plugin. The game has a wide set of features, which work together and complement each other to create an immersive play experience. These features represent both the technical and artistic challenge that went into creating a complete game.

3.1 Lighthook

The main mechanic in *Hikari Hook* is the "Lighthook", a grappling hook-like device which allows players to connect and tether to objects with beams of light. In the game, the player has two Lighthooks, which enable the player to navigate through the game environment and solve the various challenges presented by each level (see Figure 6). The Lighthook enables the player to perform three essential actions: Fire, Reel, and Swing.

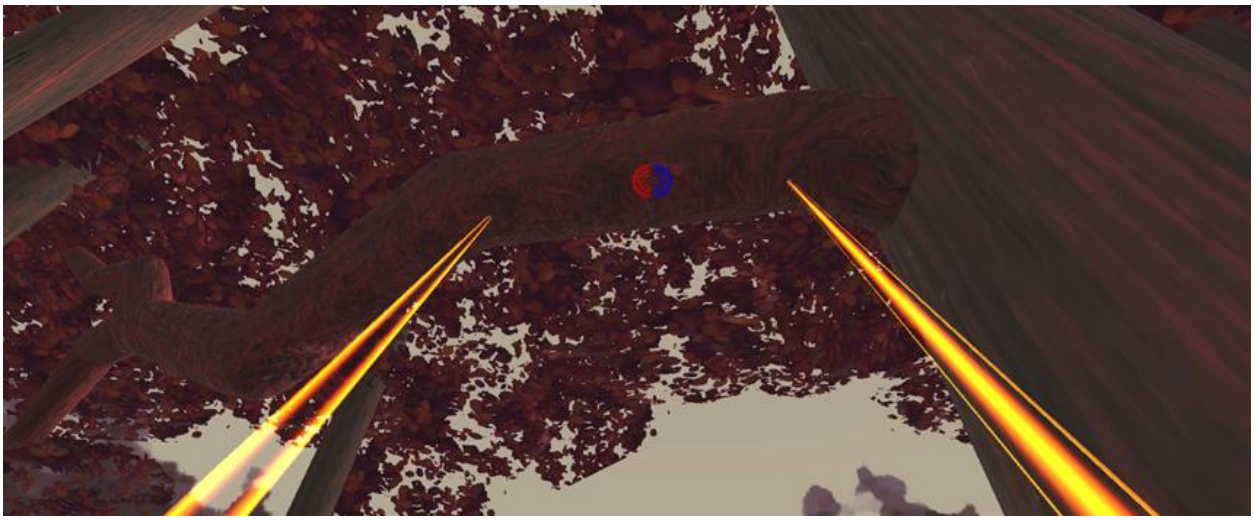


Figure 6: Player using both Lighthooks to tether to an object

The firing action of the Lighthook is made up of two different possible states: connected and disconnected. Initially, the Lighthook starts off in the disconnected state. When the player attempts to fire the Lighthook in the disconnected state, it sends out a hook object to the targeted location. In the game, there are two types of surfaces that interact with the hook:

grapplable and nongrapplable. If the hook hits a grapplable surface, the Lighthook moves from the disconnected state to the connected state and creates a connection between the player and the anchored hook. However, if the hook collides with a nongrapplable surface, the hook will bounce off and the Lighthook will remain in the disconnected state. When the Lighthook enters the connected state, its functionality changes; in the connected state, a beam of light connects the player to the hook. The beam of light is restricted to a maximum length, functioning like a section of rope. While the player can move closer towards the hook, the player cannot move further away from the hook than the distance at which the beam was created. Firing the Lighthook in the connected state will remove the tethered connection between the hook and the player, returning the Lighthook to a disconnected state.

Another main function of the Lighthook is the ability to reel when the Lighthook is connected, reducing the length of the tethered light beam. In addition to reducing the length of the beam, the reeling action also speeds up the player in the direction of the anchored hook. This allows the player to pick up speed when moving towards objects and fling themselves to reach objects beyond the grapple point. While the player can reel in their light beam, they cannot unreel. Once the length of the light beam has been shortened, the player cannot increase the length of the connected beam unless they disconnect and reconnect. This design decision was made to reduce the complexity of the game and make the gameplay feel more fast-paced. Allowing the player to increase the length of the rope would require another action that the player would have to be aware of. In addition, being able to increase the rope length is unnecessary for solving the game's puzzles and potentially slows the game down as the player carefully adjusts their position by inching up and down the rope.

Figure 7: Player swinging using the Lighthook and gravity

The last core mechanic of the Lighthook is the ability to swing. Since the Lighthook restricts the maximum length the player can travel from the hook, it also enables the player to use the power of gravity to swing like a pendulum. Figure 7 shows how the player can tether to an object and then swing. In conjunction with reeling, swinging is one of the core mechanics that allows the player to move from location to location.

Throughout the game, the player must utilize the Lighthook to traverse the environment and overcome obstacles that have been placed throughout the levels. The various obstacles will force the player to constantly think of new ways to apply the three main functions of the Lighthook. These challenges test the player's aim and reaction speed as well as critical thinking and problem solving abilities.

3.2 Controls

One of the first things we decided about our project was that it would utilize virtual reality to some degree to control the character. Our aim was to acquire an HTC Vive with its head mounted display and hand tracking motion controls. However, our application for an early

developer kit yielded no response. Without a Vive to work with, we decided to fall back on using the Oculus Rift Development Kit 2 and looked into other controllers as our means of input. We decided to base our default control scheme on the Xbox One controller, due to announcements that the consumer edition of the Oculus Rift would ship with an Xbox One controller. This would allow our game to be immediately accessible to anyone who purchased a consumer edition of the Oculus Rift. However, we still wanted to look into other avenues of control that utilized hand tracking to some degree, in order to further the virtual reality experience. In this vein, a couple different controllers were explored. To fit our game, controllers needed to fit certain criteria, namely having a very low latency, high accuracy, and preferably having controls beyond just gesture recognition.

3.2.1 PlayStation Move

The first motion controller that was seriously considered was the PlayStation Move, as there were already kits available to work with in Takemura Lab. The approach we planned on taking was to utilize two separate controllers with a means of setting each controller to recognize which hand it is being held in. At first, it appeared that the controller would suit our needs, providing the degree of precision needed to accurately track and correlate the player's real hands to their in-game ones. Unfortunately, upon attempting to integrate the PlayStation Move into our project, it was discovered that it is incredibly non-user-friendly to set up on a Windows platform. This was due to the fact that the PlayStation Move controllers were built specifically to run with Sony's Unix-based systems, and were never intended to be run on Windows. In addition, the amount of work required to hook into the Windows bluetooth stack proved to be a much larger hurdle than was originally expected. With the additional factor of Windows-only support on the Oculus Rift, the use of the PlayStation Move controllers seemed very undesirable. As a result, the idea of using a PlayStation Move was abandoned.

3.2.2 Razer Hydra

The next controller looked at was the Razer Hydra. The Hydra appeared to be a perfect fit for the game. It provided all the same standard gamepad controls as an Xbox controller. More importantly, it would allow us to use a dual joystick setup that most players are probably already accustomed to in order to move their character. In addition, the bumpers and triggers

allow us to use a controller configuration similar to most first person shooters for firing the grappling hooks, making the game more accessible to our players. The position of the controllers is tracked with extreme precision, using electromagnetic fields, in addition to the rotational orientation of the controllers.

Using this controller was not without its own hurdles. First of all, acquiring a Razer Hydra in a reasonable time frame that was also affordable was no small task, as the Hydra had been discontinued for quite some time. After acquiring a Hydra, it was discovered the copy we had ordered was slightly faulty. Readings taken from the joysticks were erratic at times. The Hydra also did not directly hook into Unity's default input manager to run alongside the previously implemented Xbox controller support. Rather, special input handling had to be implemented specifically for the Hydra. Finally, as the Hydra is tracked using electromagnetic fields, this led to inaccurate readings at times, due to interference from numerous electronics and ferrous metals in the area.

Taking all of this into account, our implementation of the Hydra controls attempted to mitigate erratic readings. The position of each Hydra controller is constantly polled. Each value returned is normalized and adjusted by a slight offset. This does not catch all discrepancies; should the value of the controller's position differ noticeably in the direction away from the player than where the player is actually holding the controller, a mismatch will occur. However, should the difference mostly be in the magnitude of the reading, the normalization will help to keep the in-game position of the player's hands close to where they would be normally. In addition to this, there is another offset applied to the controller position readings that is unique to each player. This is calculated by the player calibrating the controllers by placing the controllers at their shoulders, then pressing a button to confirm. Together, these calibrations attempt to make the position of the player's hands in-game feel as close to a one to one ratio with their movements out of the game, minimizing any potential disconnect. This is all done through incorporating the Sixense API; interfacing with the controllers and passing the data off to an input parser that would know which controller should be used in order to determine what commands the data received correlate to.

The control scheme is largely standard and straightforward. Each bumper fires the grappling beam of the respective side of the controller, aimed by pointing the controller. The triggers reel in the corresponding beam. The left control stick moves the player while the right control stick rotates the player. This control scheme is designed to be as straightforward and intuitive as possible, to make the gameplay feel more natural and therefore more immersive.

3.3 Level Design Process

Level design started very early on in the development process to ensure that there would be enough time to critique ideas and iterate on our designs. Our first step was individually creating "micro-levels", or short puzzles that would eventually become part of a level. The goal of this step was to generate a sizable pool of level design ideas to work from. From these micro-levels, we could get a sense of the different directions that each of us planned on taking the game. Each of us produced three to four simple diagrams of obstacles that the player might have to overcome (Figure 8).

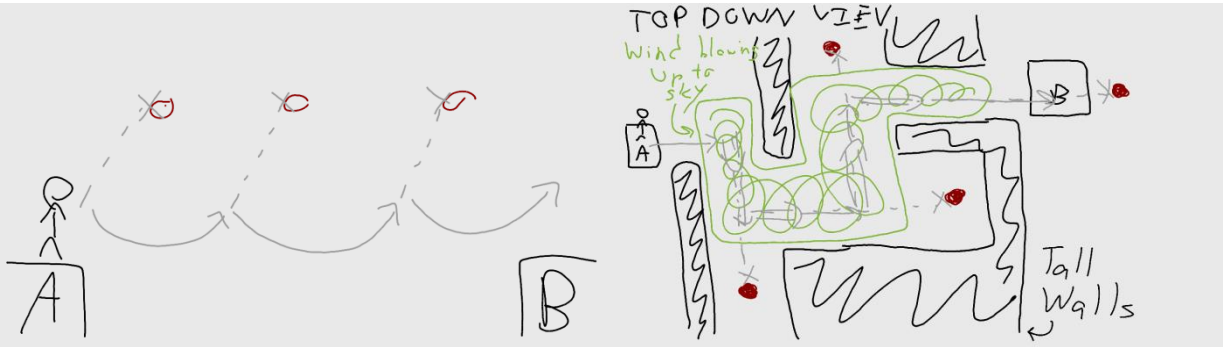


Figure 8: Example micro-levels

In level design meetings, we discussed the micro-levels that were submitted and came up with some fundamental rules for the design of our game. For example, through these meetings, we determined that the main challenge presented by the game would be in timing swings properly, rather than having grapple points that are hard to hit. We used these fundamental rules as well as personal judgment to determine which micro-levels would be a good fit for the game.

After settling on which puzzles we would use in the game, we started breaking them up into three levels. We roughly sorted the micro-levels by difficulty and the number of hooks that

the player would need to use in order to complete it. We also grouped puzzles involving similar challenges together in order to make each level feel cohesive. These rough sets of micro-levels were then implemented in Unity and arranged into levels. We then iterated on these levels to make adjustments to difficulty and increase the overall player experience.

3.3.1 Level Design Assisting Tools

The process of taking a micro-level concept and implementing it with assets in three dimensions often relied on the careful placement of grapplable surfaces to be either just within or just outside of the player's range to fire at and attach to. At first, we placed these assets with estimates and guesswork; however it quickly became apparent that we needed a tool to take out the uncertainty. The tool we made was a Gizmo, a feature of Unity to draw extra information to a scene when in the scene editor. The Gizmo we made displays information about at what range from a grapple surface the player can do certain actions. This Gizmo can be seen in three dimensions in Figure 9.

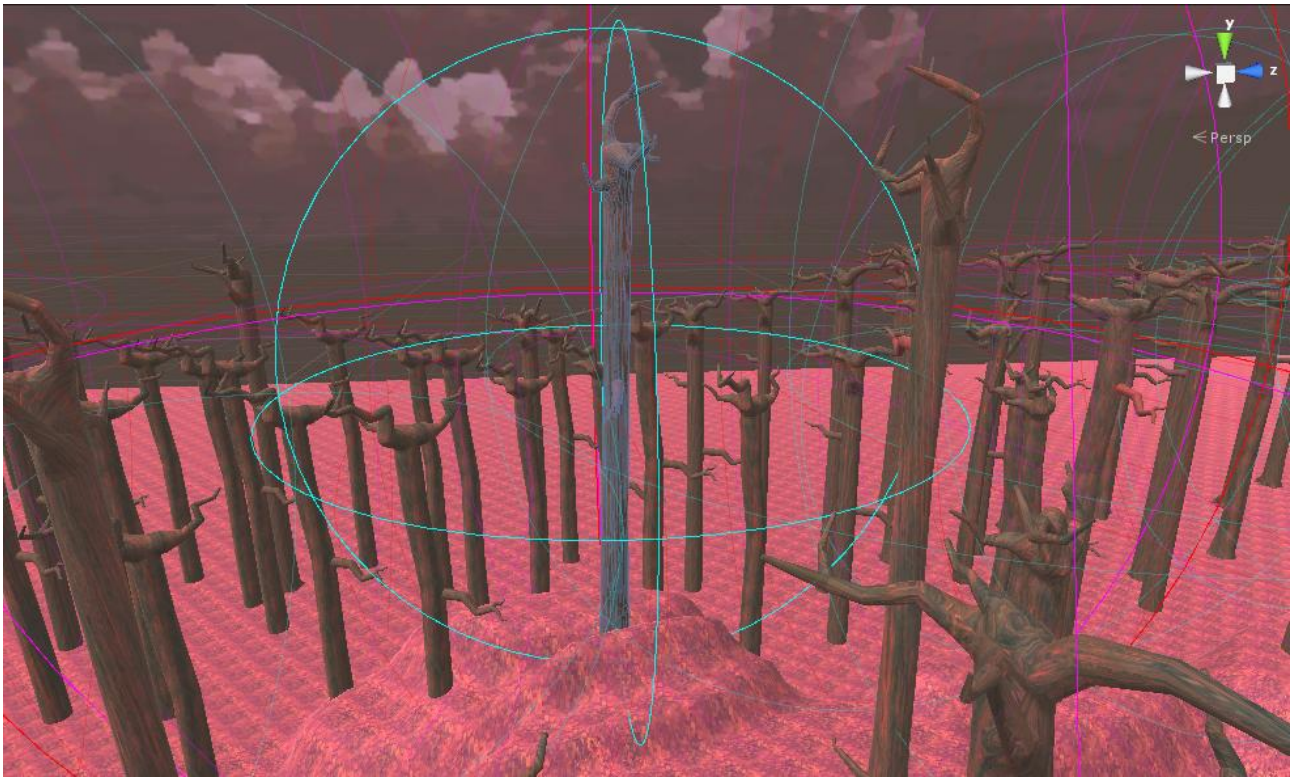


Figure 9: An example of a highlighted Gizmo showing info about a grapple point in a level

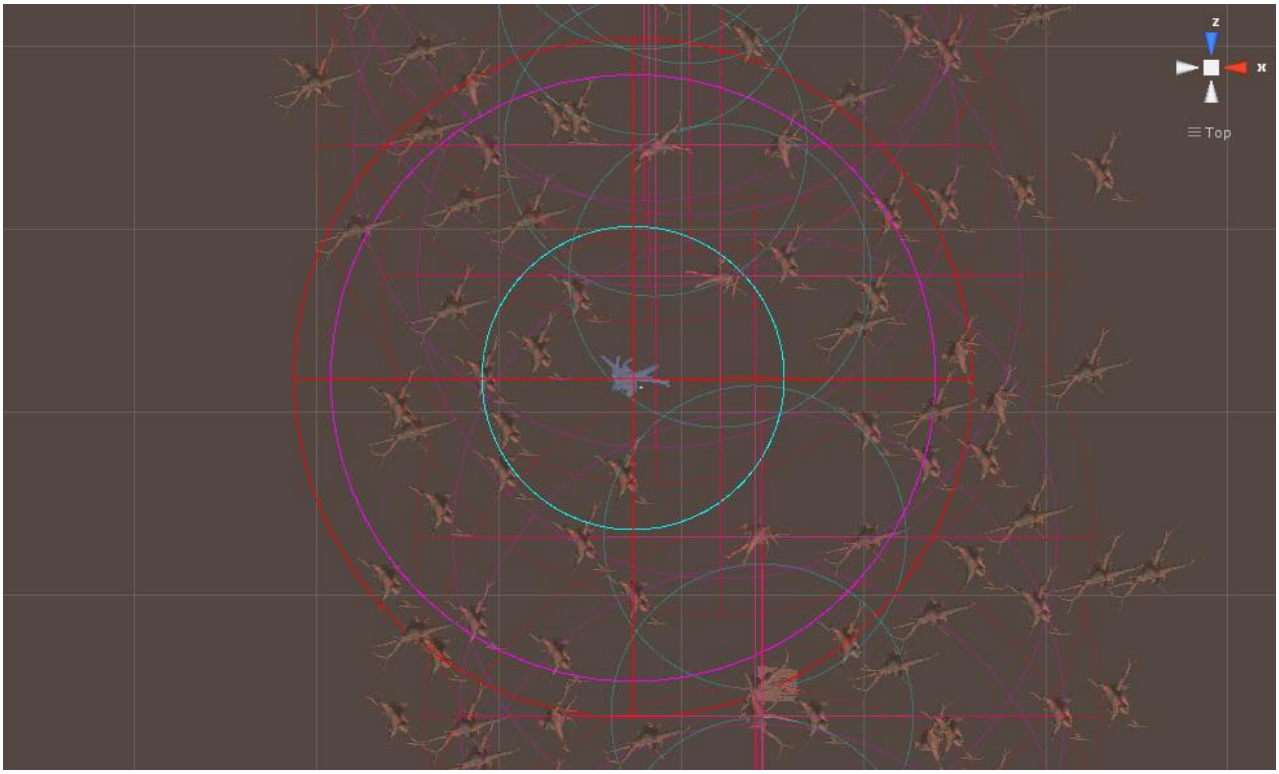


Figure 10: A top down isometric view of a highlighted Gizmo showing info about a grapple point in a level

The same Gizmo is seen in Figure 10 from a top-down, two-dimensional perspective. The Gizmo draws a cyan sphere around a grapplable surface. This is the area inside of which the player would be close enough to grapple to this surface. Outside of that is a violet sphere. If the player is swinging from this surface with one hook, inside this sphere is the range to which the player could shoot with their other hook. One step outside of this is a red sphere which is extended just a bit beyond the violet sphere to show the absolute maximum range of targets the player could shoot to when leaving from the grapplable surface at the center. Targets at the edge of this range could be hit by swinging off of the center surface and then flying through the air for a small distance.

3.4 Aiming

Aiming the Lighthook is a task the player will need to do nearly constantly throughout the game. However, we did not want aiming to be the primary source of challenge in any level.

Our design focused the challenge on planning and precise execution of level traversal, not on trying to hit a small target at a distance. With this in mind, we designed the aiming mechanisms for each of our game's control schemes with assistive tools to display helpful information right where the player is aiming, reducing the challenge of aiming.

3.4.1 Xbox Controller Gaze Aim

In first person perspective games played on a two-dimensional display with an Xbox controller, aiming any sort of projectile weapon is most often done with a two-dimensional crosshair fixed at the center of the player's gaze that marks where the player is aiming. In a two-dimensional space, the crosshair can be rendered as a sprite at the center of the player's display.

We chose to use the same style of aiming for players using an Xbox controller. In games played on a two-dimensional display, the right analog stick on an Xbox controller is typically used to tilt and pan the direction of the player's gaze, thus changing where they are aiming with the crosshair. In our game we do not allow the player to tilt their gaze with the right analog stick, because that would result in the player's head being oriented level with the horizon in real space but not in virtual space, which is disorienting. We do allow for camera panning with the right analog stick, which rotates the player thus adjusting their gaze and their aim. Most of the player's gaze adjustment is done by looking around with physical panning and tilting of their head. The precision of these movements is the reason we chose to use gaze aiming, where the player is always aiming at a crosshair fixed to the center of their gaze.

However in a three-dimensional space that players experience in VR, there is no two-dimensional display to draw a sprite for the crosshair on. To draw the two-dimensional crosshair in three dimensions, it must be placed on a plane somewhere in three-dimensional space. If the crosshair plane is very far away from the player, then objects in the environment will be able to move in between the crosshair and the player's viewpoint, making the crosshair useless for aiming. If the crosshair plane is close enough to the player's viewpoint so that objects are unlikely to occlude it, then the player's eyes cannot focus on both the crosshair and what they are aiming at. If you hold your finger in front of your nose, and try to look at both

your finger and any object a few feet or more in front of you, you will experience the same problem of not being able to focus on both. In addition, a crosshair drawn this way fails to assist in aiming, whether in focus or not. An object so close to one's viewpoint, if viewed with one eye closed, appears in a different place depending on which eye is looking at it and which eye is closed. The player's aim is not based on which eye they have open, so the crosshair must remain consistent no matter which of the player's eyes are open.

We tested two solutions to this problem, which both take the crosshair out of three-dimensional space and attempt to draw it on the player's head mounted display in a way that is both comfortable to view and in focus for the player.

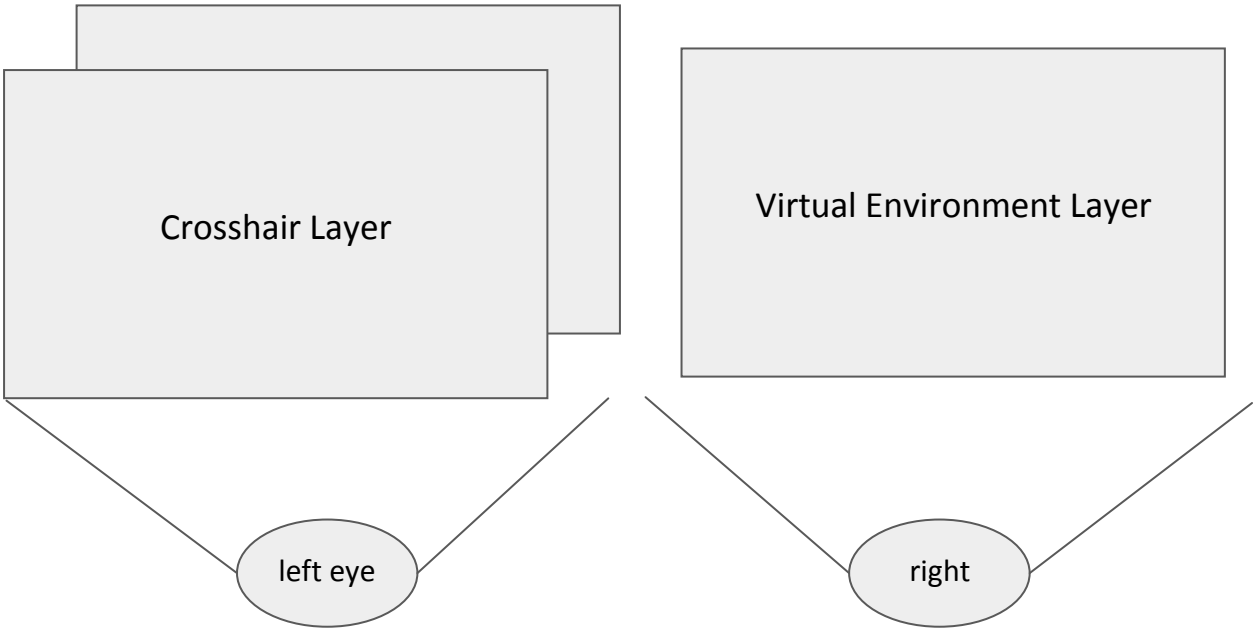


Figure 11: A diagram of the layers used to render the crosshair to one eye

Figure 11 shows our first solution, which was to draw a crosshair directly on top of the center of the image that is displayed in front of the left eye. The left eye was chosen arbitrarily; however we tried the same setup with the right eye and found no different results. With the crosshair drawn directly on top of the environment, even if parts of the environment were closer to the player than the crosshair appeared to be, the crosshair would not be occluded. The reason this was done for only one eye was so that the crosshair could only be seen as over one target instead of two depending which eye was open. If the player had their left eye closed,

then they would see no crosshair, which is better than an incorrectly placed crosshair. If the player had their right eye closed, then they saw the crosshair in the correct location, the same as where it was with both eyes open. In practice, this worked as intended, but at the cost of bothersome eye strain to the player. This was likely due to the player's eyes trying to focus on something that was only visible to one eye.

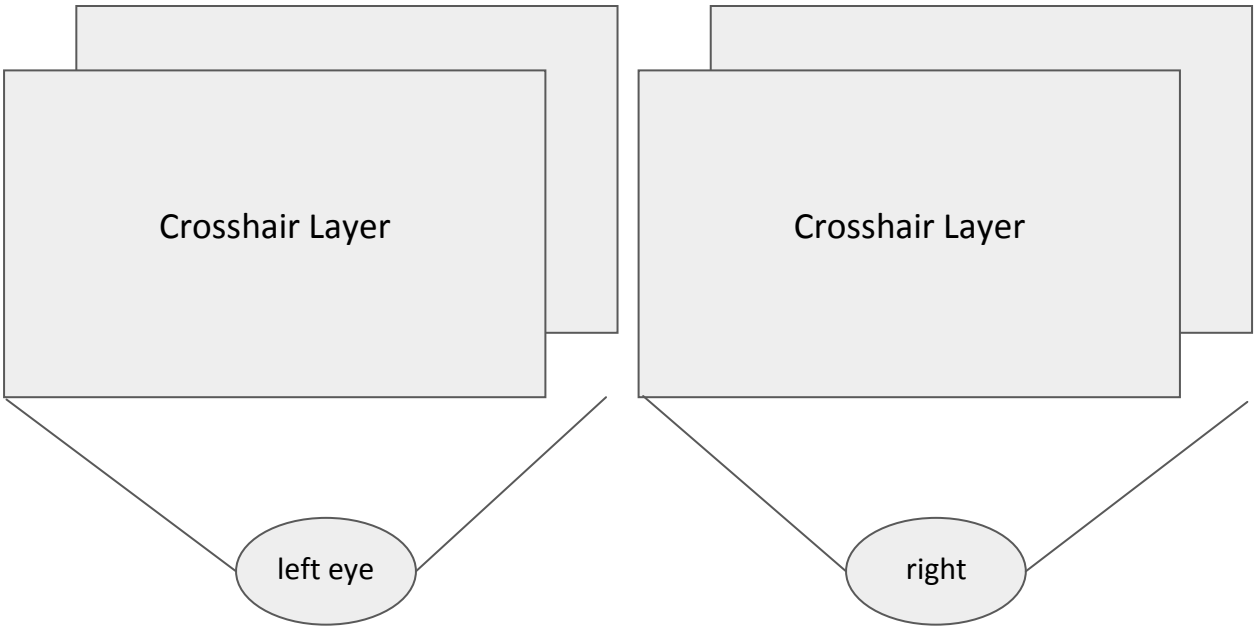


Figure 12: A diagram of the layers used to render the crosshair to both eyes

Figure 12 shows the solution we ended up using in the game. The crosshair is drawn on both eyes this way. The crosshair is drawn as a very large sprite on a plane in three-dimensional space very far away from the player, but in this case, it is drawn last so it occludes the environment even though objects in the environment are closer to the player's viewpoint than the crosshair. The environment and crosshair each exist in their own space with their own distance from the player's viewpoint, and are rendered separately to frames for the left and right eye. The frames that contain the crosshair are placed on top of the frames with the environment, as seen in Figure 12. The resulting effect is a crosshair that appears in the same place in the player's view no matter if either eye is closed, is always in focus, and is not significantly uncomfortable for the player's eyes.

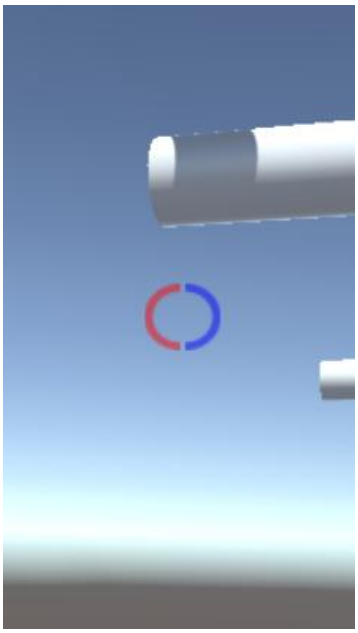


Figure 13: Dynamic Crosshair over no surface

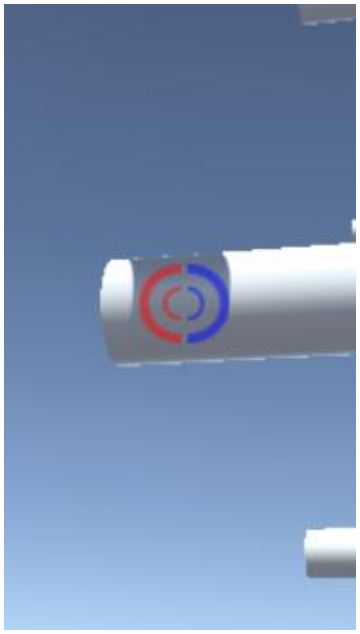


Figure 14: Dynamic Crosshair over grapplable surface in range

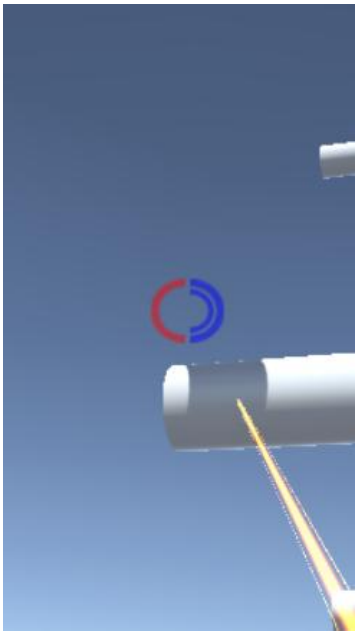


Figure 15: Dynamic Crosshair over no surface, and with right Lighthook attached

The first iteration of the crosshair was just an X drawn at the center of the screen. In its current iteration, the crosshair is used to display information about the player's two Lighthooks. The crosshair itself that is being drawn has a left and right half each with three parts. The left half of the crosshair is for information about the left Lighthook and the right half of the crosshair is for information about the right Lighthook. In Figure 13 the outermost semicircles of the crosshair are displayed. These semicircles are always present and used just for aiming.

In Figure 14 the crosshair has lined up over a surface which is within range and grapplable, so the innermost semicircles have become visible to indicate that if the player fires a Lighthook, then that Lighthook is able to attach to the current aiming target. This helps the player know when to shoot as they are trying to line up a shot in situations like falling through the air.

In Figure 15 the middle right semicircle of the crosshair is visible because the right Lighthook is attached to a surface. This is important to know when aiming, since if the player

does not know a hook is already attached and they try to fire that hook, the hook will detach instead of firing.

3.4.2 Motion Controller Pointer Aiming

The goal of utilizing motion controllers for aiming was to make using the Lighthooks as natural as possible by allowing the user to point their hands at a target, press a button, and hit the target. When implementing this goal however, we had to overcome a few design challenges. The closest real world aiming experience to what we were trying to create is that of firing a pistol. The motion controller we used, the Razer Hydra, is held in one hand and aimed, just like a pistol. When a pistol or any weapon is fired, the projectile it shoots originates from its tip near the shooter's hand. However, with the Lighthook, we did not want to have the projectile originate from and anchor to the player's hands, because then the player might expect to be able to exert force on the light beam by pulling on it. Although this could have been an interesting feature, we did not want to include it because it would alter the gameplay significantly between Xbox controller and motion controller play, as the player would have no way of doing this with an Xbox controller. Furthermore, the Razer Hydra simply is not accurate enough at tracking to rely on small changes in its position as important input to the game. This led us to have a solid hook at the end of the beam of light which is fired from the location of the player's hands, which upon being fired traces the beam of light back to the player's hips.

First to assist the player with aiming, we put some primitive shapes that matched the tracking of the motion controllers and had ends that pointed like a gun. The experience was roughly like being able to point with one's hands in the game. However, given how hard it is to accurately aim that way in real life, this was insufficient for enabling players to aim accurately with the Hydra. We then switched the virtual hand metaphor for aiming from something pistol like to more of a laser pointer shape that emitted a thin bright beam in the direction it points, as seen in Figure 16. This helped aiming from a stationary position immensely; however, aiming while moving through the air was still difficult.

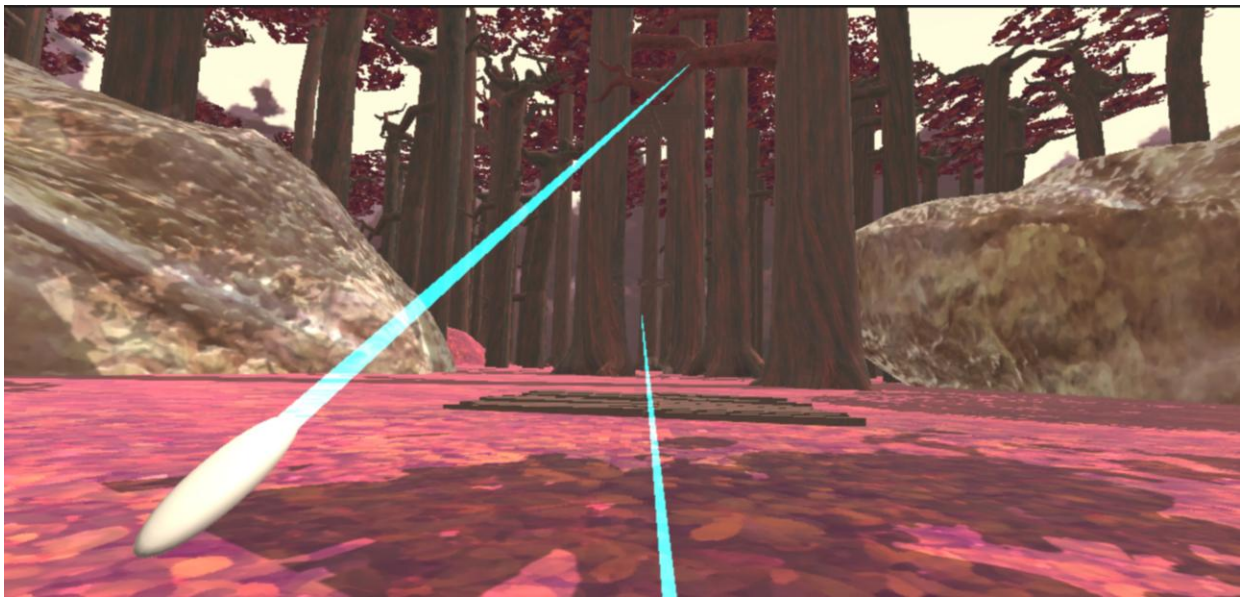


Figure 16: Screenshot of a first person view aiming with the Razer Hydra controllers

Given more development time we would like to continue to improve the ease of use of aiming with motion controllers. Unfortunately the Razer Hydra's motion tracking is not perfectly one to one with the physical movements of the user's hands, so some optimizations we may implement to make up for this might be unnecessary if we add support for more accurate motion controllers. Nonetheless, we need to implement the same level of user feedback we have with the dynamic crosshair with the Xbox controller into the aiming system for motion controls. This could be accomplished by changing the color and intensity of the laser pointer aiming beam. The three things that must be signified to the player in this design are where the player is aiming, if the player is aiming at a target they can grapple to, and if the player is already attached with one of the Lighthooks. One design we came up with to do this was to have the laser pointer beam as it is now to signify where the player is aiming, to have the beam change to a contrasting color when it is aiming at a target it can grapple to, and to turn on a second much shorter beam parallel to the aiming laser when that laser's Lighthook is already grappled to an object. The reasoning behind this design is the same as with the dynamic crosshair; put the signifiers where the player's eyes are looking, which is where they are aiming.

3.4.3 Implementation of Firing Based on Aim

The rule we upheld for firing the hook based on the aim taken with either control method was that if the player fired at the instant they are pointed at a surface they can grapple to, the hook projectile will always hit that surface and cannot miss. In our first implementation we did not prioritize this, and the hook was fired and traveled at a constant velocity that took into account how the player was moving when it was fired. Done this way, hitting targets while moving was very difficult and added the challenge of having to lead targets, which was not the kind of dexterity challenge we wanted for the game. We then increased the projectile's velocity until leading the target was nearly negligible, but ultimately we designed the path of the projectile to be pre-calculated at the instant it is fired to hit whatever object was in the player's aim. So if the player had the correct aim, it was impossible for the projectile to miss.

3.5 Dynamic Environments

Dynamic environments are one of the key features of the game, implemented to enhance the core grappling mechanic and to add challenge. Each of the dynamic environments target a specific area of the grappling ability that either changes the way that area is utilized or extends it to serve a new purpose. The game features four distinct dynamic environments: stretchy vines, fog walls, wind zones, and swinging platforms. These four different environments were chosen because they are relatively intuitive to use for beginners. In addition, some of the dynamic environments form pairs, which complement each other and are utilized in the actual game to create unique puzzles and further increase the difficulty by showing the player how the dynamic environments can interact with each other.

3.5.1 Stretchy Vine

Stretchy vines add a slingshot-like mechanic to the game. They first make their appearance at the end of level one in the level transition. The vines enhance the distance the player can travel using grappling techniques. In-game, the vines are represented by a small sphere, which is the point at which the player can attach to the vine. The in-game representation of the vine can be seen in Figure 17. After attaching to the vine with the Lighthook, the player can reel, pulling the vine towards them. Once the vine reaches its maximum tension, it can no longer be reeled in any further. The player can then release the

reel button to fling themselves over a large distance. This entire process takes place over three stages, which represent different states of the stretchy vine: rest, stretched, and released.

Figure 17: Example of Stretchy Vine. The picture on the left shows the Stretchy Vine without the tree leaf covering. The bottom sphere of the vine can be grappled and reeled in to create tension in the rope.

The stretchy vine starts off in rest state. Figure 18 represents a diagram of the initial state of the stretchy vine. In this state, the vine can be seen as inactive. A stretchy vine enters the rest state when its vine length reaches the original starting length of the vine, which is set when a stretchy vine is created. While in this state, the spherical bottom part of the stretchy vine is grapplable.

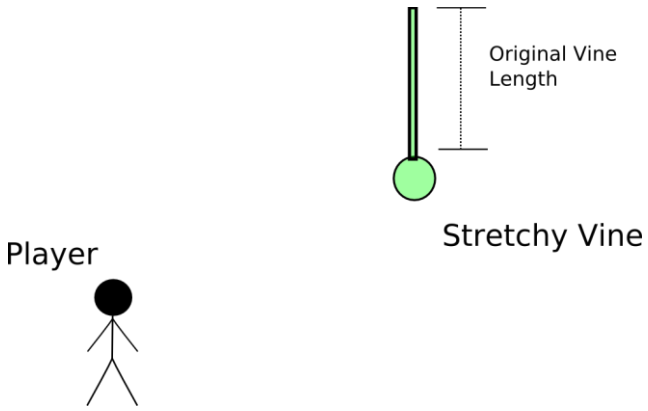


Figure 18: Diagram of Stretchy Vine at rest. The vine has an initial original length.

Once the player grapples onto the stretchy vine, the stretchy vine marks the player's grapple as a connected object, changes state to stretched, which is depicted in Figure 19, and the stretchy vine becomes no longer grapplable. Objects that are marked as connected objects move with the end of the vine. This overrides the normal functionality of the hook, which is to remain stationary once an object is hit. In this state, the player's reel is also overridden. Normally, reeling in pulls the player towards the grapple. In this case, however, the overridden reel functionality now pulls the vine towards the player. A tension force is placed on the vine and causes the vine to become displaced. The vine can be displaced up to a specified max distance or until it reaches a certain point in front of the player. Once one or both of these limiters are hit, the vine cannot be displaced any further.

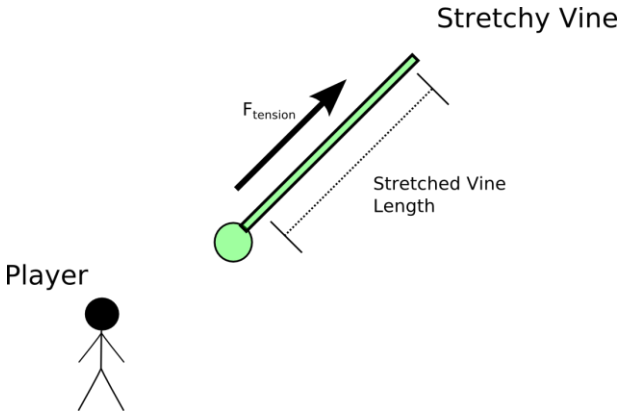


Figure 19: Diagram of Stretchy Vine when player is grappled. The vine is then displaced, creating a tension force.

If the player stops reeling on the stretchy vine and the vine is in the stretched state, the vine will move to a released state. Figure 20 shows the release state of the stretchy vine as well as a rough outline of the player's trajectory. The moment the stretchy vine enters the released state the previous tension force applied to the end of the vine is reversed and applied to both the vine end and the player in the opposite direction to create a slingshot effect. The force is applied to both objects because when the force is only applied to the vine end or the player, it creates an unwanted trebuchet-like motion and path of movement. After the player has traveled a certain distance based on the displacement of the vine before release, the grapple attached to the vine automatically disconnects and allows the player to shoot off into the distance. Once the player is disconnected, the vine will slowly automatically revert back to its original length and return to a rested state.

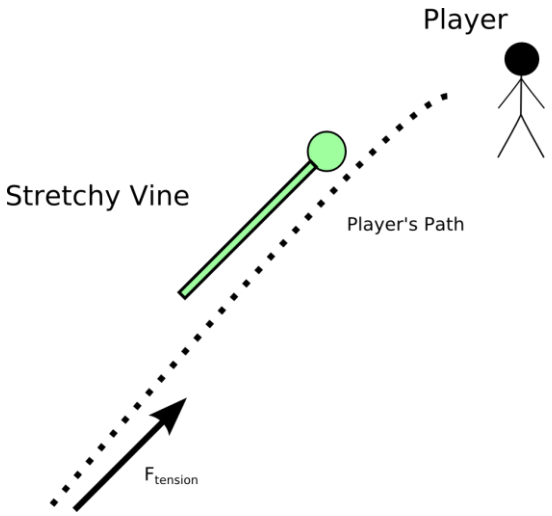


Figure 20: Diagram of Stretchy Vine when the player releases a stretched vine. The player is flung based on the tension force.

3.5.2 Fog Wall

Fog walls appear in level two as a cloud of black fog that obscures the player's vision. They function as an obstacle that the player can freely travel through, but cannot see through. An in-game example of the fog wall is seen in Figure 21, where it obstructs the player's view of what lies beyond. The first occurrence of a fog wall teaches the player this by using a stretchy vine to automatically fling them through. From this point the player learns that they can indeed go through the fog. From that point on, fog walls are used to challenge the player's reaction

speed. As they soar through the air and into a fog wall, they are temporarily blinded. When they emerge through the other side of the fog wall, they must quickly assess their situation and look for something to grab onto in order to avoid falling to their death.

Figure 21: Example of Fog Wall. The black fog obscures the player's vision of what lies behind it.

3.5.3 Wind Zone

Wind zones first appear in level two and are visually represented by areas filled with leaves being blown in a particular direction. Figure 22 depicts an in-game representation of a wind zone that the player has to navigate through as part of a puzzle. They affect the environment by pushing anything within the zone in a predetermined direction. This is used in the game to lift the player upwards, suspend them, suspend platforms, and throw the player to their death. Wind zones are helpful in creating areas where the player has time to stop for a moment and assess their situation, but is unable to walk around. This encourages the player to use their Lighthook in clever ways. For example, in one level the player must jump into a large wind zone in order to progress. However, they are then stuck floating in the zone. To get out, they must attach to branches nearby and slowly pull themselves around trees to exit the wind zone and progress further through the level. Wind zones also allow us to impart another force

on the player besides gravity, causing them to consider how differently they will move when they swing through a wind zone.

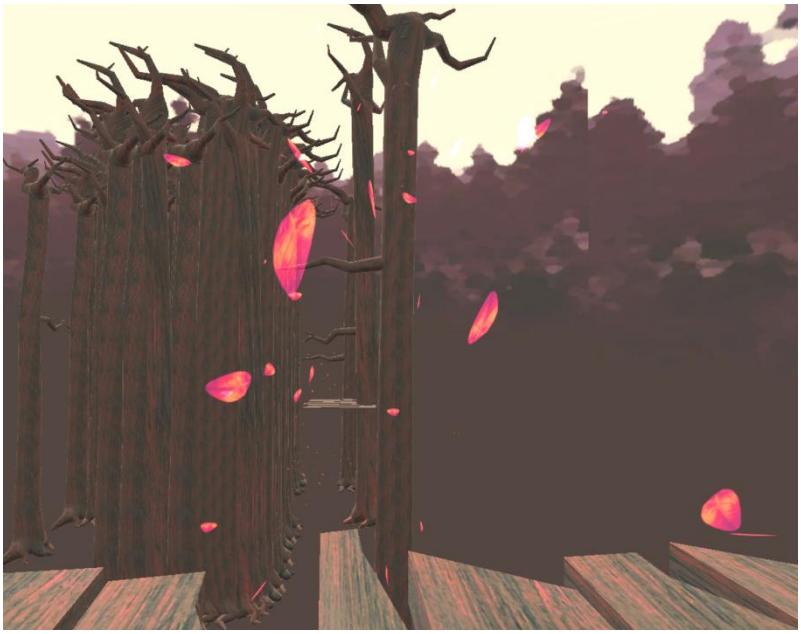


Figure 22: Example of a Wind Zone. The blowing leaves represent the area and direction of the wind force.

3.5.4 Swinging Platform

Swinging platforms are dynamic environment pieces that appear in the third level of *Hikari Hook*. Their primary feature is a platform similar to the ones the player is used to interacting with. However, instead of being attached to a tree trunk like the normal platforms, the swinging platforms are suspended by four ropes. An example of an in-game swinging platform can be seen in Figure 23. One way the player can interact with the platform is to connect one hook to the platform while standing on it, then fire their other hook at another point further away. If they reel in this second hook, they will slowly bring themselves and the platform up towards that point, generating potential energy. Releasing this second hook will then cause the platform to start swinging like a pendulum, possibly allowing the player to reach parts of the level they could not reach before. The player can also interact with the platform by firing one hook at it from a distance, using their second hook to secure them to their current location. By reeling the first hook, they can remotely pull the swinging platform towards them, allowing them to jump on and make it start swinging.

Figure 23: Example of Swinging Platform. The platform is able to pivot around the anchors of the four ropes holding up the platform, which allows it to swing.

While the hook can attach to the top of the platform, it is not able to attach to the bottom. In fact, if the player attaches to the top and attempts to swing underneath the platform, their rope will be severed by the platform. This was a conscious decision in order to help the player understand the main function of the swinging platform. By doing so, we keep them from getting stuck on the underside of the platform with nothing nearby to grab onto or attempting to use the swinging platform as an extension to their length of rope. We wanted to keep the focus of swinging platforms on puzzles that require the player to make use of both hooks.

3.6 Rotating the Player Camera

One of the features our game employs to deliver a more immersive experience is rotating the character as they swing using the Lighthook. The goal was to adjust the player's camera to simulate how the body would be oriented if it were on a rope swing. While rotating the player is a trivial task, rotating them smoothly and in a way that felt natural required us to answer significant design questions and tackle several challenges. In the following explanations,

the terms yaw, pitch, and roll will be used to describe rotation of the player. Yaw refers to rotation about the character's vertical axis, essentially turning the character left and right. Pitch refers to rotation about the axis that spans from left to right, tilting the character to look up and down. Roll refers to rotation about the axis that spans from front to back, quite literally rolling the character sideways.

3.6.1 Naive Implementations

Our earliest idea for rotating the character involved maintaining the player's head pointed at the location where the hook was attached. While this approach ensures that the pitch and roll of the character are always correct, it fails to adjust the yaw of the camera. We discovered early on that changing the yaw would be important to maintaining a comfortable user experience. It is important that the character continue to face forward as they swing in arcs (Figure 24) or circles (Figure 25) so the player can constantly see the obstacles they are swinging into without having to constantly turn their head to adjust. As a result, an implementation that uses only the player's location relative to the hook is insufficient.

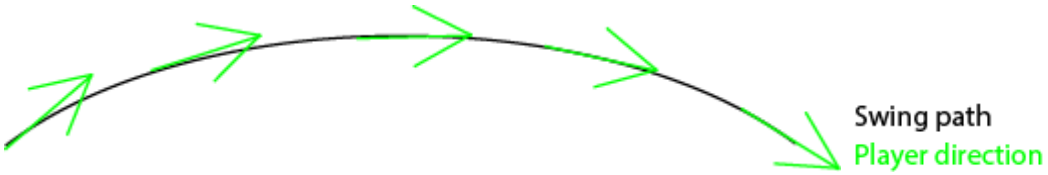


Figure 24: Desired arc-shaped swing behavior

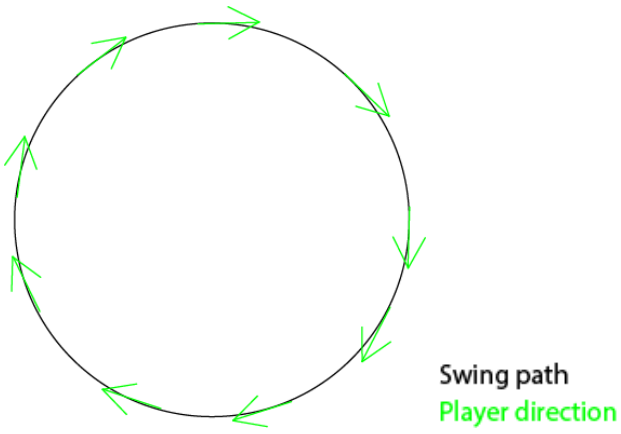


Figure 25: Desired circular swing behavior

3.6.2 Velocity-based Forward

It was clear to us from the arc and circle cases that the driving factor for the yaw of the player would be the direction that they were travelling in. As a result, our next draft for the feature involved rotating the character such that their forward direction was in the direction of their velocity and their head was pointed at the hook point. This was the first implementation that actually made it into the game. Due to limitations of Unity's physics engine, we found that the Rigidbody component of our controller did not always report an accurate velocity when it was connected with a rope. For example, when the character was hanging from a rope at rest, the y component of its velocity was reported as some negative value, instead of 0. This lead to the player being rotated until they were horizontal and facing the ground. To remedy this, we projected the player's velocity onto the plane perpendicular to the vector pointing from the player's position to the hook (Figure 26). This fixes the velocity to directions the player could actually be travelling in, since by hanging from the rope, movement must be perpendicular to the previously described player to hook vector.

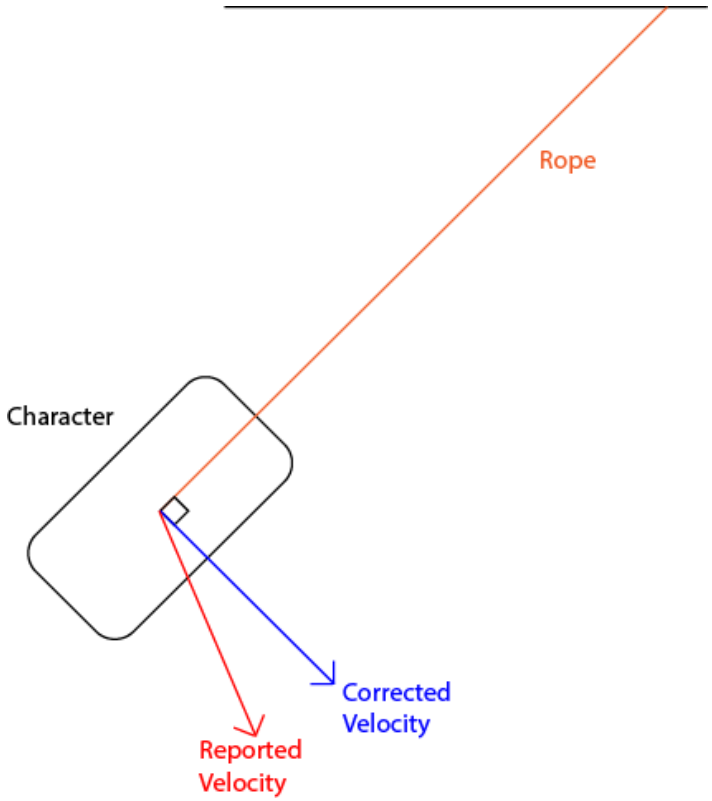


Figure 26: Velocity correction

In testing, we discovered that in most cases, we would need to interpolate between the player's current rotation and the desired rotation in order to prevent the camera from suddenly snapping and disorienting the player. Tweaking the speed at which we interpolated required some testing to strike a balance between a value that was too slow to ever "catch up" to the desired rotation and a value that was too high and disoriented the player with sudden jerks. Interpolating to this velocity-based forward generated the desired behavior for the majority of general use cases, but left many edge cases to be ironed out.

3.6.3 Return Swings

In simple pendulum swings and thin elliptical swings, it is desirable after reaching the apex of a swing for the camera to continue facing the same direction on the return swing, rather than quickly turning 180° and disorienting the player (Figure 27). In order to handle this, we first tried to see whether the direction the player is traveling in is on the front or the back side of the player. If it is the front, then we handled rotation normally with velocity-based forward. If it's the back, we adjusted the yaw of the desired rotation by 180°, so the desired rotation became the direction opposite of the velocity of the player. This means that as the player reaches the apex and then starts falling backward, they will continue to face forward, which is more likely to be the direction the player wants to travel in.

Figure 27: Desired return swing behavior

While this worked for the return swings, it created unwanted behavior in an edge case where the player, coming out of a previous swing slightly rotated, would suddenly pick the wrong direction to swing in, consequently rotating in a confusing fashion (Figure 28).

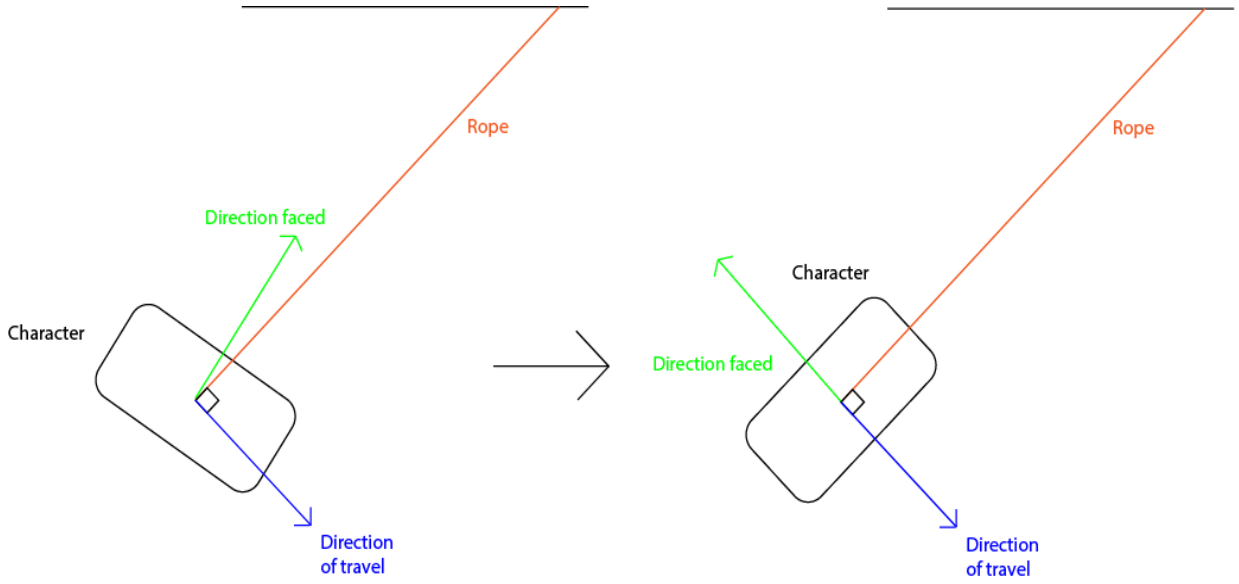


Figure 28: Problem swing: The character rotates to swing backwards

In order to cover this edge case, we made use of another piece of information: the direction in which the player's head was pointing. A distinguishing trait between our desired behavior in a simple pendulum and the unwanted behavior in the edge case is whether the player's head is pointing towards the hook point or away from the hook point (Figure 29). Taking this into account, we can be more selective about when to have the character swing backwards.

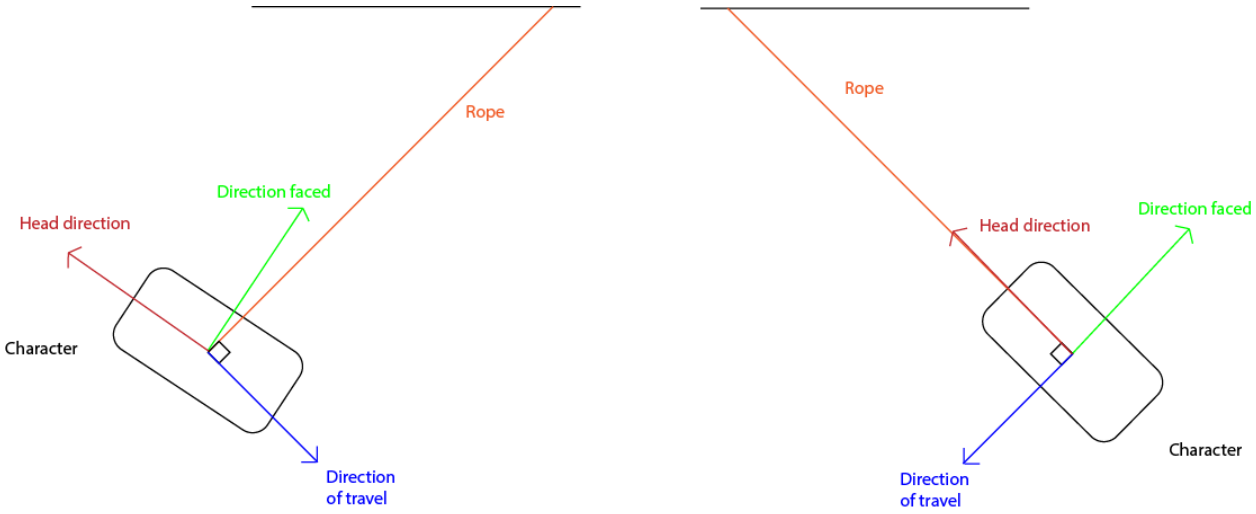


Figure 29: Difference in head direction between problem case where player should swing forwards and normal case where player should swing backwards

3.6.4 The Dead Zone

With the exception of swings that are mostly circular, the vast majority of swings involve some apex of the swing where the speed of the player reaches a minimum, then starts to pick up as the player swings back down towards the equilibrium position. In some swings, the speed of the player as they reach the apex becomes so small that velocity becomes unreliable for determining what the player's forward vector should be. As a result, player rotation needs to be handled using a different method when they are traveling at low speeds. We referred to these low speed intervals as being in "the dead zone".

Our first method for handling players in the dead zone would slowly set them upright. This idea stemmed from the case where the pendulum comes to rest. In this case, we wanted to make sure the player was upright and able to look around normally. Furthermore, we expected the player to be making their next grappling hook shot at the apex of swings, i.e. in the dead zone. As a result, we considered that it might be helpful for the player to be upright when they need to make these shots. After testing, we quickly found that this behavior actually harms the player's ability to aim instead of helping it. Oftentimes, the player was lining up their shot as they entered the dead zone, meaning that as we uprighted them, we also ruined the alignment of their shot. Furthermore, swinging in and out of the dead zone felt very unnatural; the player would be uprighted briefly, then rotate back into having their forward vector follow their velocity vector.

The next attempt rotated the player's pitch and roll to keep their head pointed in the direction of the hook as we do outside of the dead zone, but held the yaw of the player constant during their entire stay in the dead zone (Figure 30). While this works well for simple back and forth swings, we found that it felt unnatural in elliptical swings. This is due to the player noticing that at some arbitrary cut-off point, their yaw stops adjusting for a brief period of time, then starts to adjust itself again. It was a very obvious transition to the player, and we wanted behavior that would feel more natural.

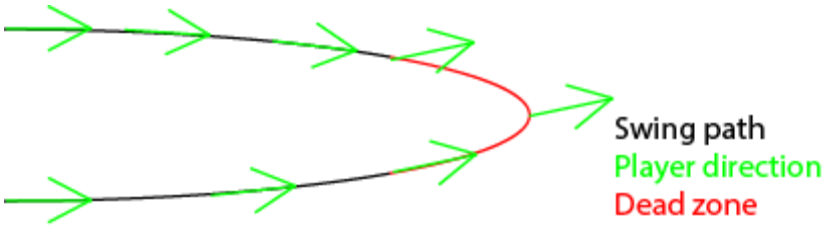


Figure 30: Early implementation of dead zone rotation

Our solution to handling the camera in the dead zone uses an average of our forward vectors outside of the dead zone to estimate what forward should be at the very apex of the swing. As the player enters the dead zone, we record the vector they are facing, then start to rotate them towards the estimated forward. When they reach the apex of the swing, which we can detect by keeping track of the rate of change in speed, we mirror the vector we recorded upon entering the dead zone across the plane containing the hook point, the player, and a point directly above the player. We then rotate the character towards this mirrored direction as they leave the dead zone (Figure 31). This creates a smooth transition in the dead zone from the rotation they entered with to a rotation that is very close to the one that will be generated once the player is outside the dead zone. This removes the need for sudden jerks, camera snapping, or movement that feels unnatural in the dead zone.

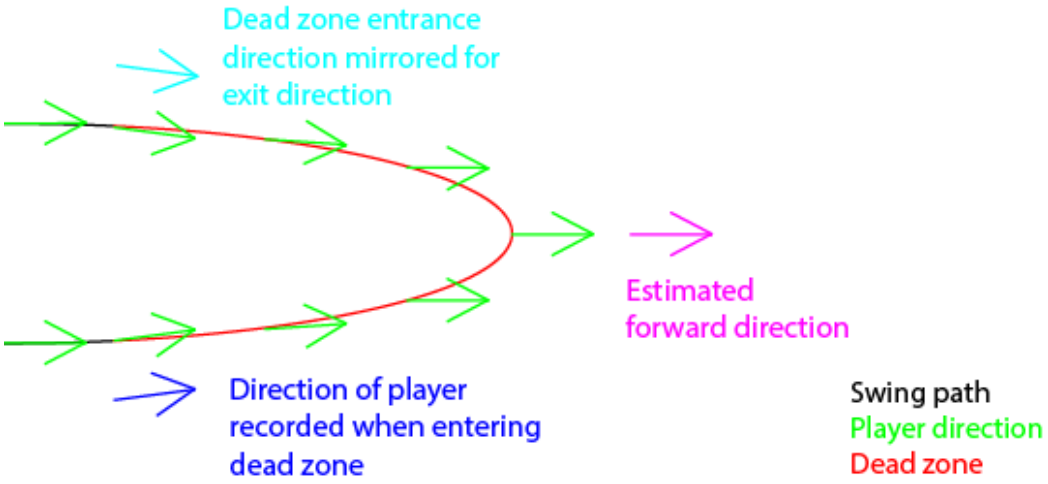


Figure 31: Final implementation of dead zone rotation

3.6.5 Circular Motion

As previously described, circular swings will continuously adjust the player's rotation in order to keep them facing forward, tangent to the circle of their swing. The nature of these swings also means that the player's speed does not oscillate. Rather than moving in and out of the dead zone, the player will be outside the dead zone for most of the swing until they slow down enough to enter the dead zone permanently. The previous method described for handling dead zone rotation does not work well for this type of dead zone. Since the player's forward was constantly turning in a circle while they were outside the dead zone, the average of these vectors is an arbitrary direction that holds little relevance. As soon as the player enters the dead zone in a circular swing, they will suddenly start rotating towards this arbitrary direction. As a result, we needed to find a way to reliably detect circular swings and determine an appropriate method of rotating the character once they entered the dead zone.

We were able to use the fact that circular swings stay outside of the dead zone for significantly longer amounts of time than simple pendulum swings or thin elliptical swings. By keeping track of the time spent in a swing before the player sufficiently slows down to enter the dead zone and designating a cut-off for circular swings, we can handle rotating the character differently in the dead zones. We settled on just maintaining the player's current yaw and adjusting their pitch and roll to point their head at the hook point. This is because we expect this case to occur very infrequently, and this is a simple way to handle the case that is not very jarring to the player.

3.7 Debug Tools

Most of the debugging in our game revolved around studying the player's movement in real time to be able to identify and quantify undesirable or uncomfortable player movement. A single developer testing their own code in a head mounted display like the Oculus Rift is cut off from seeing the real world and thus unable to see any real time logs or external information about what is happening in the game's systems.

First, we needed a way to replicate printing to a real time log as the game is running. Using the same rendering technique as the player's crosshair, where the plane of two-

dimensional content is rendered separately in three-dimensional space very far away from the player and then placed as a layer on top of the rendered three-dimensional virtual environment, we made a heads up display with a window for showing text from the log, seen at the center of Figure 32 under the title Debug Window. The first iteration of this debug log was designed after the log in Unity's Debug class, where logging text is a static function callable from anywhere in the game's code that prints the text to the top of the log and pushes old lines down as new ones are printed. However, printing out any value in real time would quickly fill and overflow the screen with printed lines pushing down old printed lines of values. So we changed the design of this log to be different from the normal Unity debug log and gave ourselves the option to overwrite printed lines instead of pushing them down in the log. Along with this we could set how long that text should be displayed before disappearing, so that after the wearer of the HMD has time to read the text, it will go away so as to stop obstructing their vision.

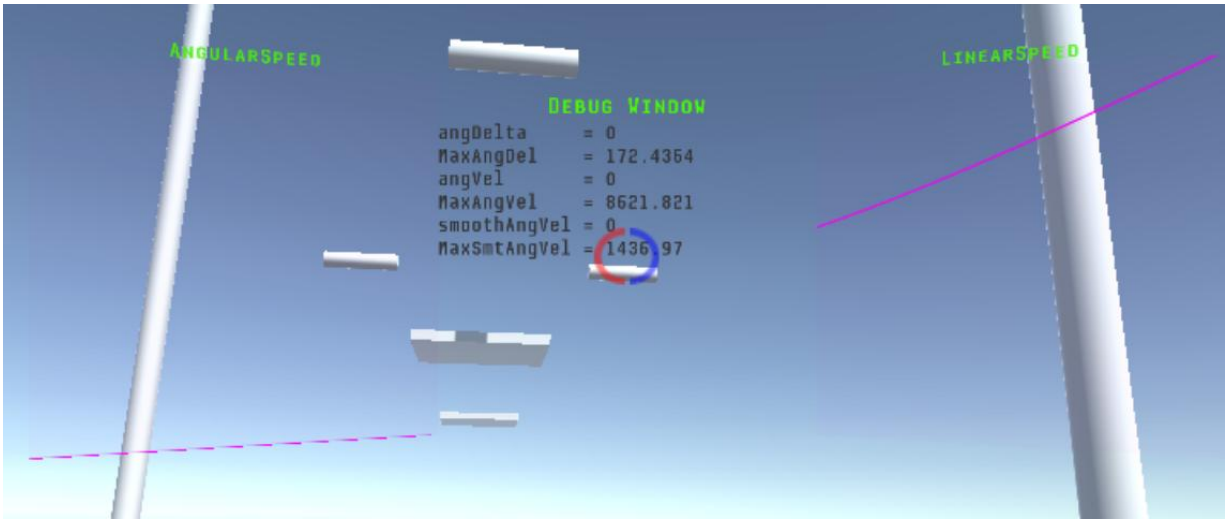


Figure 32: First person perspective of the Debug HUD seen in the Oculus Rift

Being able to show text to a developer wearing an HMD as the game is running was useful, however we wanted to be able to see the change in values over time to better understand the player's movement. We particularly wanted to find cases that caused jerks in angular and linear speed which would be uncomfortable to the player. To do this, we created a graph system that could place graphs on the HUD and have values pushed to it to make the

graph scroll. Because these graphs were to be viewed on the Oculus Rift's low resolution display, we did not focus on displaying detailed scales and labeled axes. Instead we put time into making the graphs bold and easy to read with smooth lines and autoscaling to best display whatever range of data is logged, as seen at the left and right of Figure 32. This was sufficient to analyze what we were interested in, trends and changes in data.

3.8 Replay System

One of the major features of the game is the replay system. The replay system was developed alongside the game with the idea of being able to provide assistance to players who are stuck on particular challenges, as well as to introduce a pseudo-multiplayer system into the game. The replay system is composed of two parts: the client and server. The idea behind the replay system is to provide replays to the player of their past actions or actions of other players in the form of a shadow player. In the game, the shadow player is made up of black smoke. This was to both save time on having to produce new assets for the character as well as add to the sense of mystery. Figure 33 depicts how the shadow player would look in-game. The player would then be able to watch and follow the shadow player to either learn from their mistakes or figure out how to solve the puzzle.

Figure 33: Shadow Player in the game that reproduces player actions.

3.8.1 Client

The client portion of the replay system is composed of a simple script. The script encapsulates each of the actions the player can take into five main types: Move, Shoot, Environment, Hand, and Hook. Each of these actions represents a movement that could be performed by the player or the dynamic environment. **Move** encapsulates the location and rotation of the player at any specific point, **Shoot** encapsulates a Lighthook fire action, **Environment** contains the location and rotation of dynamic environments in the scene that the player interacted with, **Hand** represents the location and rotation of the hands when dealing with motion controllers, and **Hook** represents the location of the hook and what it was attached to. Each of these actions was recorded on a set fixed interval by the script. The default recording rate (frames per second) was 60, the highest possible amount. However, this could be decreased at the cost of replayability to improve performance.

The most complex portion of the recording system was handling the dynamic environments. Dynamic environments are special objects which the player can interact with to solve the puzzle. Because of this interaction, dynamic environments and the way they are

manipulated can be essential to solving the puzzle. This meant that dynamic environments also needed to be recorded for a comprehensive replay. In order to accomplish this in a generic and extendable way, dynamic environment elements that needed to be recorded were tagged with “Dynamic Environment”. This let the replay system know which objects in the scene needed to be observed. It would then record the positions and rotations of each of the objects tagged. During a replay, the system would then create a shadow of the object and replay the movements of the object and its interaction with the player character from the recording. The shadow of the objects were created by making copies of the in-scene game objects and modifying the opacity of the object to differentiate it from the real one that the recording player can interact with.

3.8.2 Server

The server portion of the replay system consists of a Flask server. A Flask server is a server which runs on Python using the Flask framework to listen, handle, and respond to web requests (Ronacher). Since the server was only responsible for transferring data between two machines and temporarily holding it, Flask was chosen due to its ease of use and the extendibility of the framework. Although not the most efficient, it is one of the easiest types of servers to set up, which made it a prime choice for the simple tasks that needed to be performed. Including a database with the server was decided against due to the extra steps that it would require to set up. In addition, the server portion of the replay system is an optional component and is not required for functional gameplay. For these reasons along with project time constraints, the use of a database was left out. However, if a database was needed in the future, the Flask framework provides an easy way to integrate one.

The Flask server API has two main uses: saving recordings and retrieving recordings. Figure 34 shows a very simple diagram of the Flask server structure and provided API. All post requests sent to the server attempt to save a serialized list of recorded actions. The recordings are saved in two different key based hashmaps. The first hashmap saves the recording by a unique identifier for the recording itself. This allows for specific recordings to be pulled down from the server later on. The second and more important hashmap saves the recordings in a list of different recordings sent to a server for a specific checkpoint in the game. This is used when

enough failures for a specific checkpoint occur; a random successful attempt can be pulled from the server and shown to help the player progress. The second use of the server is to retrieve recordings, which is initiated through get requests to the server. As mentioned earlier, there are two types of retrievals. A specific recording can be retrieved by specifying a unique identifier or a random recording can be retrieved for a specific checkpoint by specifying a level checkpoint identifier.

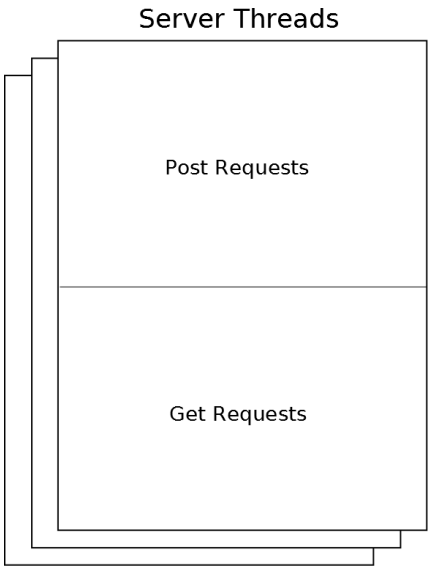


Figure 34: Diagram of the server threads, which allow multiple concurrent requests to be processed simultaneously

3.8.3 Client-Server Communication

The most complex portion of the replay system is the client server communication. A diagram of the client server communication can be seen in Figure 35. It is composed of a static class which manages communication between client and server through threads. Figure 36 depicts the structure between the client and server. Threads were used to manage the IO connections, because Unity is set up with one main thread which handles all of the game's runtime function calls. Blocking on the main Unity thread would cause the entire game to freeze for the duration of the block. This is bad for the player experience because it either creates the need for a loading screen or creates a drop in frames per second, both of which negatively affect gameplay. In order to combat the negative effects, threads were used to

perform asynchronous operations. This allows both operations to run seemingly simultaneously.

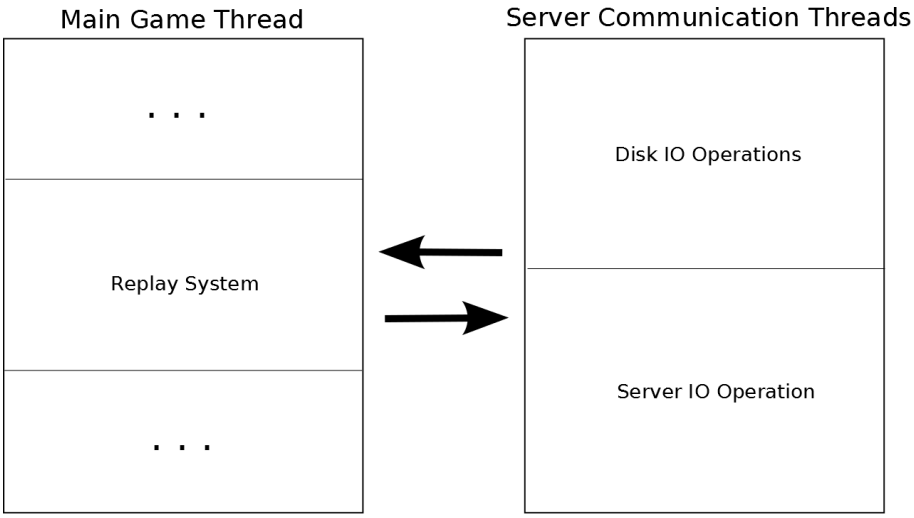


Figure 35: Diagram of the interaction between Unity's main thread and the Server Communication Threads.

Another important part of the client server communication is efficiency. Even though the operations appear to run simultaneously, they actually take turns on the CPU depending on the current workload. This can also severely affect the performance and frame rate of the game. In order to avoid this, the threads are all initialized on play and are kept for the entire duration of the game. All threads were assigned the second lowest level of priority (high niceness value). Since starting and stopping threads requires a lot of resource allocation from the computer, the threads were initialized on program start. A request system was used and implemented with a custom thread-safe queue system to pass processing requests by reference to the worker threads. A custom thread-safe queue system was implemented because C# thread-safe queues are unsupported in Mono 3.5, which is what Unity 5.2 runs on. With the exception of the output result field of the thread, which requires a lock to access, all of the items inside the request object were set to read-only to prevent read-write race conditions.

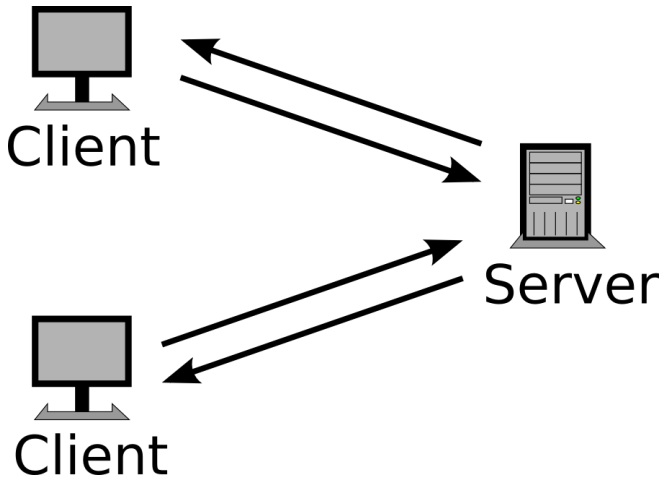


Figure 36: Diagram of the structure of the client server connection of the replay system.

Each of the threads blocks until a new item enters the queue. When a new item enters the queue, each of the threads wakes up and performs the allotted action based on the type of request entered. Once the thread finishes its work, it places the result in the request and releases the lock on the output field, which allows the main thread to then retrieve the result and handle it accordingly. In order to safely handle and exit the threads upon quitting the application, a 30 second default timeout was placed on the various IO operations. This ensured that after 30 seconds, the thread would give up on the IO operation and continue on. In addition, upon receiving the application quit event from the game, the abort flag is set and the threads all release to attempt to process a dummy request, so they can stop blocking and check the abort flag to return and safely exit their operation. Although this does not guarantee the thread to instantly stop, it guarantees that after 30 seconds, the thread will safely abort on its own. This is preferable compared to the alternative, which is to force abort the thread and potentially corrupt a read or write on a file IO operation as well as leave open file handlers.

3.9 Level Transition

Level transitions are important when working with multiple constituent scenes that have to be stitched together to create the entire game. One route instead of doing level transitions is to place all scene objects into one massive scene. However, this causes problems on many fronts. The first problem is that due to the binary nature of Unity scene files, they cannot be merged with the version control software Git. Instead, when a merge conflict occurs

on a scene, only the changes on one side of the merge can be kept while the others are thrown away. This means that any work done on that side of the merge must be redone. Although prefabs can remove some of these issues, the main underlying problem still exists and creates a huge bottleneck in the project: only one person can work on a scene at a time. Alternatively this problem can be avoided by creating sub scenes at first and then merging everything at the end to create the mega scene. However, this does not solve the second issue with mega scenes and arguably the most important one when dealing with virtual reality games: performance. Performance takes a big hit by having all the objects in the scene at the same time. This issue becomes increasingly apparent the more levels there are. In order to avoid these issues, the level transition route was chosen.

Unity has two types of level transitions available: streaming and loading. The streaming level transition asynchronously imports all scene objects from one scene into another when called. This is done in another thread in order avoid disrupting the main thread's execution. The second type of level transition is the traditional loading bar type load, which stops the execution of the main thread and essentially swaps scenes to the new scene. Although stopping gameplay to load levels is expected for traditional games, it becomes more of an issue when using virtual reality systems. In order to combat this, streaming asynchronous based loading was used. In this way, the current and next levels are seemingly connected with little to no interruption in gameplay. However, streaming levels has problems of its own that needed to be fixed in order to create a seamless transition.

The first problem with asynchronous loading in Unity is object placement. When objects are loaded in asynchronously from a different scene, they are placed at the same global coordinates as their placement in the global space of their original scene. This can create issues because the objects in the scene might not be reachable from the end of the previous scene, or worse, they might have been loaded directly on top of the previous scene. One way to combat this is to place the levels in the scene at the exact coordinates where you want the level transitions to occur, so the objects line up nicely. This causes other issues, namely requiring scenes to be realigned if they are modified or moved around. In addition, having the levels perfectly line up means that the next level needs to be loaded before the player reaches a point

where they are able to see it. By doing this, some of the benefits in performance are lost, because at times there are two completely active scenes in the game.

Instead of taking this approach, a third combination approach was taken by combining the asynchronous load with a pseudo load screen technique. This technique employed deception and hiding the player's vision in order to avoid having two scenes from ever being loaded into the game at the same time. This can be seen in Figure 37, where particle effects were used in the areas where the player needed to transition between different scenes. The effects clouded the player's vision while the old scene was deleted and the new scene was loaded. While this was happening the player was suspended in a limbo state. During this state, the player's movement was artificially restricted to make it appear like the player was moving when in fact they were not. This was achieved through heavy use of SFX and different particle effect settings. This allowed the player to interact with the game, although in a limited way, while not interrupting their gameplay.

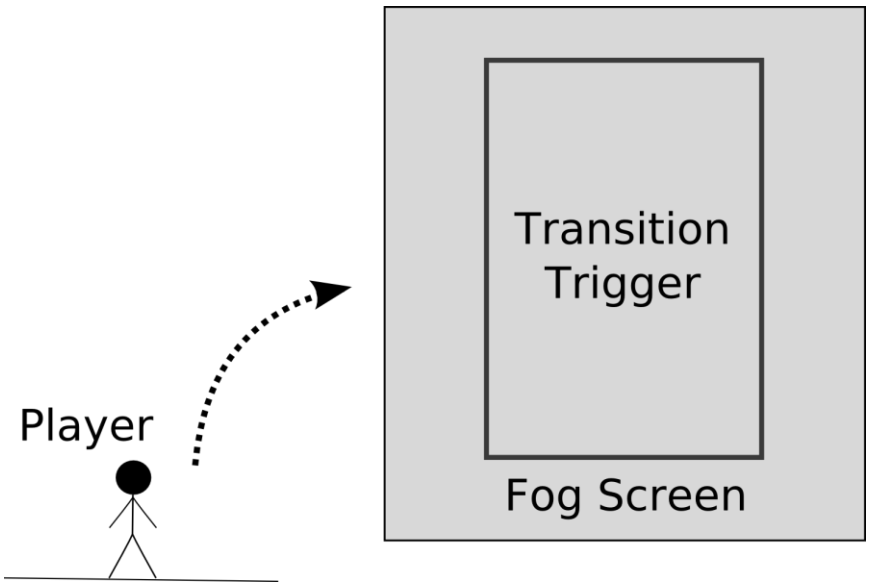


Figure 37: Diagram of level transition. Fog Screen prevents player from noticing outside changes. Trigger correctly places the player and loads new level.

Artificially restricting the player's movement does not solve all the problems of the asynchronous load. The problem still exists where loading in objects asynchronously can pull in objects at locations that do not provide a nice transition between the scenes. A second

technique was used, which correctly positioned the player upon entering a new scene. In order to achieve the desired relocation of the player, the transitions between the different levels of the game were performed by shooting off the player into the distance or involved the player flying through the air on their own to reach a special trigger zone. Having the player move while the level transition takes place is important as it allows the relocation of the player to be masked. When the player enters the trigger zone, the trigger zone calculates the current velocity of the player. Using the player's current velocity and a known ending location, the start of the next level, it calculates the spot that the player needs to be to land at the start of the next level. It then proceeds to change the location of both the zone and the player to the new calculated location, so that when the player exits the limbo state they naturally move to the start of the next level with the previous velocity they entered the trigger with. Normally, moving the player unexpectedly can cause motion sickness. However, the trigger zone that the player enters obscures the player's vision through the use of particle effects. In turn, the particle effects also mask the relocation done to the player. The particle effect and player move together as one unit, which makes it impossible for the player to perceive the motion. Figure 38 shows an in-game example of the implemented level transitions in *Hikari Hook*.

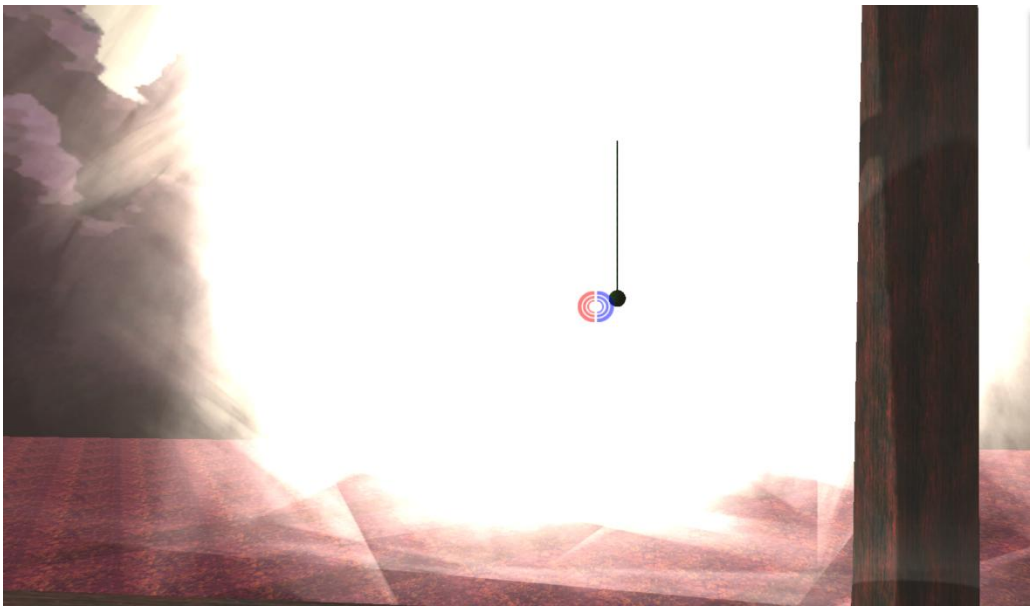


Figure 38: Shows the level transition from level 2 to level 3 in the game. The stretchy vine depicted in the scene slingshots the player into the level transition trigger.

3.10 Editor Scripts

Editor scripts made certain tasks for the development of the game easier. There were two main editor scripts used during the project, which assisted in importing finished assets into the game. These scripts were designed for convenience, saving time that would have been spent on monotonous tasks.

During the level design process, the levels were initially created by gray-boxing the levels with primitive shapes that were combined to resemble the final assets. However, the gray-boxed levels needed to be updated to include the new assets when they were finished. The first tool assisted in replacing the old gray-boxed items with the completed assets. The script worked like a mass replace and allowed the user to specify a type of asset that they wanted to replace with another or several others. The script would find all instances of the specified asset and replace them with the new ones, copying the parent, rotation, and transformation components of the previous object. In addition, if there were several new assets it would randomly choose one of the assets to replace the old one.

The second editor script we created was used to help add the background scenery to the levels. The tool is shown in Figure 39, where it acted as a paint brush tool and allowed the user to paint custom objects into the scene over a large area. This was mainly used for painting in the surrounding background trees. In order to avoid potential overlap, an algorithm was used to create a pseudo random distribution of the trees. The algorithm worked by partitioning the highlighted paint area into different zones. For each of the partition zones, a location in the zone was randomly chosen where an object would be placed. Although trees could still be randomly placed very close to each other, it removed the possibility for an exact overlap and maintained a semi-random distribution of trees like one would expect to see in a forest setting. The finer details could then be manually added to each of the levels.

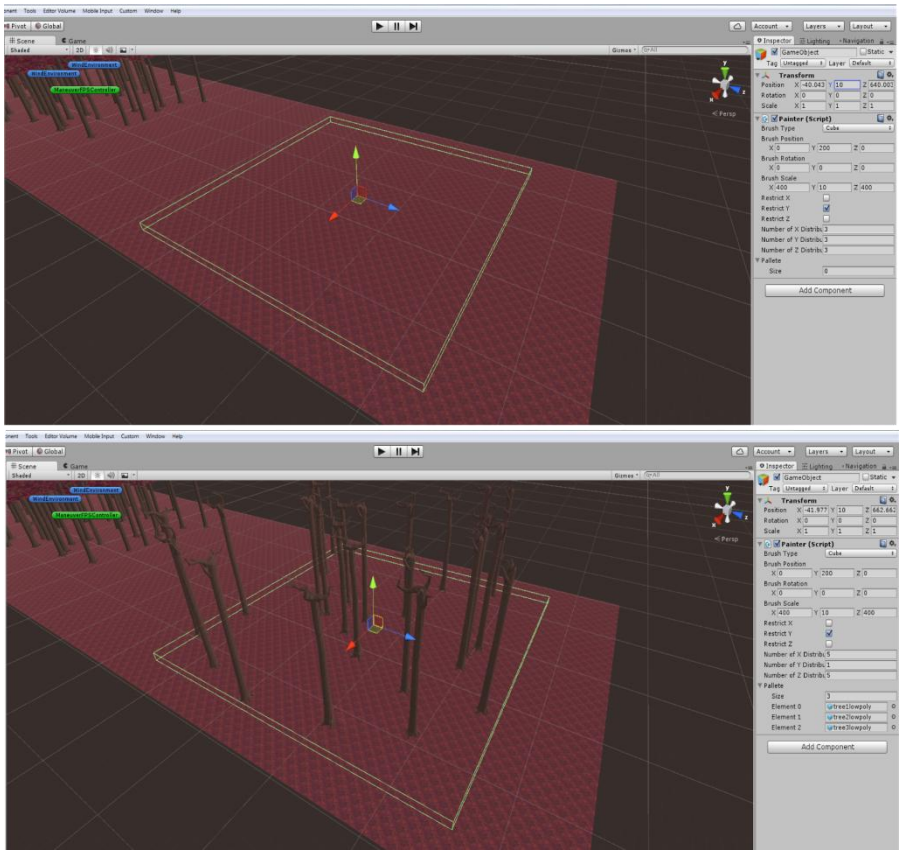


Figure 39: Paintbrush editor tool that allows objects to be painted. In this case, the tool is painting trees.

3.11 Testing

Testing of the project was split into two main subparts: two-dimensional testing and three-dimensional testing. Our process followed the approach of constraining the number of variables as much as possible and slowly unconstraining them as desired behaviors for the set of constrained variables were reached. This resulted in the two aforementioned main categories of testing.

3.11.1 Two-dimensional Testing

A separate scene was created in Unity where movement along the z-axis was restricted. This created a scene that was essentially two-dimensional, which could be used to test the character's movements. For testing purposes, the main camera was swapped from the first person view on the character to an orthogonal camera, which displayed the entire scene view,

making it easier to observe the behavior of swings. This allowed the properties of the swing to be adjusted one at a time until the ideal swing was achieved for two-dimensional space.

After completing a basic implementation of the rope swing and exposing various values to modify the speed, trajectory, and arc of the swing, a genetic algorithm was created to attempt to learn the parameters required for a smooth fluid swing (Kumar et al.).

The genetic algorithm that was implemented followed a simple approach of taking the three highest scoring parameter sets based on a heuristic function. The top three sets were then mutated using a random number generator combined with a Gaussian curve, as well as randomly combining or swapping parameters of the other top sets to create new sets. There were three levels of mutations that were applied depending on the magnitude of the parameter being mutated. Values under 10.0 were mutated using a max standard deviation of 0.2, while values between 10.0 and 100.0 were mutated using a standard deviation of 5.0 and values over 100 were mutated using a standard deviation of 20.0.

A simple heuristic was employed to track the performance of the sets of values. The heuristic function measured the deviation from the expected start and end location of the swing as well as any unexpected rotational movements caused by the swing.

Although the heuristic algorithm worked partially, most of the success with the algorithm was with only two dimensions of movement, where it was easy to predict the expected result. As the movements became more complicated, it became harder to define expected behaviors, and instead, a manual approach of setting values took over, because it was faster and easier to maintain.

3.11.2 Three-dimensional Testing

The majority of the testing done during the project was three-dimensional testing. This testing included tests conducted within a virtual environment using the Oculus Rift DK2 as well as using Unity's scene view to gather data. All the initial testing was done manually and fine-tuning was achieved through a series of playtests done by various students at Osaka University. Altogether, three major rounds of playtesting were conducted. These three rounds were not exclusive and a lot of playtesting was done continuously during the development of the game.

The first playtest conducted was a very simple test aimed at determining the motion sickness effects of swinging through a virtual environment on various people with differing levels of gaming and Oculus Rift familiarity. There were only five participants in the first play test. During the playtest, the participants played two different versions of the game: one was just the standard swinging game without rotational camera movement to mimic gravitational forces on a swinging rope, and the second version, which had rotational camera movement. After the playthrough, the participants were asked to explain their thoughts on the game and what, if anything made them sick. The data gathered from this playtest influenced the design choices of further increasing the ease with which the player can grapple onto objects as well as orienting themselves in their environments. Much of the feedback garnered from the experiment was put to use in creating the tutorial level.

The second round of testing took place after the tutorial level was finished. This was a smaller round of testing and only included students from Takemura lab. In this second round, the testers played through the tutorial level. The main goal of the test was to determine how intuitive the game was to play after having a brief tutorial to help guide the player. During this round of testing, several more improvements were suggested to the game towards player navigation and orientation as well as the rope rotation mechanic in the game. Feedback from the tutorial level was used to improve the level designs for the main levels of the game.

The final round of testing was done after the completion of levels one, two, and three. The last test positioned the players at the start of level one and was conducted in order to obtain information about the difficulty and time requirements for each of the levels. It featured a much larger pool of around fifteen people who had varying levels of gaming and virtual reality experience. Levels of completion from this playtest varied greatly. In the time allotted for the playtest, only one person managed to fully complete the game, while several others managed to get to level two. Feedback from this round of testing focused mainly on the playability of the game and how much motion sickness was caused from prolonged play.

3.11.3 Replay System

The replay system of the game had an interesting role in the overall testing of the game. Although not its intended purpose, the replay system provided an easy means to reproduce

bugs that were found during testing. The replay system provides the ability to record actions performed by the player character and then dump the actions into a recording file. This file can then be processed by another game session to reproduce the same actions that the other player performed. This was very useful for being able to reliably reproduce bugs. It provided a means to redo the exact same steps that were performed if a bug was encountered. This made the testing process a lot faster, because instead of wasting time to try and reproduce the bug, the bug could be reproduced on demand. In addition to being able to reliably reproduce bugs, the replay system also allowed bugs to be sharable. When filing a bug report on the system, the recording file could be included along with the issue. By doing this, a lot of time was saved from having to write up steps to reproduce the bug or detail the behavior since it could be easily reproduced.

3.11.4 CPU Optimization

As outlined throughout this methods section, our game had a large number of technical features and systems running concurrently. As a result, while coding our game, we kept several best practices in mind to minimize CPU overhead. In a managed language like C#, most overhead, outside of poorly written algorithms, comes from having too many memory allocations and deallocations. To minimize this we upheld the practice of not allocating any memory or using the “new” keyword in any loops or areas of code that are regularly called.

Another very common source of CPU usage in Unity is calling the method “GetComponent” on a Monobehavior. This function finds a component of a given class that is also operating on the same GameObject as the Monobehavior that the function is called off of. This lookup involves lots of type checking and unnecessary CPU usage, when instead of finding the reference to a component every time it is needed, the reference can be stored in whatever class needs it. We followed this practice in all of our code.

However optimized we were able to make our CPU usage with these best practices, we found that the real bottleneck for performance in our game was on the GPU, where performance was most related to the art and visuals in the game.

3.12 Art Background / Art References

The nature of our game changed several times in the weeks prior to development. Each of these conceptual iterations changed the proposed artistic style of the game immensely.

3.12.1 Bending game

The first of our concepts began in the early spring. Originally, our plan was to create a game focused around throwing objects. From an asset perspective, this meant that most of the assets would represent the player's hands and the rocks they throw. We would have needed to generate many different rock assets, as we were planning on making the rocks destructible. While environment assets would be important, they would be ultimately secondary to the elements the player would be interacting with. The environment would be in a static location such as a sporting arena, with the player acting in the center.

Since we drew a lot of our core mechanics for this game idea from the TV series *Avatar: The Last Airbender*, we also intended on designing the playable area to visually mimic the Ultimate Bending sporting arena from the series. For our designs, we also considered incorporating features from the Blitzball stadiums present in the 2001 video game *Final Fantasy X*. Both designs featured a relatively small arena disconnected from the stands by a water moat. However, as technical limitations caused us to discard the gameplay mechanic of controlling rocks, these ideas too, were dropped as we moved onto other game concepts.

3.12.2 Desert Rider

Shortly after the Bending game concept was retired due to technical limitations, we worked on drafting several new ideas. We considered designing a game built around first-person vehicular combat, drawing inspiration from the 2015 film *Mad Max: Fury Road*. During the planning stages of this concept, we drafted an early list of key assets that would be necessary for the project, much like our previous concept. With only one artist, it would be time-consuming to make several animated characters in the game. We considered generating a model for the player character so the player could observe their arms. Our plans to reduce the asset load involved giving the player control over an open-topped motorcycle, while enemies would control closed vehicles that concealed their bodies. The larger enemies would drive

armored cars, while the lighter enemies would drive closed motorcycles. For design ideas, we looked at the Mototank from the PlayStation 1 game *BattleTanx: Global Assault*.

The environment for this concept would be simple; it would be set in a barren desert with small amounts of foliage and some trees. To keep the game from looking lackluster, we planned to make the vehicles more brightly colored and vibrant compared to the environment.

3.12.3 Color Game

After arriving in Japan, our first few days were spent brainstorming new game ideas. The "color game" concept involved the player starting in a colorless world, where color had drained from the environment. As the player progressed through the game, they would unlock colors that would be added to the environment.

This meant that each asset in the game would require a skinless version as well as a skin for each of the colors. We considered trying to implement a method for spraying color similar to the color mechanics of *Splatoon*, avoiding the need to create alternate skins for objects. However, we ended up raising several concerns about the art direction of this game idea. This color concept would effectively limit our ability to create a specific look to our levels. Since color would be applied dynamically, we would have to make sure the levels looked appealing no matter what combinations of colors were present. As a result of this concept, the game would look very lifeless from the start without any color, almost artificial. Furthermore, we would be losing control of much of the color direction by giving it to the player. We would also be eschewing the use of Unity's colored lights. These downsides led us to ultimately give up on the idea and move on to other concepts. Additionally, with only one artist on the team, we were concerned about making a more artistic-focused game.

3.12.4 Shadow of the Colossus

The *Shadow of the Colossus*-inspired game was our last prototype pitch before we settled on a working idea. Under this concept, a large, singular, highly detailed character would act as both the stage environment and the only enemy. Due of the sheer amount of work that would need to be done on the boss character, the environment would be relatively sparse,

consisting of a desert or an open field. This would give the player a lot of room to work with, and would help make sure the boss was visible at all times.

The biggest artistic challenge of this idea was the design of the boss-character. We decided that the boss would be static, much like the final boss in Shadow of the Colossus. It would resemble a moving structure that the player could climb. The original design of the boss character was a humanoid castle with ancient/medieval Persian influences. The character would have a red and gold hue. We had planned to add several sections inside of the boss where the player would pass through chambers embossed with open-fire pits, gold coin piles, and stacks of jewels. Other discarded designs for the boss character included a massive snake built up of multiple different sections and a massive walking humanoid.

In order to provide challenging gameplay, the boss needed to be designed in a way such that it was climbable, but with scaling levels of difficulty. In order to accommodate this, certain parts of the character would have to be static, while others would have to be animated. Instead of rigging the character to a single skeleton, this new idea meant we would have to divide the boss into several different assets. Additionally, this meant that only critical assets such as the hands and head would have to be animated.

Ultimately, the idea was abandoned. We felt that our ideas were too similar to Shadow of the Colossus, and that continuing with the plan would hamper the originality of our game.

3.12.5 Current Design

The current artistic design of our game evolved from our previous concept, the *Shadow of the Colossus*-inspired game. However, the idea of the boss-type enemy was removed in favor of a game centered more on the environment. We switched the scene from an open landscape to a thicker forest. From here, our visual design for the game went through several different iterations.

After establishing the general mechanics of our concept, we decided to focus on the color scheme of our game. While we had been considering a yellow and green color scheme with the Shadow of the Colossus concept, the movement to the forest-centered game led us to reconsider in order to make our environment more visually appealing.

One of the most difficult details to iron out was the design of the trees. Since the trees present in our game were both aesthetically and functionally crucial to game, we went through a number of different designs, shapes and sizes. The first couple designs of the trees were found to be too short and too small to fit the gameplay. With each iteration, we expanded the scope of the trees and made them thicker. Our latest design of the trees in the game in particular drew inspiration from the Californian redwoods and Sequoia trees (Figure 40), particularly those featured in the Battle of Endor in Star Wars: Return of the Jedi.

Figure 40: Sequoia trees which we used as the basis for our trees in the design (Unique Landscapes)

3.13 Visual World Design

3.13.1 Reference photos

Prior to the development of any game assets or concept art, we decided to compile a library of reference photographs. The wide range of photographs available on the Internet was a helpful resource as we decided on key visual aspects of the game. By compiling a folder of nearly thirty photographs of forests, we were quickly able to nail down several themes, looks and ideas that we all liked. Alongside pictures of landscapes and forests (Figure 41), we also collected photographs of objects. Several of our photos included pictures of crystals (Figure 42), which we had intended to use as a light source. These helped us narrow down the visual appearance of the game.

Figure 41: Shows one of the primary reference photographs we used in designing the game's visual appearance (Redwood)

Figure 42: Shows a photograph we had used as reference for the crystal model (Lovely Terminated White Crystal Rock)

In addition to influencing the physical design of objects in the world, the reference images also helped us make decisions on the color palette. Shying away from the traditional forest colors of green and brown, we found several photographs of Japanese forests which featured red, black and gray coloration (Figure 43 and Figure 44).

Figure 43: Shows one of the main reference photographs we used to influence the color and foliage of the game (Chillstep)

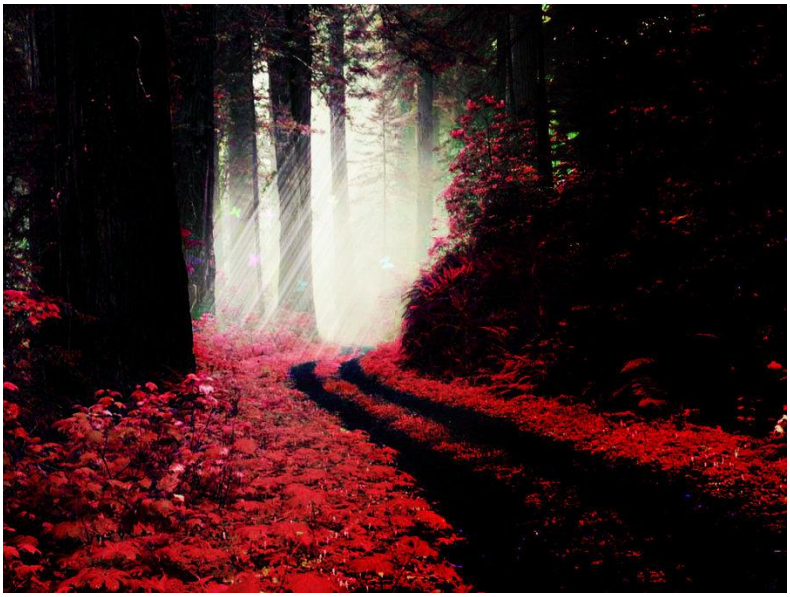


Figure 44: Shows one of the primary reference photographs we used in designing the game's visual appearance (Dawn in the Red Forest)

Although the use of reference photographs helped us immensely in defining the rough design of the game world, we soon reached the point where the specifics of the environment needed to be solidified. At this point, we decided to create concept drawings of the game environment. This would allow us to articulate specific details such as branch shape, tree size, precise color-palettes, and object placement.

3.13.2 Concept Art

Concept art production began during the early stages of development. Before starting on any assets, a range of concept art images were created to help finalize the visual design of the game. We used concept art as a way of quickly displaying how certain specific features such as crystals, foliage, rocks and trees would look. These images were much faster to create than full 3D models, and acted as a low-fidelity prototype. The concept art in this stage was created in a sprite editor. Although the game was not sprite-based itself, sprite editing is a quicker and simpler process than digital painting. We avoided digital painting for concept art as it would have been difficult to change and would have taken longer to produce.

Using the first iteration of the concept images, we determined that a lighter atmosphere would fit the game better, echoing the freedom the player has in swinging around. The images

were used to influence the color scheme, and they quickly helped eliminate and narrow down the style of the game. Figure 45, for example, was one of the pieces that heavily dictated the visual style of the game. The concept art also played an important role in the creation of the first prototype models for the game. Refer to Appendix C: Concept Art for more of the concept art that was produced.

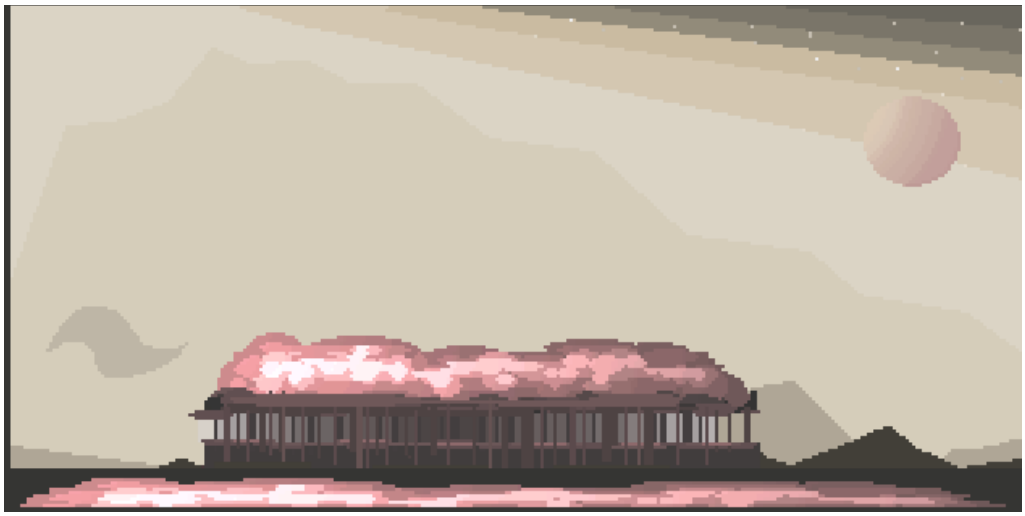


Figure 45: Shows one of the early concept images of the forest from a distance. Visual traces of this concept image can be seen in the skybox texture in Figure 53

3.14 Story

3.14.1 Lore Development

Lore development began directly after our preliminary design. With the game being set in a mythological forest, it was decided that the outside world would be designed prior to the interior of the forest. Before writing the story, an outline was created. Without any details, the outlined story involved the player following in the footsteps of a famous explorer, with the game ending in the player surpassing the famous explorer by doing what they could not.

Initially, the first draft of the lore existed as a detailed timeline. Under this initial storyline, the setting of the game would take place thirty years after a cataclysmic event known as the Great Divide. The Great Divide was triggered by the exile of a powerful warlock, splitting up the world into a chain of island-nations, each containing a massive sum of treasure. The player, an island-dwelling archaeologist, departs on a ship in search of the missing fortune of

one of the newly formed islands. Landing on an island overcome by forest, the player follows the advice of a long-since-deceased explorer, obtains the treasure, and escapes.

After analyzing the original plot draft, it was abandoned because the world contained a plot entirely unrelated to the events of the actual game itself. With the exception of the explorer, all of the characters present in the detailed backstory could be removed with no effect on the game. In addition, many parts about the plot were too similar to generic plotlines found in fantasy movies and books. One example of this is how the exiled warlock character greatly resembled the exile of Jagar Tharn in the *Elder Scrolls* series or the exile of Zarok in the *Medievil* series. The great divide event paralleled the "Great Flood" event from the *Legend of Zelda* series, as it created a series of island-nations from a once-united kingdom.

Following the rejection of the originally proposed lore, a new plot was created entirely from the beginning. While almost no story elements were reused in the new story, the original plot helped immensely in the creation of the new plot by providing a direction that the story should take.

Spurred by our newfound direction, a story was created that was less invested in society and more invested in the natural world. Set on a planet trapped in a never-ending loop, the entire world except for the forest in which the game takes place is wiped away clean every millennium. Although the people in the world are innately aware of the reset and are reborn into new lives the following cycle, they do little to try and resist. Each reset brings about the rise and fall of a new civilization. Although the forest remains unaffected by the reset, it is collectively believed to erase the souls of those who enter it, removing them from being reincarnated again the next millennium.

The player, a person who has entered the forbidden forest, learns the story of the world by following the tapes of a long-dead explorer from a previous cycle. From the final audio tape, the explorer is heavily implied to be one of the previous lives of the player. This plot-point disproves the myth of the forest erasing the souls of those who enter, and paves the way for civilization to be spared from being erased.

3.14.2 Creation of the Audio logs

Following the unanimous approval of the new storyline, the focus turned to figuring out the best way to communicate the storyline to the player. Although a difficult decision, the limitations on the game provided some initial direction. Having no non-player characters in our game, for example, meant that the story had to be delivered through other means. After a brief meeting, it was decided to use pre-recorded sound files done in the form of an explorer's diary. Not only would this allow the story to be delivered in segments, but it would also allow plot development to occur periodically.

Writing the audio logs themselves proved to be a challenge. Although the backstory was nearly finalized during the start of script production, many finer details ended up being determined by the dialogue. The decision to shape the immediate story with the audio logs led to a lot of problems, but also gave us a much stronger degree of player-involvement.

Editing the audio logs took place over the course of several weeks. On the Fridays of each week during the editing process, all group members would pan through and check the audio logs for spelling errors, plot holes and other issues. In order to make sure everyone on the team was happy with the story, we also allowed group members to voice their own personal opinions on the plot. The revision process occurred over several weeks, and many sections were removed or rewritten to accommodate these ideas. The plot of the game changed immensely during these re-writes. While it took many revisions to achieve the final result, we ended up with a well edited and an articulate series of audio logs. The full script of the audio logs can be found in Appendix B: Audio Log Script

3.15 Asset Production

3.15.1 Model Production

After deciding on the visual appearance of the setting, we started producing some early assets. These were created early on in development, when we were still uncertain about the graphical limitations of the computer provided by the lab. The first couple prototype models were produced with a very low polycount. While the tree model originally consisted of 200 polygons (Figure 46), our early in-engine tests proved we could increase both the scale of the

model to be much larger as well as the polycount to allow higher detail, especially in the branches.

Figure 46: Shows one of the early tree designs. Note the high concentration of polygons at the base of the tree

The tree models were the first priority, but other models were also important for early development of the game. For example, a wooden platform for the player to stand on was vital to the gameplay. In addition, separate branch models needed to be produced, which could be inserted into the tree models to provide modular points for the player to grapple to.

Changes to the models began after several iterations of testing. Trees were made wider, taller and were given fewer polygons to improve performance. Playtesting found the boards on the wooden tree stands to be too large compared to the player, so they were modified to

consist of a large number of small boards rather than a relatively few number of large boards (Figure 47).

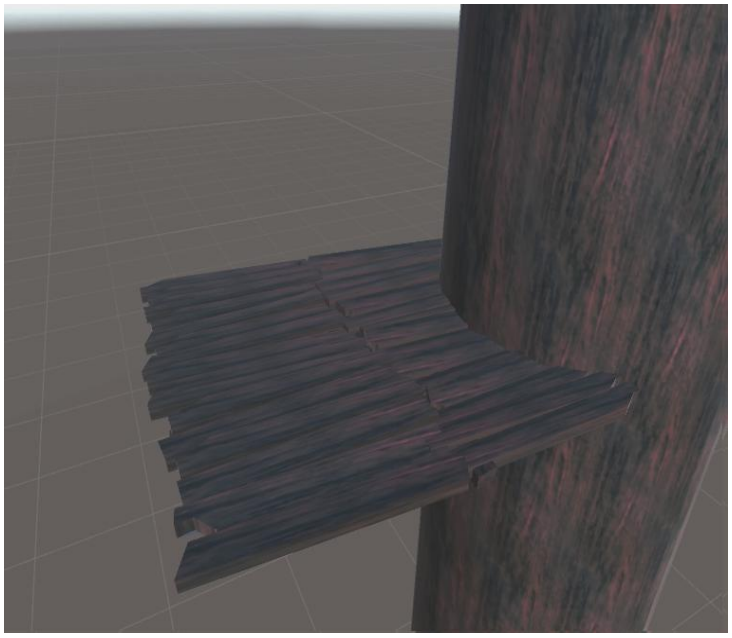


Figure 47: Shows the revised wooden platform to have a larger amount of boards

While asset tweaking was relatively quick, it cleared the models of any existing UV maps and required that they be manually replaced. Most of the detail in the early models (Figure 46 and Figure 48) was concentrated in the roots and branches. When we reduced the polycount of the tree models to improve performance, these areas of the model were more specifically downscaled compared to the rest of the model. Figure 49 depicts the tree and branch models as they exist in the final version of the game.

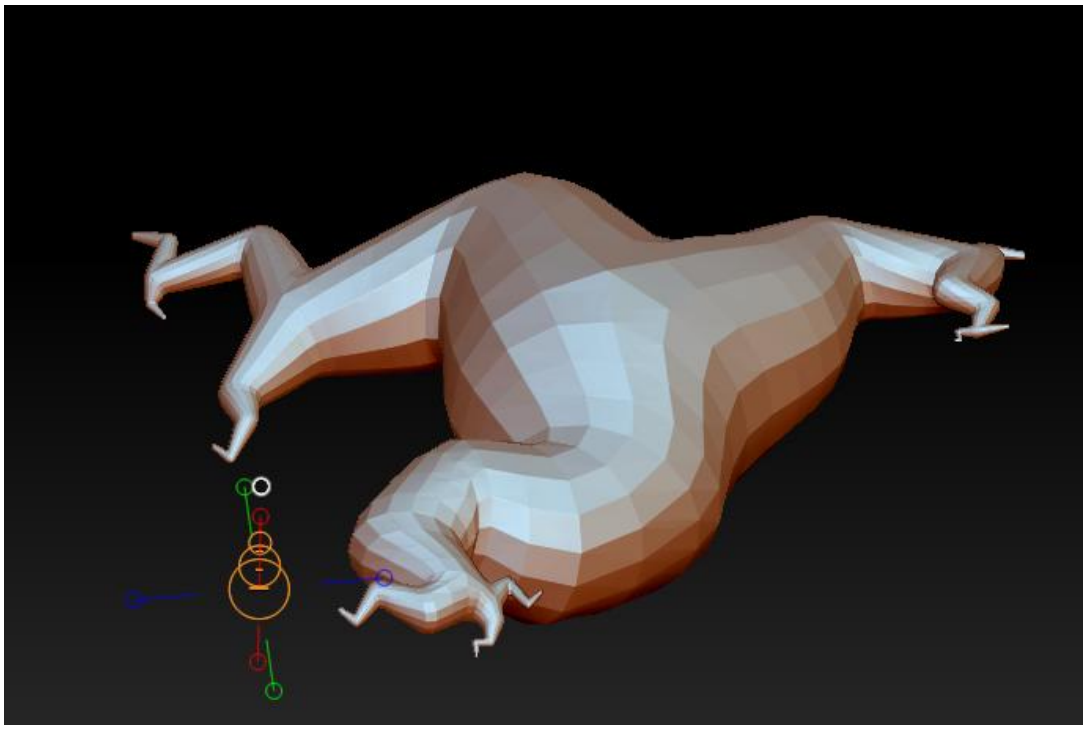


Figure 48: Shows one of the roots of a tree in production. Later tree model revisions were less focused on root detail.

Figure 49: Shows the tree models and branches in the game

Some of the last models completed were the non-essential game objects. To represent the long-deceased explorer character from our audio logs, we used a skull model (Figure 50) which coincided with the final audio recording.

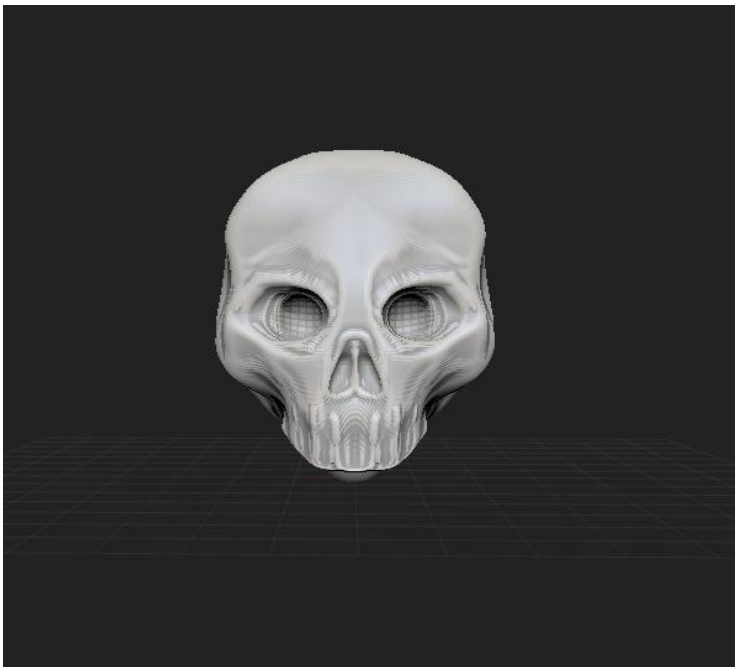


Figure 50: Shows the skull model created for the remains of the explorer character

3.15.2 UV Mapping

One of the most difficult aspects of creating the 3D models was UV mapping them. UV mapping is the process which defines how a texture will wrap around an object. As an example, Figure 51 depicts the UV map for the audio log recording device while Figure 52 depicts the model with the texture applied to it. While several programs such as Autodesk Maya and ZBrush are capable of auto-generating UV maps for 3D models, most of the maps for organic models, such as trees, had to be done manually. Although UV mapping proved to be very time consuming, simple objects could be mapped fairly quickly. Complex models, such as trees with branches and roots proved to be more of a challenge, and took several iterations before looking realistic.

In several instances, some models had to be UV mapped multiple times. As models were modified throughout the course of production, their UV maps were commonly rendered void.

Remaking the UV maps for the models each iteration became fairly tedious, although our decision to divide the trees into separate trunks and branches made the process easier.

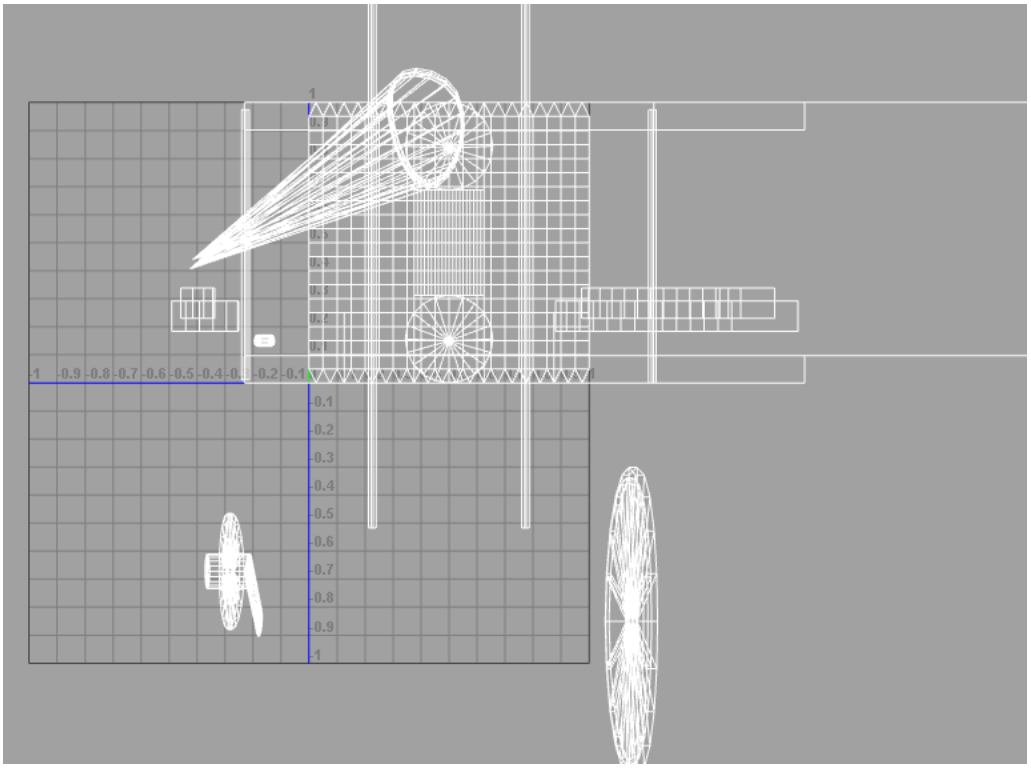


Figure 51: Shows the UV map of one of the game models in Autodesk Maya 2015

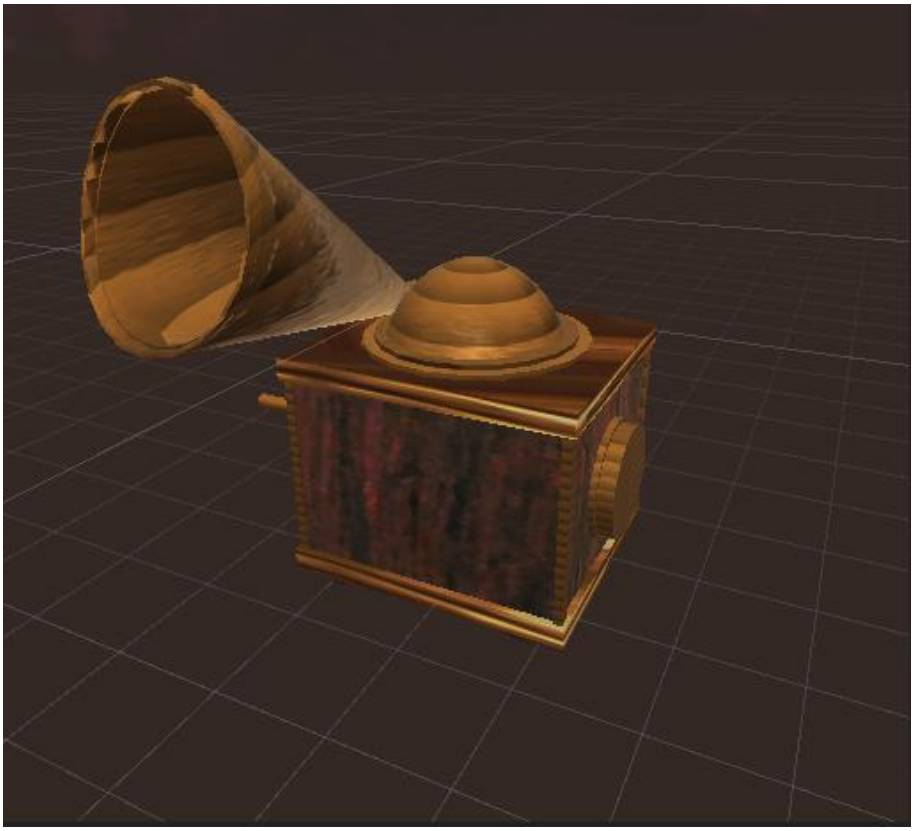


Figure 52: Shows the UV from Figure 51 applied in Unity

3.15.3 Texture Production

Texture production began shortly after several models were completed. While we had the option to download and modify free textures from the internet to speed up the process, we decided to create our own. We were concerned that using free textures would make our game appear similar to other games and would detract from the overall uniqueness of the game. Creating our own textures was a fairly lengthy process, but gave us the ability to more accurately portray the visual style that was envisioned in the reference images and the concept art. For example, we were able to add hints of the red color palette to the bark of the trees and the foliage on the ground (Figure 53).

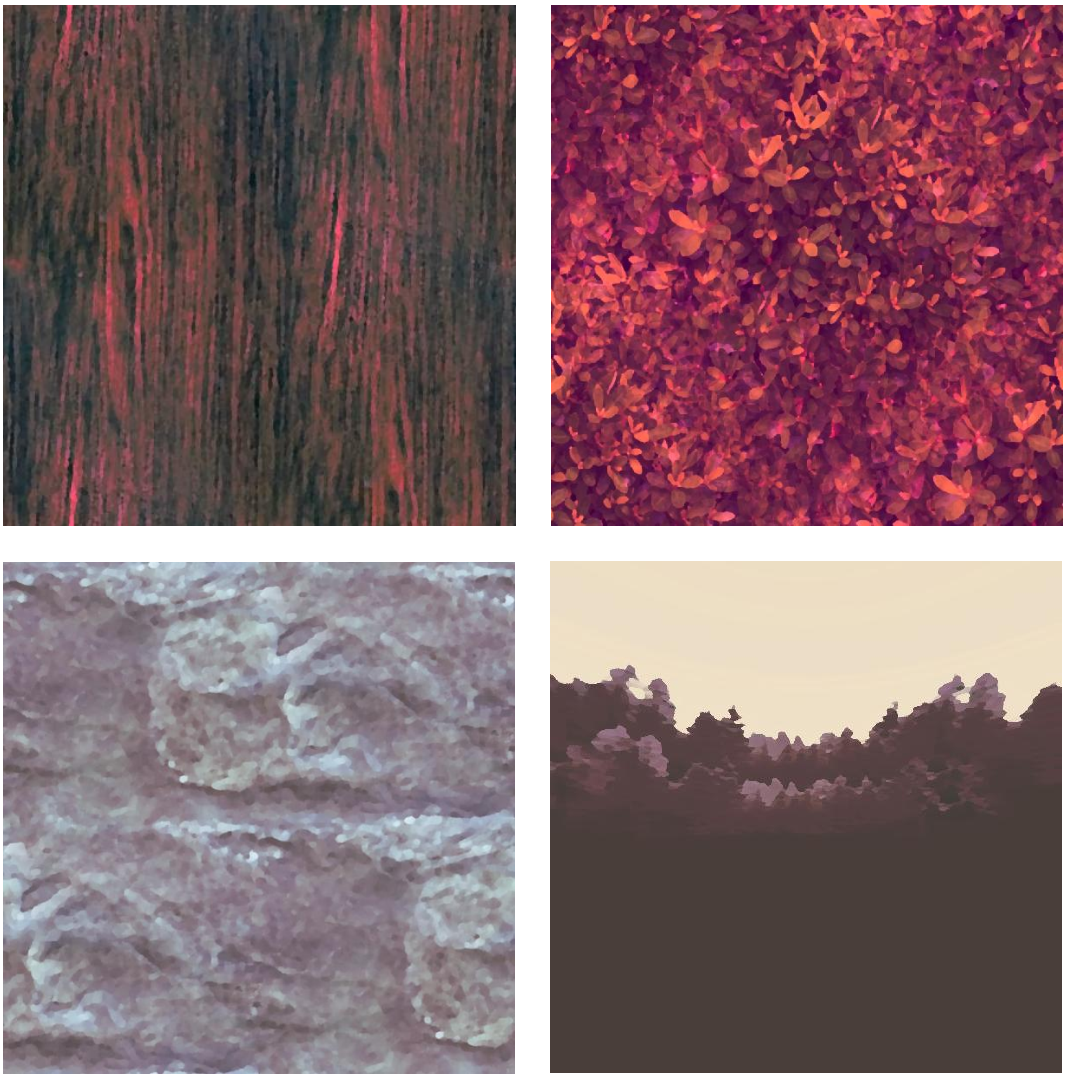


Figure 53: In clockwise order from top left: tree bark texture, leafy ground texture, skybox texture, rock texture

Originally, the textures produced were created at a resolution of 2048x2048 pixels, a standard size for high-resolution images. While this provided us with a good amount of detail, it later became apparent that the size of the textures was too substantial. With the average file size of each individual texture averaging at 16 MB, scenes with large numbers of objects would slow the frame rate below the targeted 60-75FPS required of the Oculus Rift.

To rectify this issue, we scaled down many textures, in particular those of which the player would only see from a distance. The texture resolutions were scaled down to 512x512 pixels; these changes alongside changes to the model polycounts improved performance with

the Oculus Rift. While we were able to do so without causing too much blurriness in the textures, the UV maps of the models had to be tweaked to accommodate the new changes.

One of the more difficult aspects of creating textures for *Hikari Hook* was the creation of realistic looking tree foliage. Because the Oculus Rift is incompatible with billboarding, the process of moving a flat texture in-line with the camera to make it look 3D, we were limited in the ways in which we could effectively display fauna and leaves. Ultimately, we decided we would have 3D planes mapped with transparent textures. The process to accomplish this involved the creation of an alpha map in Adobe Photoshop. Because we did not have access to Adobe Photoshop, we created the alpha mapped textures in Gimp 2, a free competitor of Adobe Photoshop, and converted the file into a standard .PSD file. Because Unity supports alpha cutout shader transparency with .PSD files, we were able to accomplish the visual appearance of a transparent canopy (Figure 54).

Figure 54: Shows the alpha transparent foliage texture mapped onto a 2D plane

While it visually appeared accurate to what we were trying to accomplish, the cutout shaders proved to be taxing on the performance of the computer we were provided. Particle

effects had to be mapped onto 3D planes as well, rather than being billboarded, causing further performance hindrance. Because we did not have access to a stronger computer, this meant we had to be more conservative with our foliage and particle usage. We had planned to use a lot of particle effects in our original design, particularly in the form of mist (Figure 55), falling leaves, and with the Lighthook beam (Figure 56), but we ended up only using them in essential areas to keep the game running at optimal performance.

Figure 55: The fog particle effect which was used in level transitions

Figure 56: Example of a particle effect that was cut to improve performance.

4 Sound Design

4.1 Music

We wanted to accomplish several things with our soundtrack in order to add depth and substance to our game. First, we wanted the music to reflect the story and world we had built and to complement the tone conveyed. Furthermore, we wanted the soundtrack to fit with the aesthetic of our visuals. We also aimed to pay homage to Japan for hosting us by using some traditional Japanese instruments and styles thrown into the score.

4.1.1 Influences

We pulled from a few different sources as inspiration for the tone of our soundtrack. One of the largest such influences was Jeff Broadbent's soundtrack for the game Dawngate (Broadbent). The soundtrack for that game used a wide range of instruments from various cultures and styles, ranging from Asian, African, European, to even synthesized instruments. This created an otherworldly sense and feeling of mysticism. Another part of the soundtrack that was stylistically influential was how the entire soundtrack primarily revolves around two themes. This has the effect of creating a sense of cohesion, even if the style of the piece changes significantly, as the themes tie the pieces together.

Another source we drew influence from was traditional Japanese styles and instrumentations. The common combination of a shamisen and koto, in addition to the shakuhachi and taiko drums, as shown in Figure 57, bring a very distinct sound. The texture of these instruments typically brings about a spiritual, down to earth feeling. We felt this would complement the tone we wanted to go for well while also giving the soundtrack the Japanese flair we were looking for.

Figure 57: In clockwise order from top left: shakuhachi (YungFlutes), taiko drum (Evans), shamisen (Also), koto (readMedia)

4.1.2 Composition

For reference, the full musical scores described in this section can be found in Appendix D: Music Scores Our soundtrack starts off by laying down the basic foundations the whole set of scores is based on. First, it is largely driven by a strings section with two solo violins alongside a small group of traditional Japanese instruments with a distinct accent in the upper registers from a piccolo. The opening demonstrates creating the sense of mysticism through light use of dissonance in 7th chords, starting on a 7th chord on the tonic of A minor. While this is held, the main theme holding all the pieces together is introduced on the koto. This theme can be broken into two components, the first of which is represented by the first five notes on the koto, the second being the following two notes on the koto. Variations on these two ideas are what nearly every melodic line consists of. The harmonic structure and progression of the piece does not follow any strict direction, but rather has an ebb and flow to it. The harmony varies between moments of tension through using dissonance in ii, V, and vii chords at times voiced as

7th or 9th chords and also utilizing sus2 and sus4 chords to the same effect, to periods of lull and resolution as that tension breaks, often using standard i chords on the tonic.

Each piece does not modulate, but rather is very heavily rooted in the tonic the piece is centered around, often repeating the i chord very frequently. The changes in key occur between the pieces, rather than during them, as the tracks for the main menu, level 1, and the credits are all in A minor, while level 2 is in D minor, and level 3 is in E minor, all closely related keys in order to create a strong sense of cohesion between them. In each subsequent piece, more instruments are added to the orchestration to create a more active and increasingly more intense texture. In addition, the tempo increases in each subsequent piece and the percussion is more heavily emphasized. This is all to make each piece feel increasingly darker and more intense, to reflect the ever increasing danger and perils of the forest in addition to the unfolding mysteries of the forest and the narrative of the explorer whose audio logs you find along the way. The only exception to this is the transition from level 3 to the credits, which slows down and has a thinner texture as a resolution to the tension built by the previous three levels.

The credits piece is very closely related to the main menu piece, which is intentional to accomplish a few things. In doing this, the transition from the end of the game back to the main menu is as smooth as possible. This also plays to the heavily repetitive through variation nature that the soundtrack has as a whole, taking the majority of the structure of the main menu track while also providing a fresh take on it to evoke a different mood. While the main menu track mainly embodies a near innocent fascination with the mystic nature of the forest, the credits track puts a somber, melancholy spin on it. In the credits track, the true nature of the forest is uncovered, the fate of the explorer and his relation to you is made known, and the exhausting perils of the forest have been surpassed. This is, in a sense, a release of tension and relief, and as such the piece brings the soundtrack back to the original key and in a presentation very close to the original presentation on the main menu. However, this is also a burden of knowledge, the secrets uncovered are substantial, but yet nothing seems to come to a full closure. The nature of the forest is discovered, but nothing is actually changed within the timeframe of the game, rather the aftermath is left to the imagination of the player. As such, the credits theme

carries a more pensive tone, one that must carry this knowledge as a burden, both as the sole living witness to it and also the sole potential messenger for change. This is the end of one journey, but one that very likely would be the beginning of another struggle to put the knowledge acquired to use, which the credits piece conveys through its tone.

4.2 Sound Effects

Initially we set out with the aim to create our own sound effects from scratch, recording them ourselves. We believed this would grant us a level of finesse and attention to detail that would greatly enhance immersion. We planned on having different sound effects for each different material the player can make contact with, as well as having many smaller sounds based around more nuanced interactions, such as branches creaking slightly when the player begins to swing from them.

Unfortunately, in practice recording our own sound effects was not feasible in the time we could afford to allot to this. We realized after attempting to record our first set of sound effects that the microphone we were using was picking up a very large amount of background noise, to such a degree that it was unreasonable to attempt editing it out. Without any convenient way to replicate the scenarios we needed to record without being outside, or specifically in a room with acoustics that would be fitting, we were forced to abandon this approach for the sake of time. As such, we looked into acquiring sound effect assets online, then editing them to fit our needs.

5 Results & Conclusion

In light of the above experiences, the overall project was a success and ended up producing a complete and functioning game. Although the project was a success, there were also many difficulties and problems encountered during the development process. Drawing from these experiences, there are several things that could have been implemented or changed about the group process to make the entire process easier.

One of the biggest issues encountered was a lack of procedural testing. During each iteration, many new and important features were added to parts of the code that were dependent on the functionality of other parts. Because of this, different parts of the game sometimes broke or new bugs were introduced. Although not completely solving the problem, creating a concrete set of tests to run every other week would have definitely improved the overall process. Whether this would have actually saved time and been beneficial during the production of *Hikari Hook* is questionable because of the overhead that testing can cause to small teams. Beside this point, testing is a good practice and was something that should've been more strictly enforced during the development process of the game.

The second biggest change that could have smoothed the development process was continuous art integration. Although art was being tested and added when it was completed, huge scale testing of the art was something that should have taken higher priority. Due to the available equipment, the amount of graphical processing power available was limited. This meant that there were higher graphical restrictions on the game. Figure 58 shows one of the earlier prototype levels that had to be optimized due to lack of computational power. In addition, several art textures and models had to be redone to utilize lower resolutions and polycounts respectively. However, this was not caught until near the end of the project. Thorough mass scale testing of the assets could have caught the issue a lot sooner.

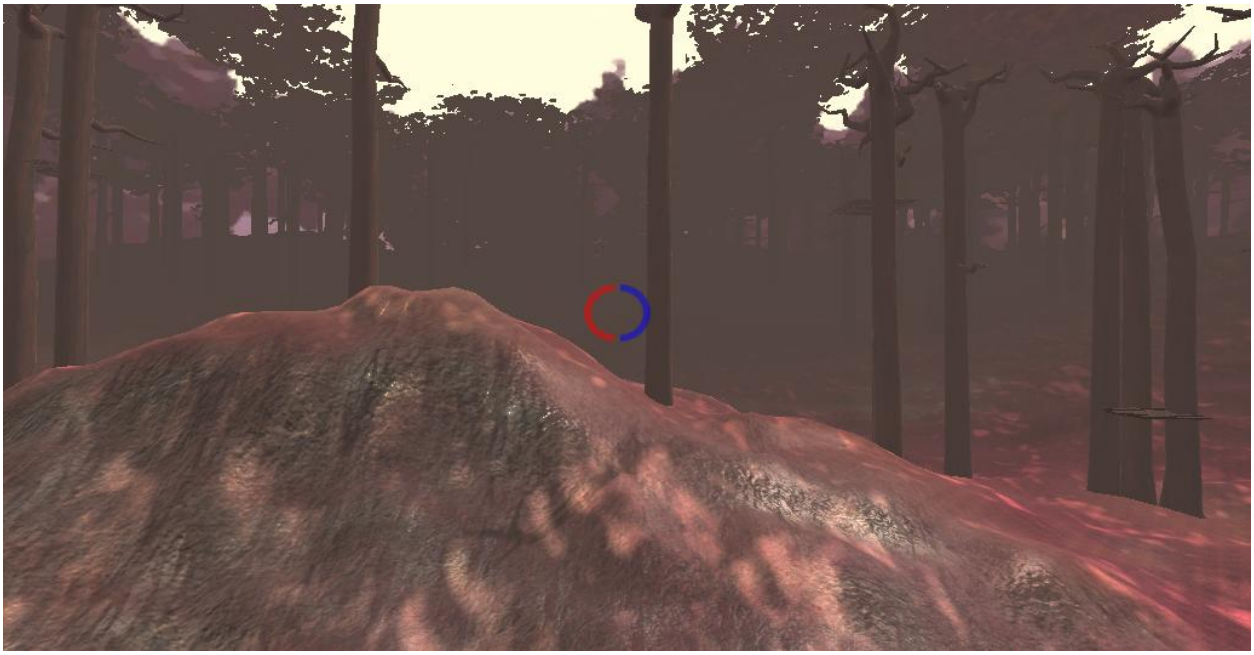


Figure 58: Shows the prototype design for a section in level one.

Besides these issues, the overall feedback from our playtesting sessions revealed that players had mixed feelings about our method of grappling hook traversal. Users tended to report that it was a fun experience, but a significant portion of our user base had difficulty playing for an extended period of time. Some players reported that they felt sick from some of the sudden movements, while others were far less susceptible to these effects. The users that got sick commented that some of the rotations felt really fast and somewhat unexpected in some cases, primarily the adjustment when the character is first entering a swing. However, they did feel that for the rest of the swing, the steady angle adjustment felt fine. In order to make the game more accessible, future work will involve tuning of the speed at which rotation of the player is done, especially where changes need to be made suddenly. As a further measure, we will also be testing with versions where only the player's yaw is adjusted. This will allow for the gameplay benefits of keeping the player's camera pointed in the general direction that they will want to be facing while possibly making the experience less unsettling.

In addition to the feedback and fixes from the playtesting, there is still much more work that could be done to improve the overall player experience. One of the major outstanding objectives is to look to promote the game at various gaming conventions and events. This

would not only allow more people to experience the game, but also provide more diverse and valuable feedback on some of the controversial mechanics in the game. In addition to showing off the game, there are small bug fixes and asset tweaking that could be done to further improve playability and game quality. One example of this is to add periodic falling leaves when walking through the forest. Even though these are not necessary for the game to function, they would improve the immersive factor of the game and would make the overall playthrough a more enjoyable experience.

Works Cited

"1.5 EARTH Model Shakuhachi" Digital Image. *YungFlutes*, n.d. Web. 8 Nov. 2015.

Akira. Dir. Katsuhiro Otomo. Toho Co. Ltd, 1988. Film.

Also. "Sanshin & Shamisen Music" Digital Image. *TheApricity*, 3 April 2015. Web. 8 Nov. 2015.

Avatar: The Legend of Korra. Dir. Bryan Konietzko and Michael DiMartino.

Nickelodeon Animation Studio, 2012. Television.

Broadbent, Jeff. *Dawngate*. 2013. Electronic Arts Inc, 2013. Video Game.

Buckley, Sean. "Here's What Valve's Virtual Reality Controllers Look Like" Digital Image.

Gizmodo, 4 March 2015. Web. 8 Nov. 2015.

Chillstep. Digital image. *Aboutcity*. N.p., 08 Feb. 2013. Web.

Dawngate. Electronic Arts Inc, 2013. Video Game.

Dawn in the Red Forest. Digital image. *Deviantart*. N.p., 17 Apr. 2010. Web.

Evans, Steve. "Taiwan Drums". Digital Image. *Flickr*. 27 Nov. 2007. Web. 8 Nov. 2015.

Final Fantasy X. Dir. Yoshinori Kitase. Square, 2001. Video Game.

"FOVE: The World's First Eye Tracking Virtual Reality Headset." *FOVE*. Fove-Inc. Web. 8 Nov. 2015.

Kumar, Manoj, Mohammad Husian, Naveen Upreti, and Deepti Gupta. "GENETIC ALGORITHM: REVIEW AND APPLICATION." *International Journal of Information Technology and Knowledge Management*: 451-54. *Computer Science and Electronic Journals*. Computer Science and Electronic Journals. Web. 8 Nov. 2015.

"Kyoko Okamoto to Present Koto Recital Sept. 22 at Lebanon Valley College" *readMedia. readMedia,* 2 Sept. 2014. Web. 8 Nov. 2015.

Legend of Zelda: Wind Waker. Nintendo Co. Ltd, 2002. Video Game.

Lovely Terminated White Crystal Rock. Digital image. *Depositphotos*. N.p., 2009. Web.

Mad Max: Fury Road. Dir. George Miller. Warner Bros. Pictures, 2015. Film.

Medievil. Sony Computer Entertainment Europe, 1998. Video Game.

Nickinson, Phil. "What It's like to Live in the HTC Vive Virtual World!" *Android Central*. Mobile Nations, 4 Mar. 2015. Web. 8 Nov. 2015.

“Razer Hydra Portal 2 Bundle” Digital Image. *RazerZone*, n.d. Web. 8 Nov. 2015

Redwood. Digital image. *Find Your Middle Ground*. N.p., 10 June 2014. Web.

Rift. Digital Image. Oculus. Oculus. Web. 1 November 2015

"Rift." *Oculus*. Oculus VR, n.d. Web. 1 Nov. 2015.

Ronacher, Armin. "Flask Web Development, One Drop at a Time." *Flask (A Python Micoframework)*. 2014. Web. 8 Nov. 2015.

Shadow of the Colossus. Sony Computer Entertainment, 2005. Video Game.

Sony. “PlayStation Move Motion Controller” Digital Image. *Amazon*, n.d. Web. 8 Nov. 2015

Splatoon. Nintendo Co. Ltd, 2015. Video Game.

Star Wars Episode VI: Return of the Jedi. 20th Century Fox, 1983. Film.

"SteamVR." *SteamVR*. Valve Corporation. Web. 8 Nov. 2015.

The Elder Scrolls Arena. Bethesda Softworks, 1994. Video Game.

Unique Landscapes. Digital image. *Our Beautiful World*. N.p., 4 July 2013. Web.

Appendix A: SteamVR HTC Vive Application

In the early planning stages of the project, we applied for an HTC Vive development kit with our primary idea at the time, a Pro-bending inspired game. The full text from our application is provided here.

Appendix A.1 Application Contents

In our game, players use martial arts forms to manipulate the elements and cast earth, fire, and water attacks at their enemies. By moving in the real world players can dodge incoming attacks; and with hand motion controllers, they can shoot blasts of water or fire and lift and throw rocks. With these powers, players compete in sports like combat tournaments.

We are Worcester Polytechnic Institute students working at Osaka University's Takemura Lab. We will leverage WPI's and Osaka University's VR work in presence and traversal in immersive virtual environments to create a consumer game alongside publishable research.

We will be documenting our development as we explore room-scale VR. Throughout our design process we will improve our product using feedback from developer communities in Japan, Boston, and the Steam network. At the end of our project, we will have a polished game, and a publicly available research paper on our designs, methodologies, and VR development process.

Appendix B: Audio Log Script

Appendix B.1: Entry 1

[Calm, peaceful, somber]

Today marks the end of the first day of my journey into the forest. I arrived early this morning and spent the last five hours climbing rocks and scaling difficult terrain. With no way to navigate the forest, I became horribly lost, until I stumbled upon a rather peculiar device. When attached to the hips, this device allows you to launch hooks connected to beams of light. I have dubbed it, “The Lighthook” for the moment. The terrain in these ancient, forbidden woods is nearly impossible to traverse on foot, and without this device, it seems I would have been left with few options.

[With a feeling of slight trepidation, hopeful to find something worthwhile for the sacrifice, but unsure of what the outcome will actually be.]

As all who enter this sacred forest, I have risked and lost so much coming here. I pray that it will not be in vain.

Appendix B.2: Entry 2

[Speaking with drive, confidence, and a resolve to his cause]

Many of my peers questioned my decision to enter the forbidden forest. Some even tried to stop me. I ignored them, and by doing so was banished from ever returning home. Perhaps I deserved such a punishment for my selfishness, as entering the forbidden forest is one of the only acts in this world that has any real permanence. As all people know from birth, the world is reset every one thousand years. Each cycle, we build vast civilizations, only to have them erased. Although we are born into new lives in the next cycle, we live only to have our achievements, our whole histories, forgotten. The only memories that seemingly survive the reset are those about the nature of the cycles, reincarnation and the dangers of the forest.

Truthfully, the reason for my exile stems from the fact that the forbidden forest erases the souls of those who enter it. In my doing so, I have prevented myself from being reincarnated in future cycles, effectively shrinking the potential human population. It was a choice I readily made, and a choice I would make again.

[Heavy seriousness, grave tone, apprehension at the increasing difficulty in navigating the increasingly perilous forest]

The discovery of the Lighthook has allowed me to make it as far as I have, but the environment is only getting more and more unforgiving the deeper into the forest I swing. A single error could bring my entire life to an end. I cannot afford to make any mistakes.

Appendix B.3: Entry 3

[Inquisitive, fascinated by the device]

In the days since I have entered this forest, I have grown quite fond of this grapple device, the Lighthook. Last night, I decided to inspect the inner workings of the Lighthook for my research. The device is incredibly intricate, yet somehow the function of each component is readily apparent to me. Peering into the inner components,

[almost in disbelief]

the device appears as if the creation of this machine has spanned a dozen cycles. Each contribution is well thought out, and, despite the varying materials, parts and technology, the device operates as if its designers were in perfect harmony. Of course, because cycles erase all traces of civilization and life, it is impossible for any two people from different cycles to have ever met.

After reassembling the pieces of the device I had removed, I realized I had no way to tell which way I had come from. With limited supplies, backtracking would be costly to me. Out of the corner of my eye, I noticed a strange anomaly.

[Puzzled and slightly concerned]

Appearing as a ghastly spectre of sorts, it drifted around me, as if to draw my attention. Suddenly, it took off in a direction, disappearing into the thick fog. In an effort to catch another glimpse, I followed it for a while and found myself venturing into thicker forest. If it was indeed a figment of my imagination, why did it lead me further into the woods?

Appendix B.4: Entry 4

[An odd sense of satisfaction at his venture, while also resigning himself to having forfeited eternal life. Speaking as if desperate to share these discoveries and legacies, so as not to let them be forgotten as the rest of the world]

It has just struck me that very few people have ever made it as far as I have into the forest. This place has remained unaffected by the reset since the dawn of time. The contents of this forest span thousands upon thousands of years, never having been wiped away by the end of a cycle. Unphased by the reset, the forest here has grown thick and strong over thousands of years. Little sunlight can penetrate the dense canopy, and I have experienced and seen things nobody has ever lived to tell about. This will be my final lifetime, and most likely my last adventure. I suppose if you're listening to this audio log, then this is your last adventure too.

Appendix B.5: Entry 5

[Filled with wonder and excitement at realizing the potential implications of this increasing familiarity with his surroundings. A newfound determination to push forward prevails]

Today marks a strange breakthrough in my journey. Several hours ago, I experienced an intense headache. When I regained my senses, much to my surprise, my surroundings felt familiar. The strange grapple device hooked on my belt felt like something I had owned my whole life. For a brief moment, it was as if I was another person, living another life.

As the feeling subsided, my mind raced for an explanation. I could feel my subconscious stirring, as if to tease me with details hidden away from my conscious. Perhaps it was fate that compelled me to come here. Perhaps it was destiny that caused me to discover the Lighthook amongst the thick rotting leaves. It is not certain to me the role here that I play here, nor is the real significance of my journey.

I must press onwards deeper into the inner circle of the forest. Although I cannot presently say why, I feel drawn in by a strange sensation of belonging. The further I get, the stronger the urge I have to keep going. Surely you must feel it as well. It is nearly impossible for me to turn back, let alone slow my pace.

Appendix B.6: Entry 6

[Exacerbated, breathing heavily and in great pain and anguish. Speaking with a sense of urgency, resolve, and enlightenment at figuring out the mysteries of the forest and cycles, yet resigned to his fate]

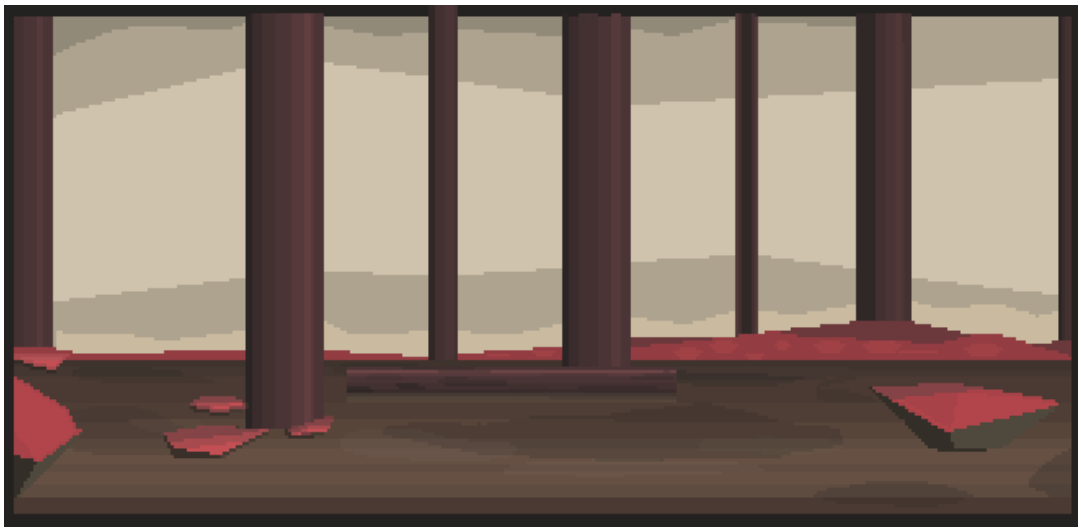
I have finally stumbled upon the truth, but at a heavy cost. In an almost dream-like state, I became careless and severely injured myself after a terrible fall. My entire body is in shock, and several of my bones are shattered. But, in this near death condition, things have never been more apparent to me.

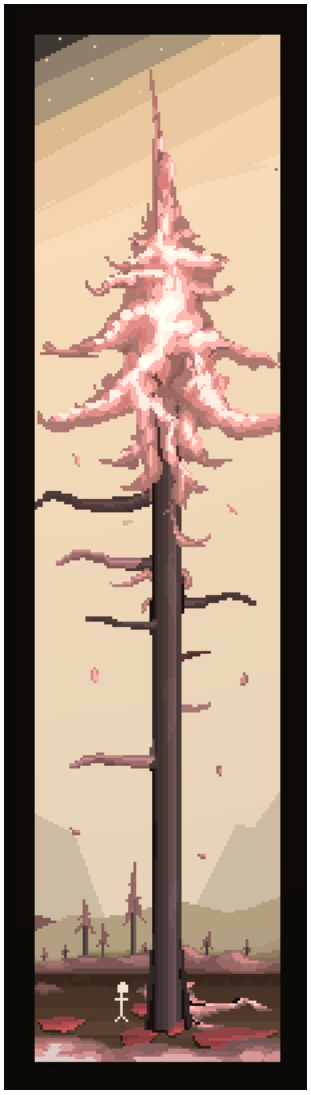
A flood of collected memories filled my head. Memories of places I have never been, of cultures and civilizations long forgotten. Of family I have never met, food I have never tasted. Of the Lighthook and many whimsical devices like it. And of the forbidden forest.

It is now clear to me that, while the forbidden forest does remain free from the effects of the cycle reset, it does not erase the souls of those who enter.

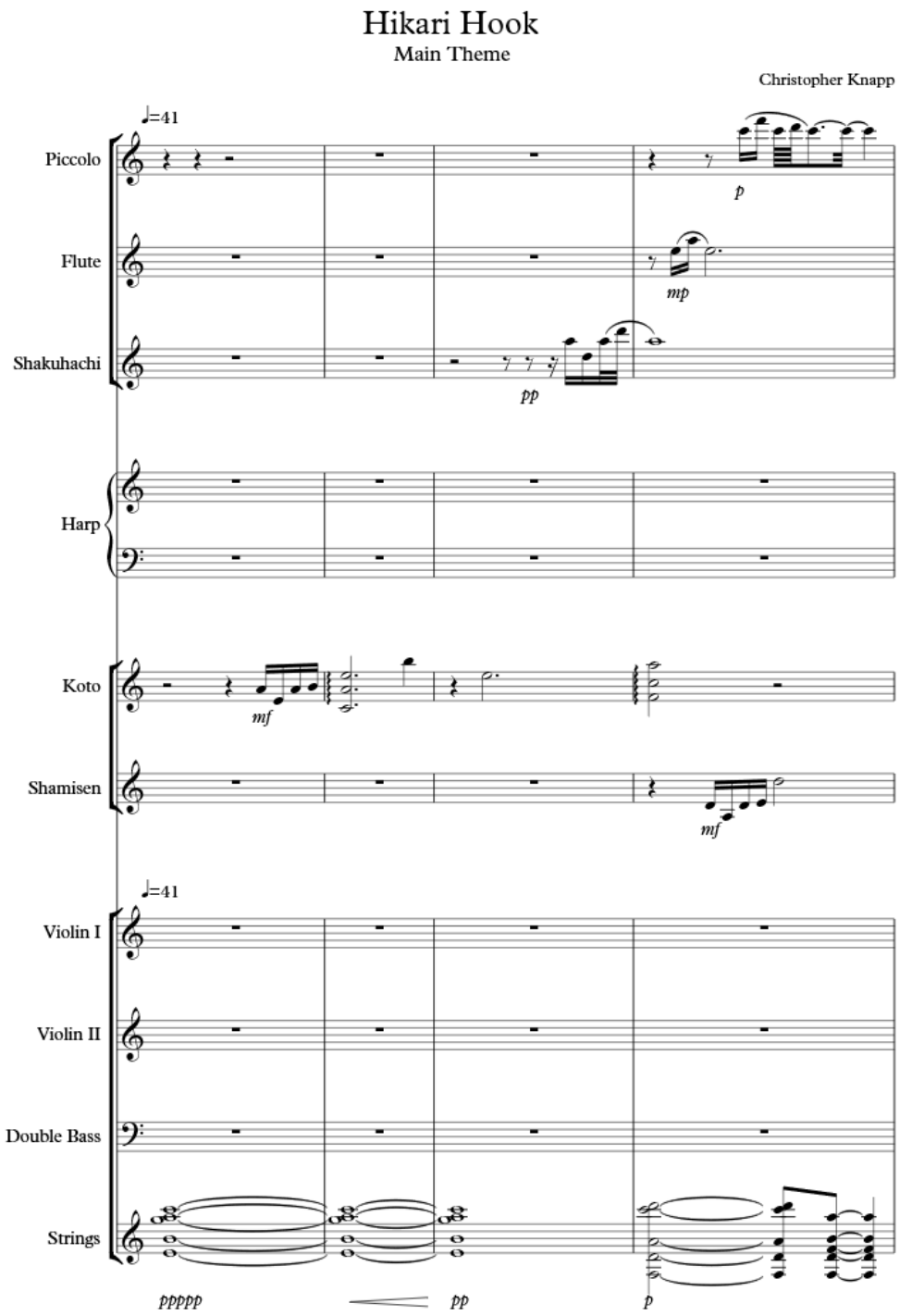
Were I able to return home, I would be able to tell the people of my cycle about my discovery. But, in my current condition, I am nowhere near capable of leaving this spot, let alone the entire forest. I can only hope that the drive that brought me here in this cycle and every cycle before remains to bring me back again in the next cycle.

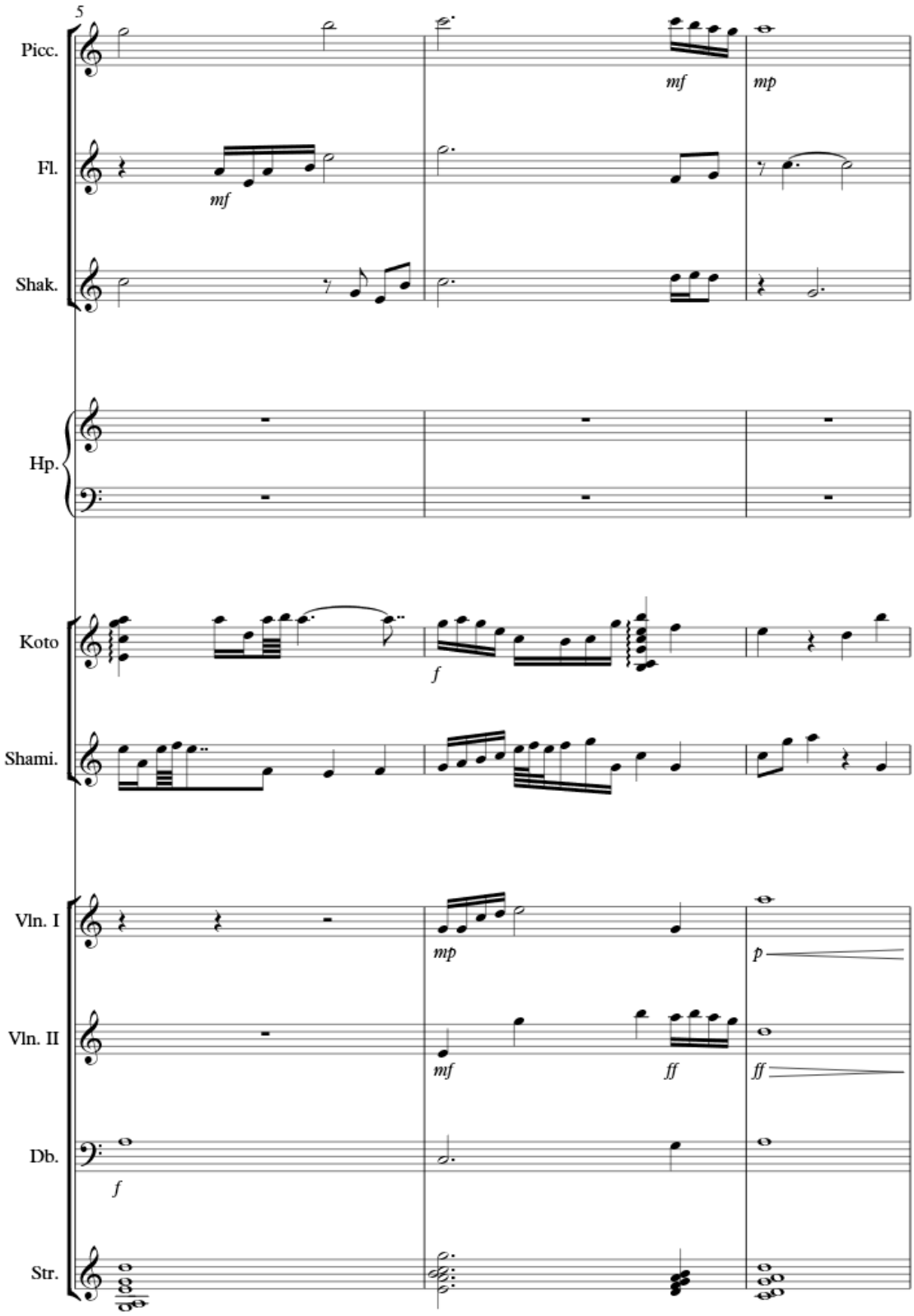
Appendix C: Concept Art

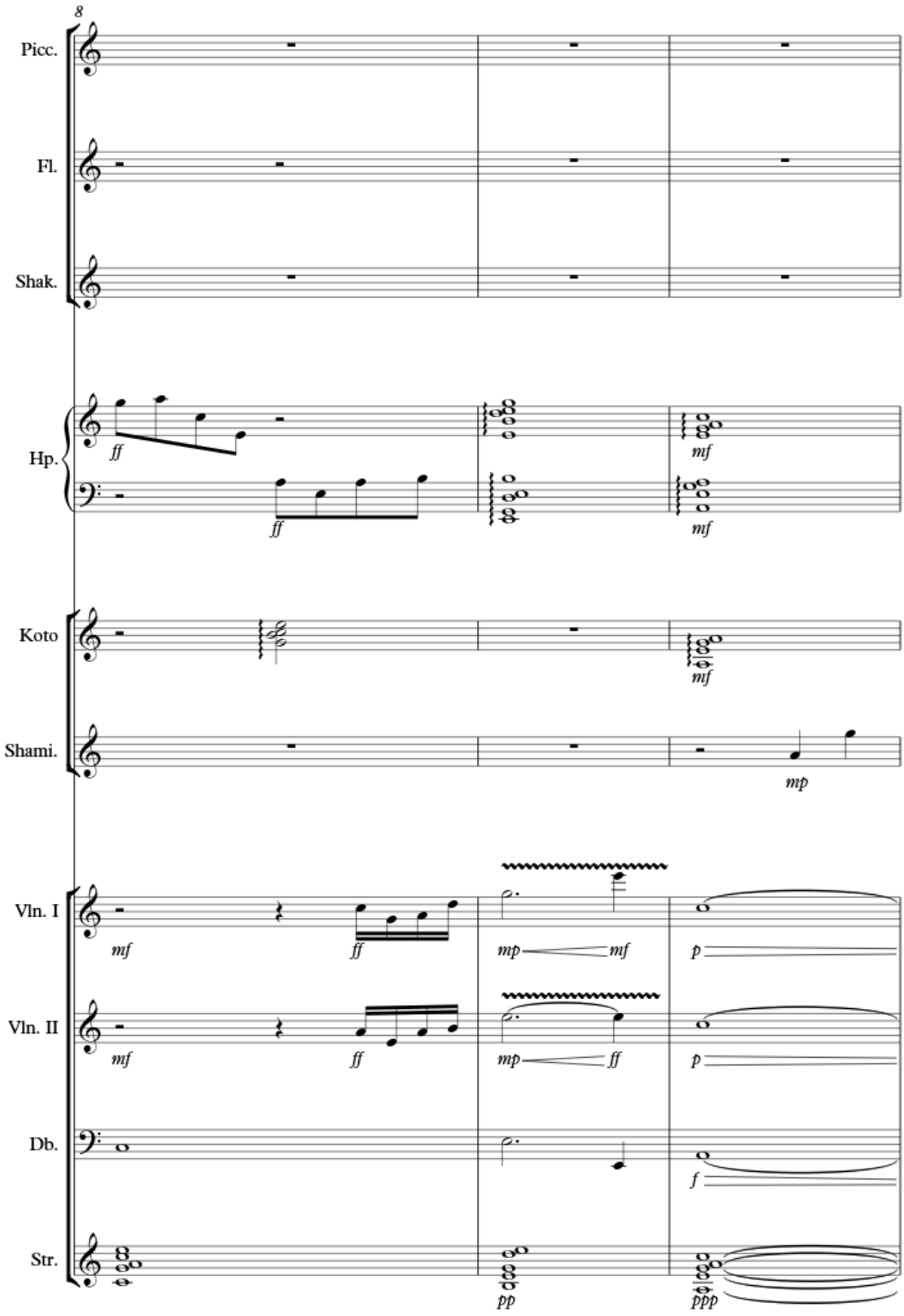


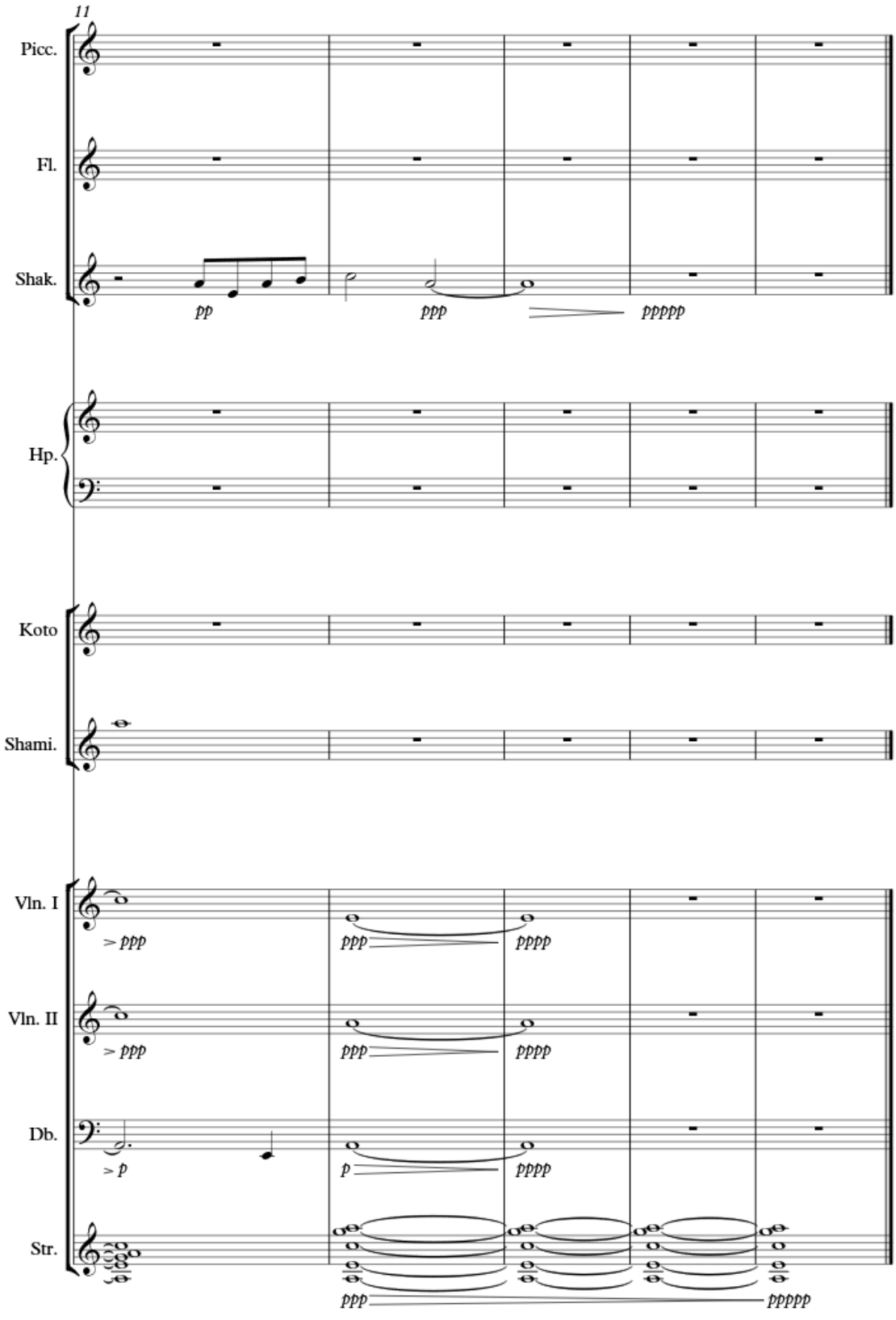


Appendix D: Music Scores




5
Picc.
mf
mp
Fl.
mf
Shak.
Hp.
Koto
f
Shami.
Vln. I
mp
p
Vln. II
mf
ff
ff
Db.
f
Str.

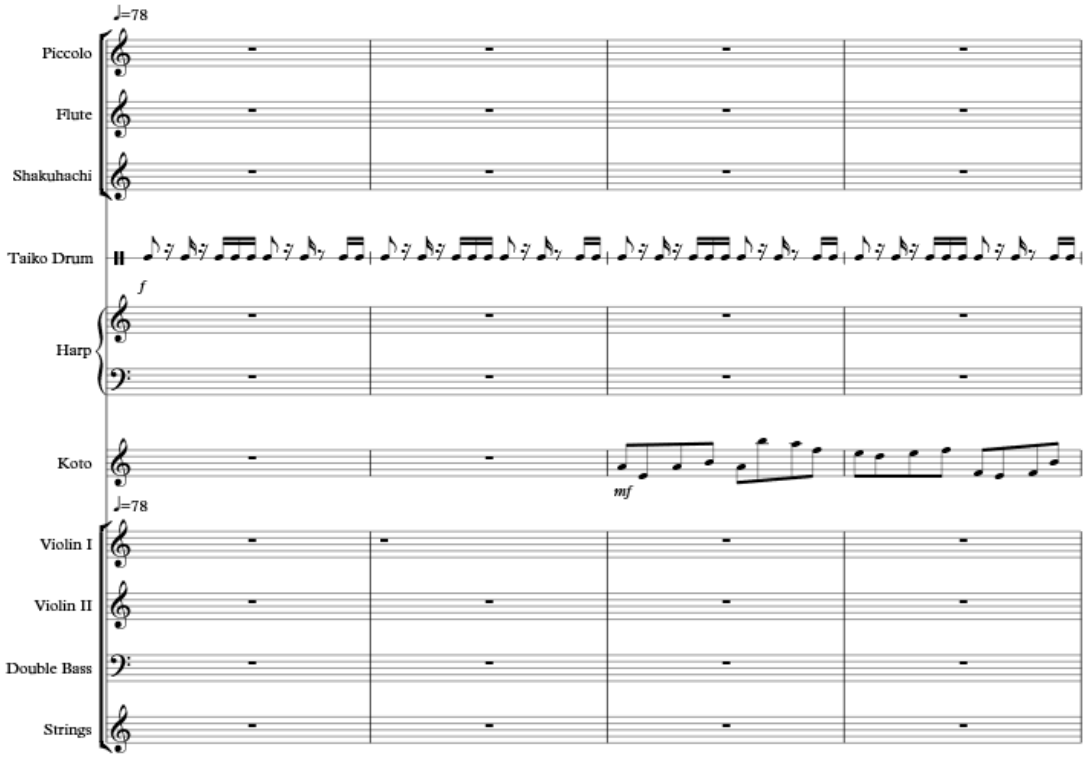

8
Picc.
Fl.
Shak.
Hp.
ff
ff
mf
mf
Koto
mf
Shami.
mp
Vln. I
mf
ff
mp
mf
p
Vln. II
mf
ff
mp
ff
p
Db.
f
Str.
pp
ppp

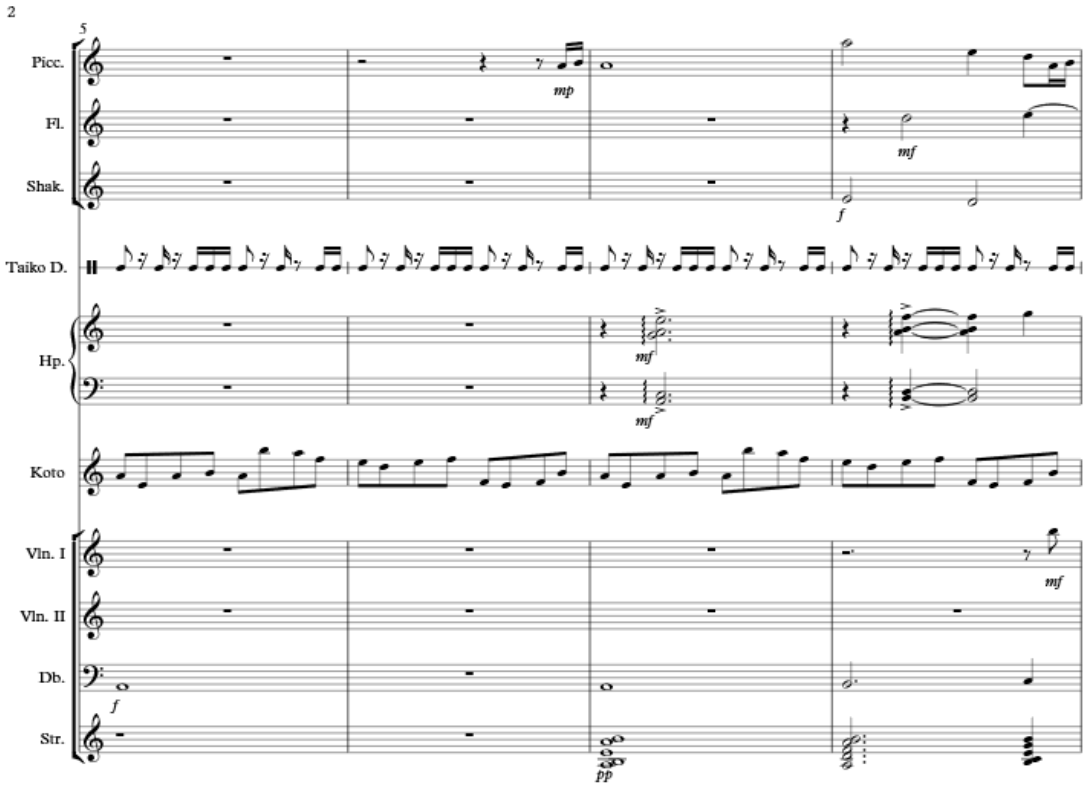

11
Picc.
Fl.
Shak.
pp
ppp
ppppp
Hp.
Koto
Shami.
Vln. I
> ppp
ppp
pppp
Vln. II
> ppp
ppp
pppp
Db.
> p
p
pppp
Str.
ppp
ppppp

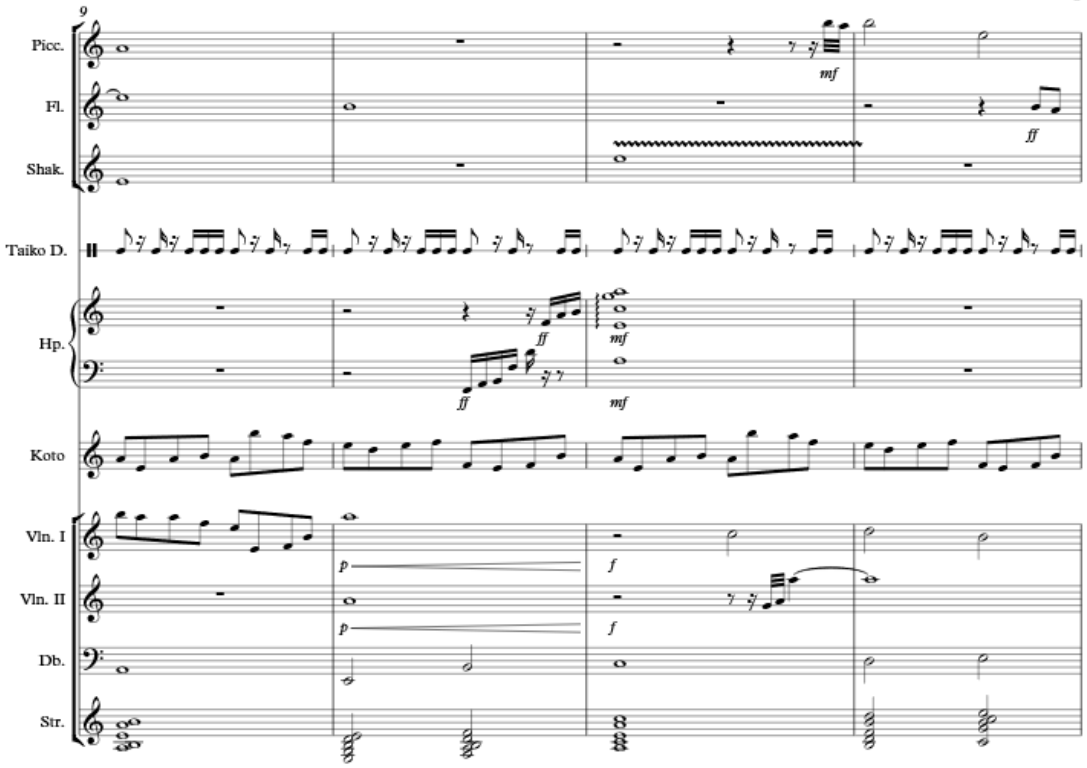
Hikari Hook

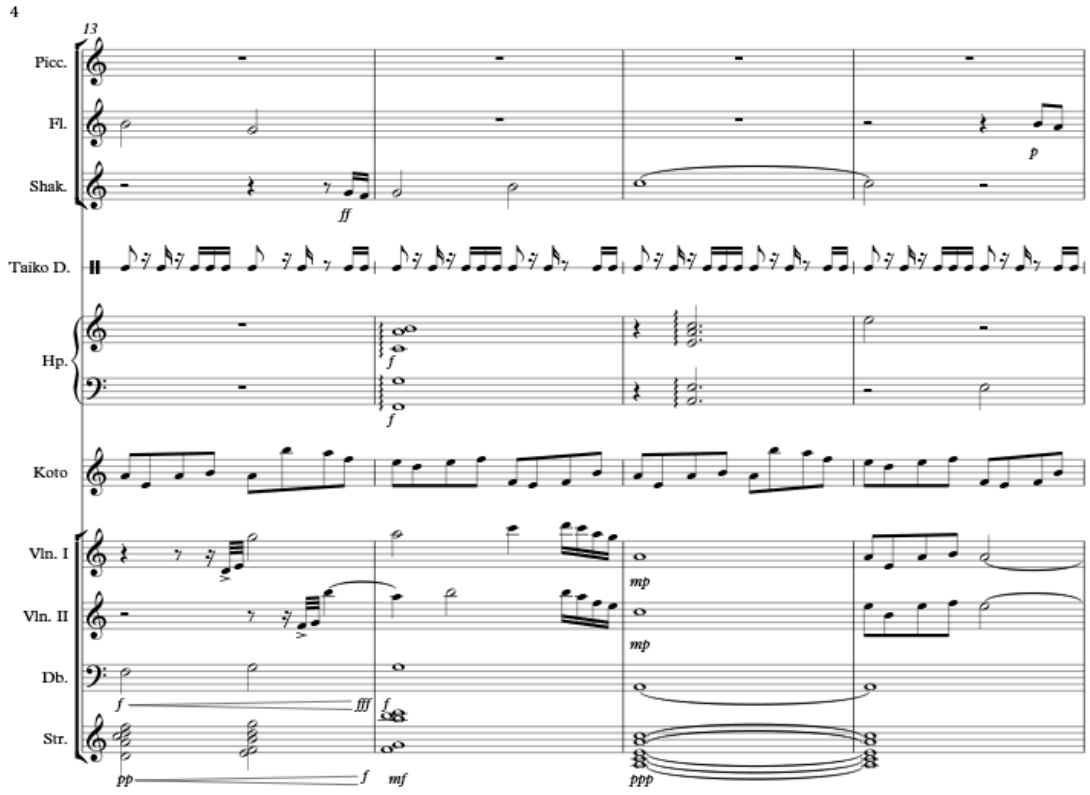
Level 1

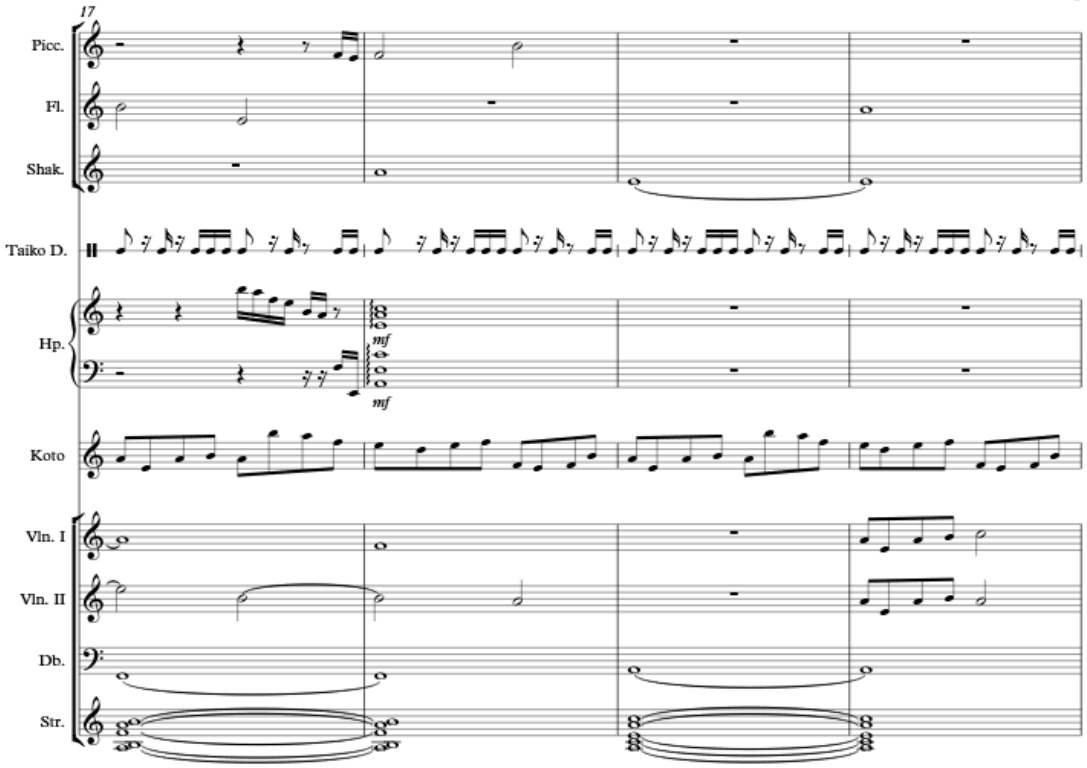
Christopher Knapp

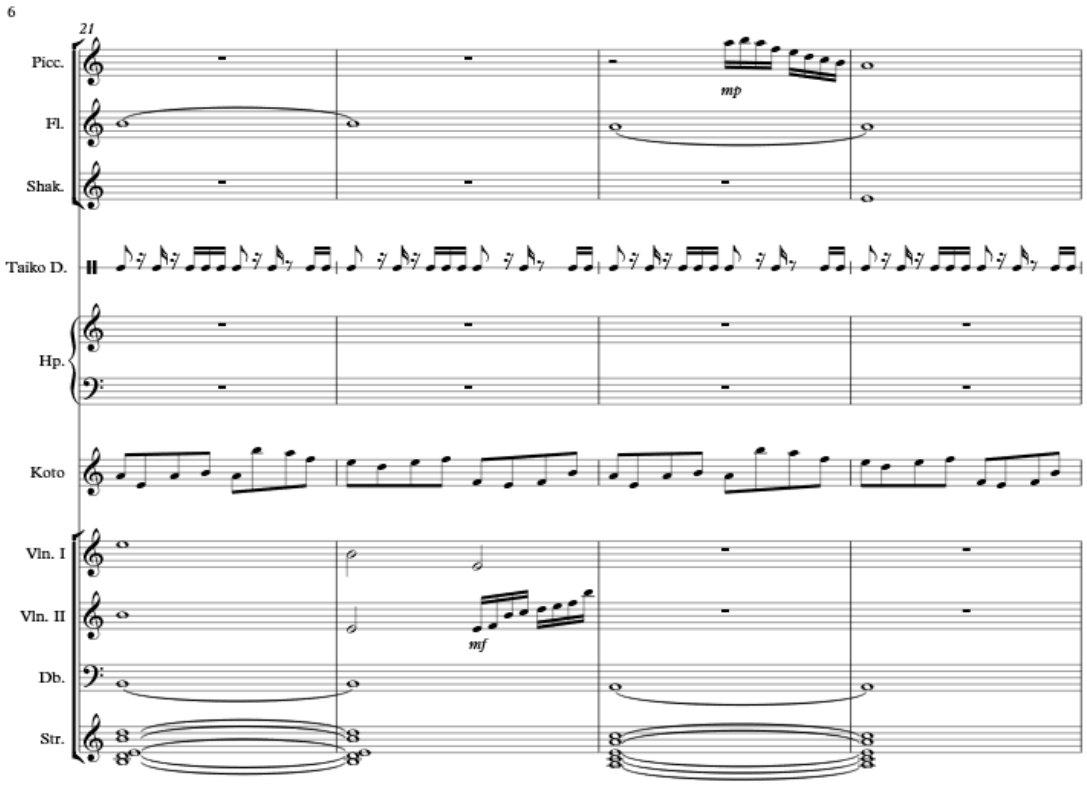


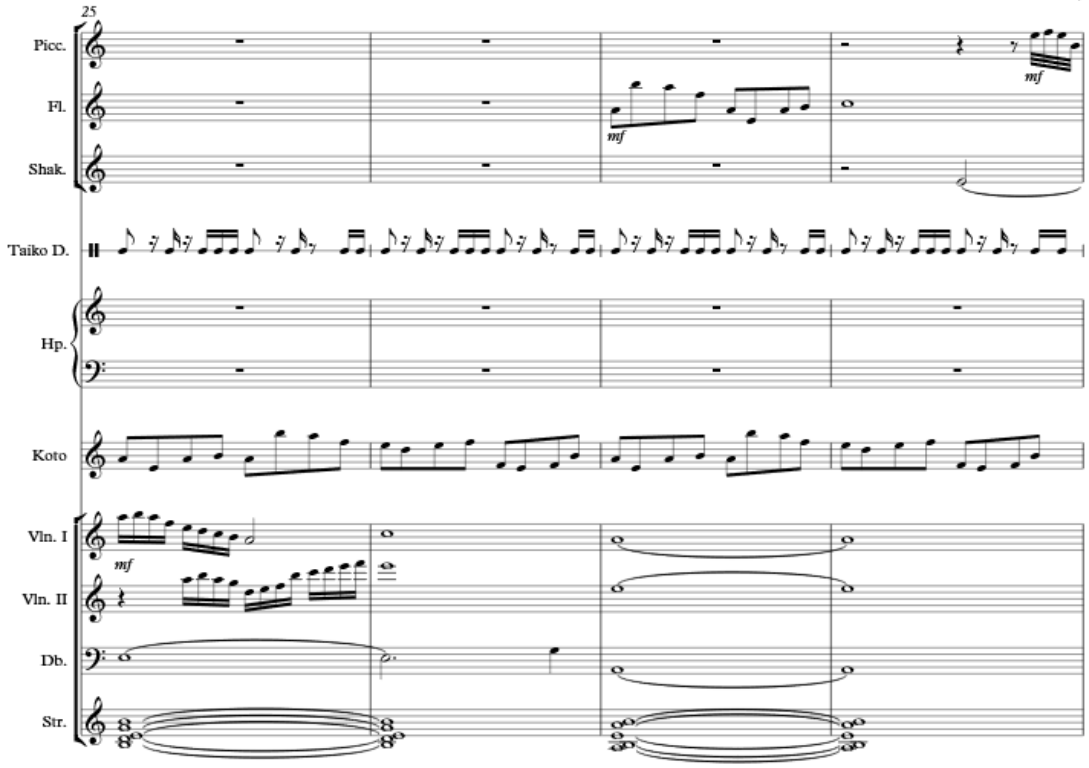

2
5
Picc.
Fl.
Shak.
Taiko D.
Hp.
Koto
Vln. I
Vln. II
Db.
Str.
mp
mf
f
mf
mf
mf
f
pp

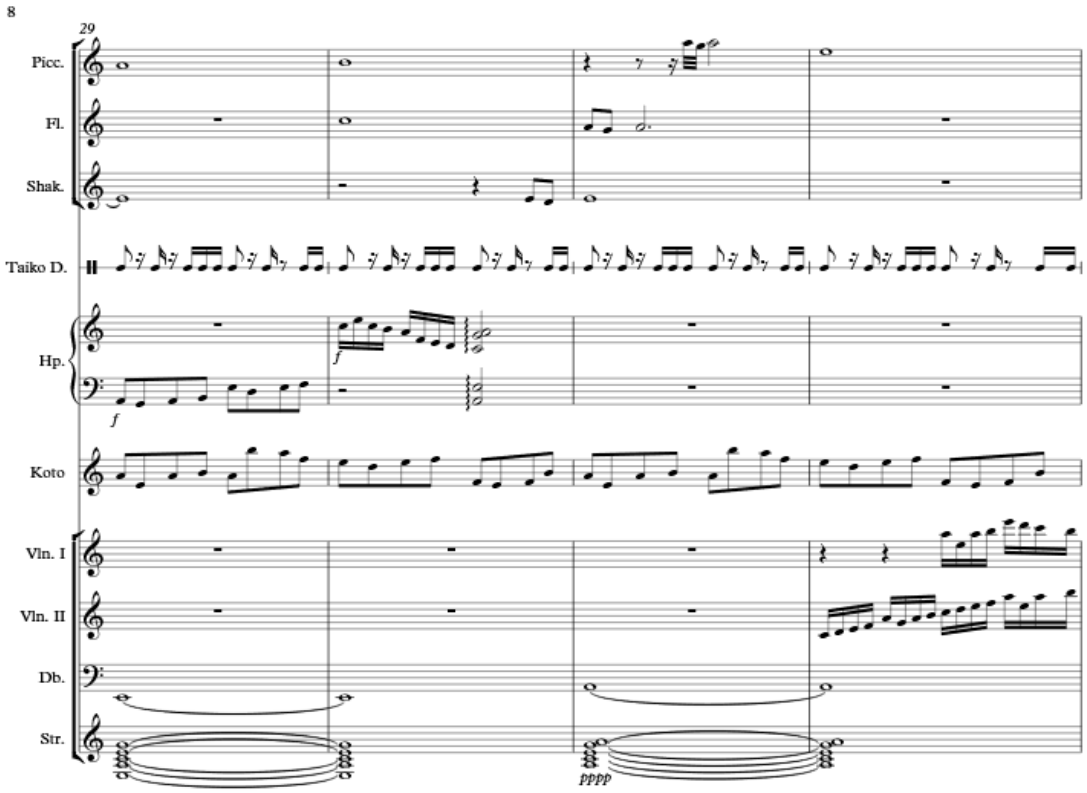

9
Picc.
Fl.
Shak.
Taiko D.
Hp.
Koto
Vln. I
Vln. II
Db.
Str.
mf
ff
ff
mf
mf
p
p
f
f

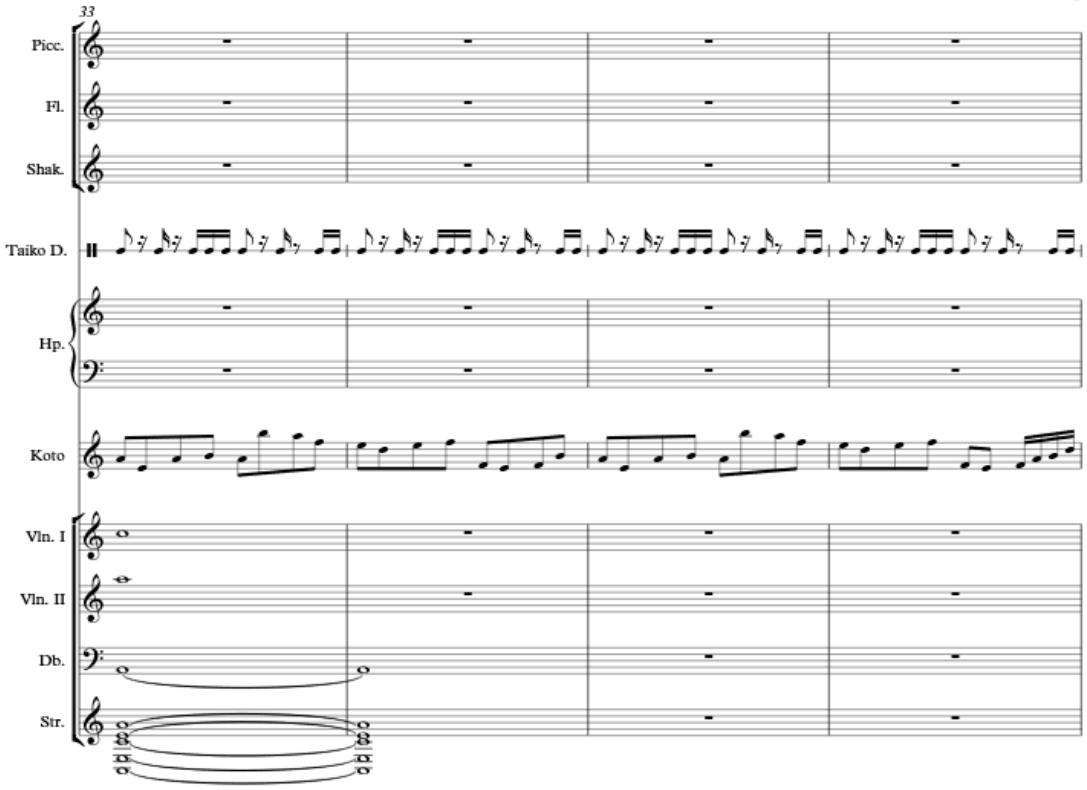

4
13
Picc.
Fl.
Shak.
Taiko D.
Hp.
Koto
Vln. I
Vln. II
Db.
Str.
ff
p
f
f
mp
mp
f
fff
pp
f
mf
ppp

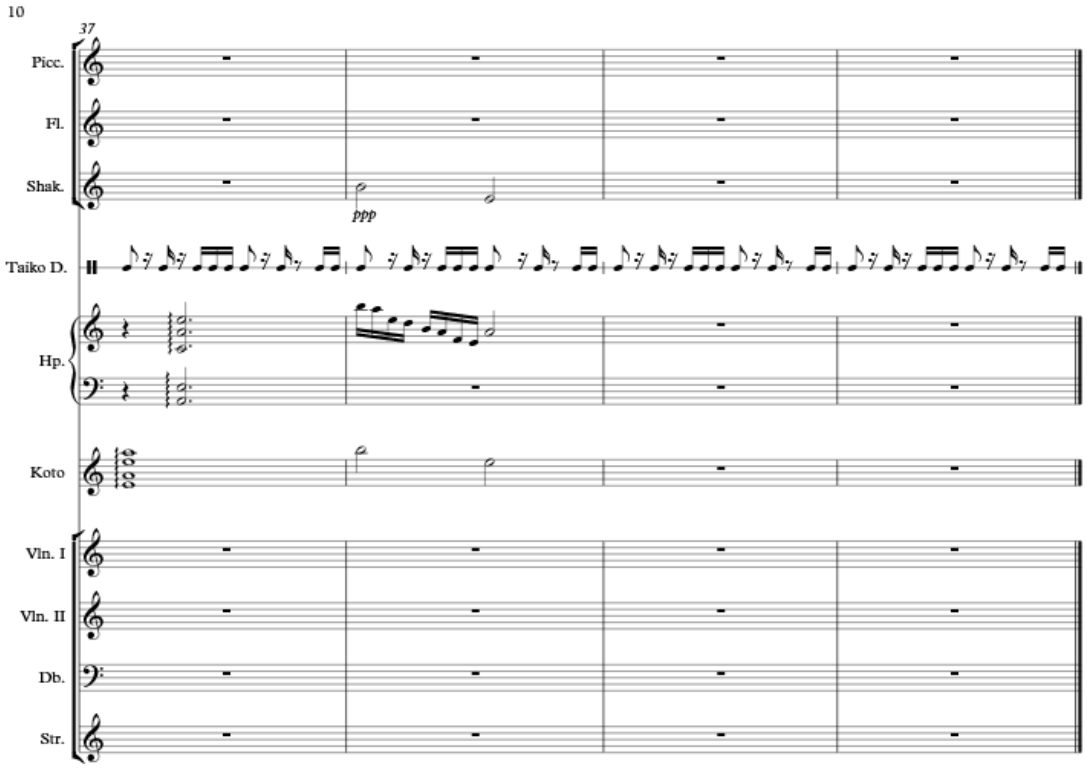

17
Picc.
Fl.
Shak.
Taiko D.
Hp.
mf
mf
Koto
Vln. I
Vln. II
Db.
Str.


6
21
Picc.
Fl.
Shak.
Taiko D.
Hp.
Koto
Vln. I
Vln. II
Db.
Str.
mp
mf


25
Picc.
Fl.
Shak.
Taiko D.
Hp.
Koto
Vln. I
Vln. II
Db.
Str.
mf
mf
mf


8
29
Picc.
Fl.
Shak.
Taiko D.
Hp.
f
f
Koto
Vln. I
Vln. II
Db.
Str.
pppp

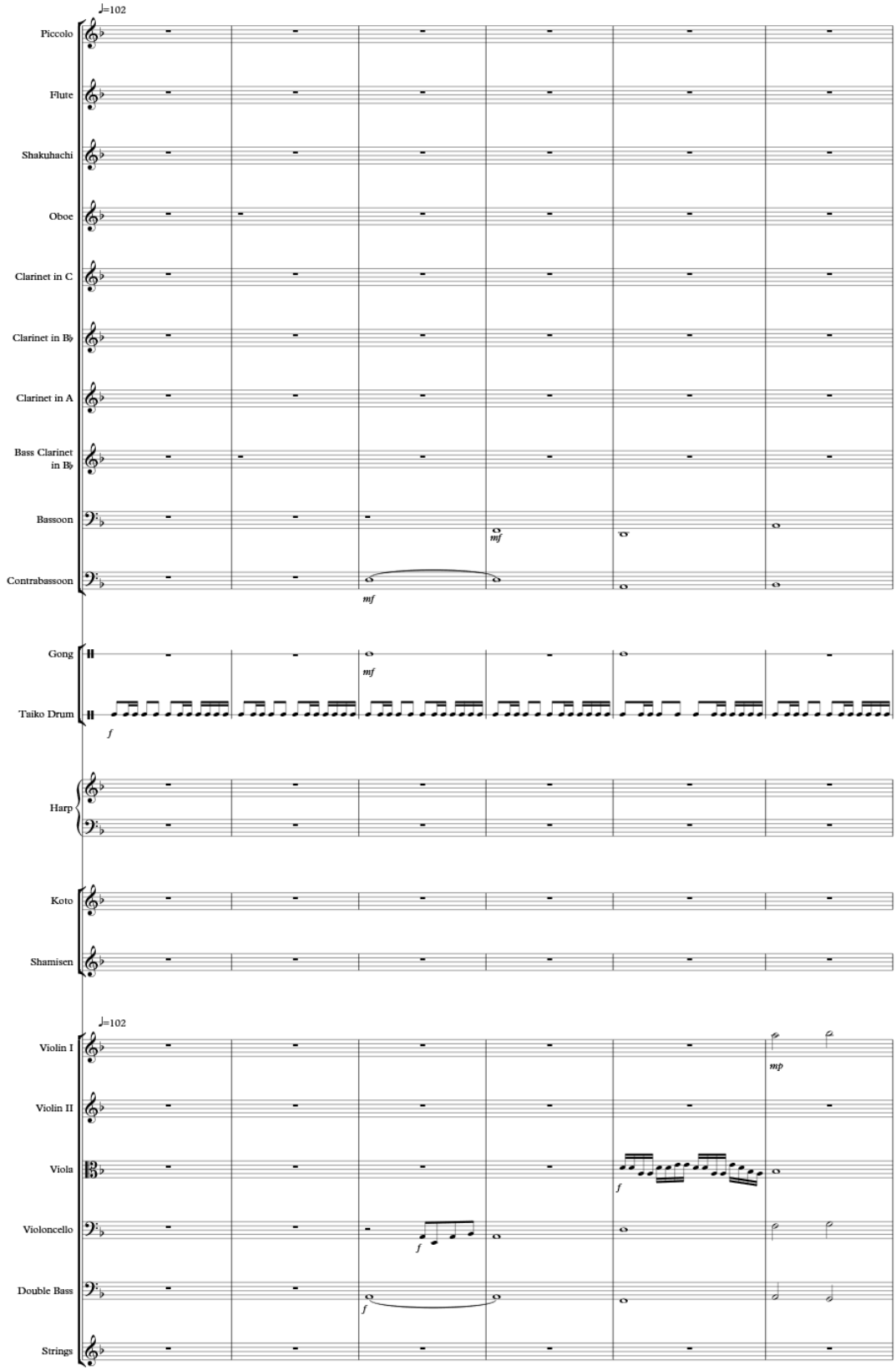

33
Picc.
Fl.
Shak.
Taiko D.
Hp.
Koto
Vln. I
Vln. II
Db.
Str.

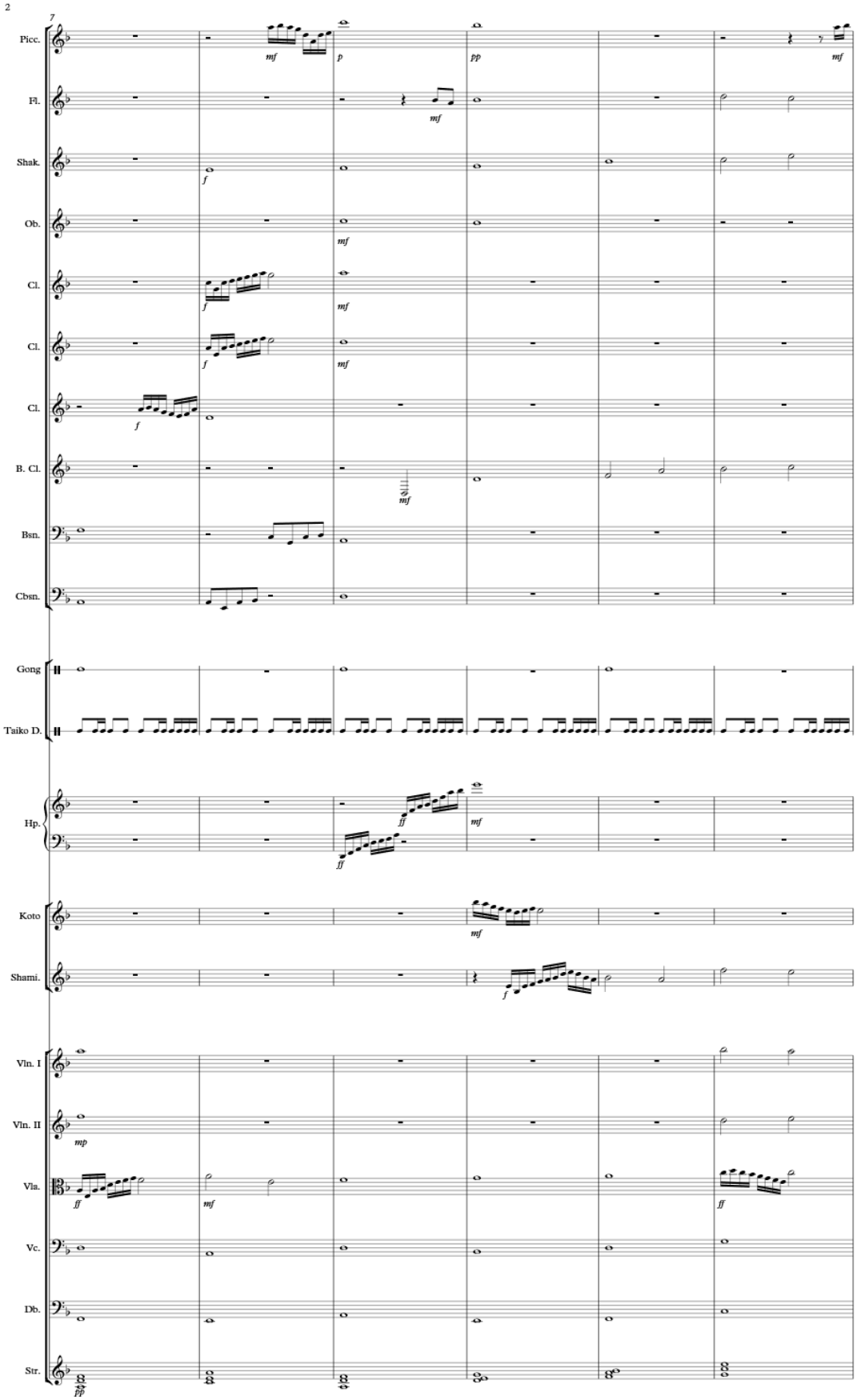

10
37
Picc.
Fl.
Shak.
ppp
Taiko D.
Hp.
Koto
Vln. I
Vln. II
Db.
Str.

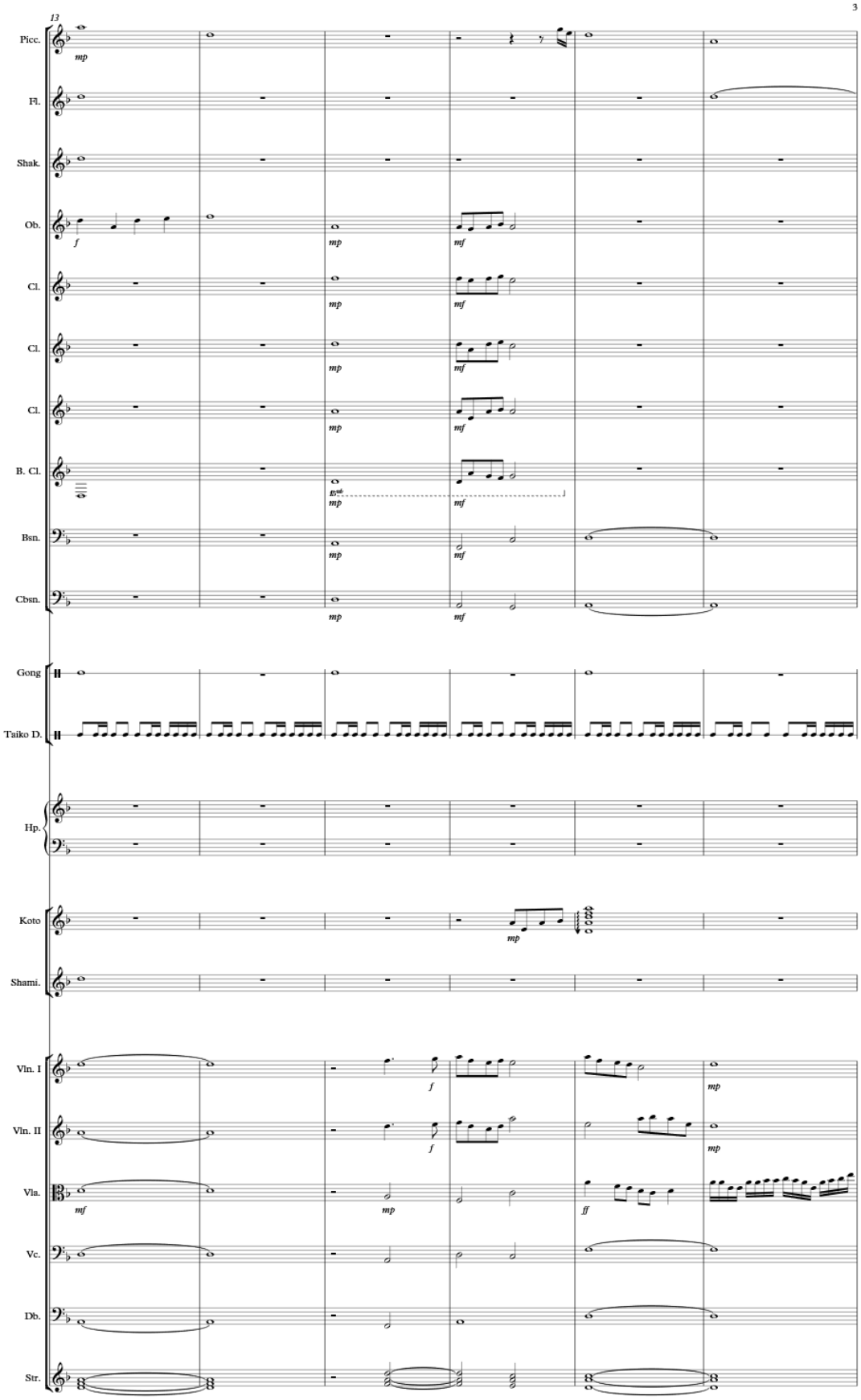
Hikari Hook

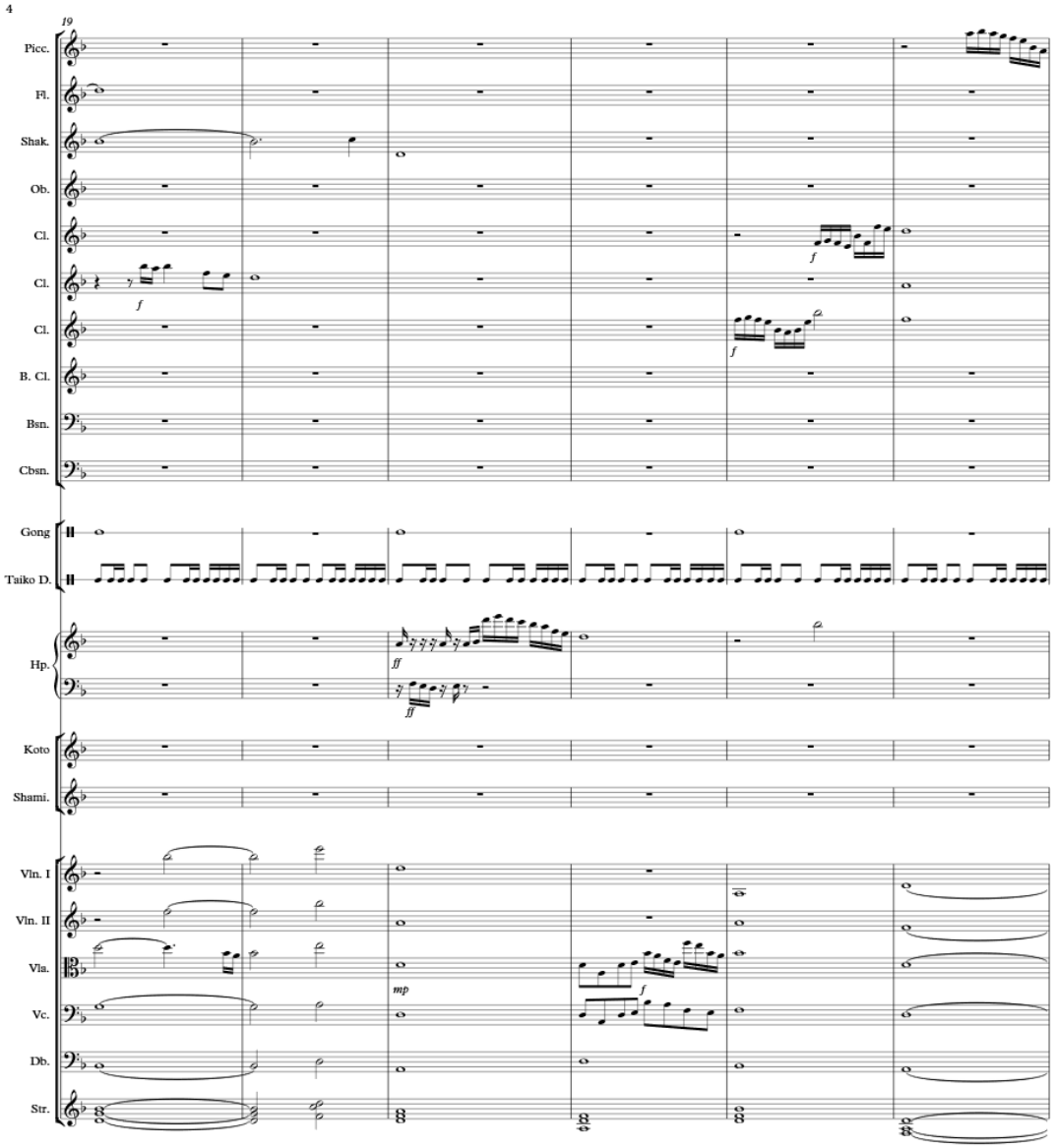
Level 2

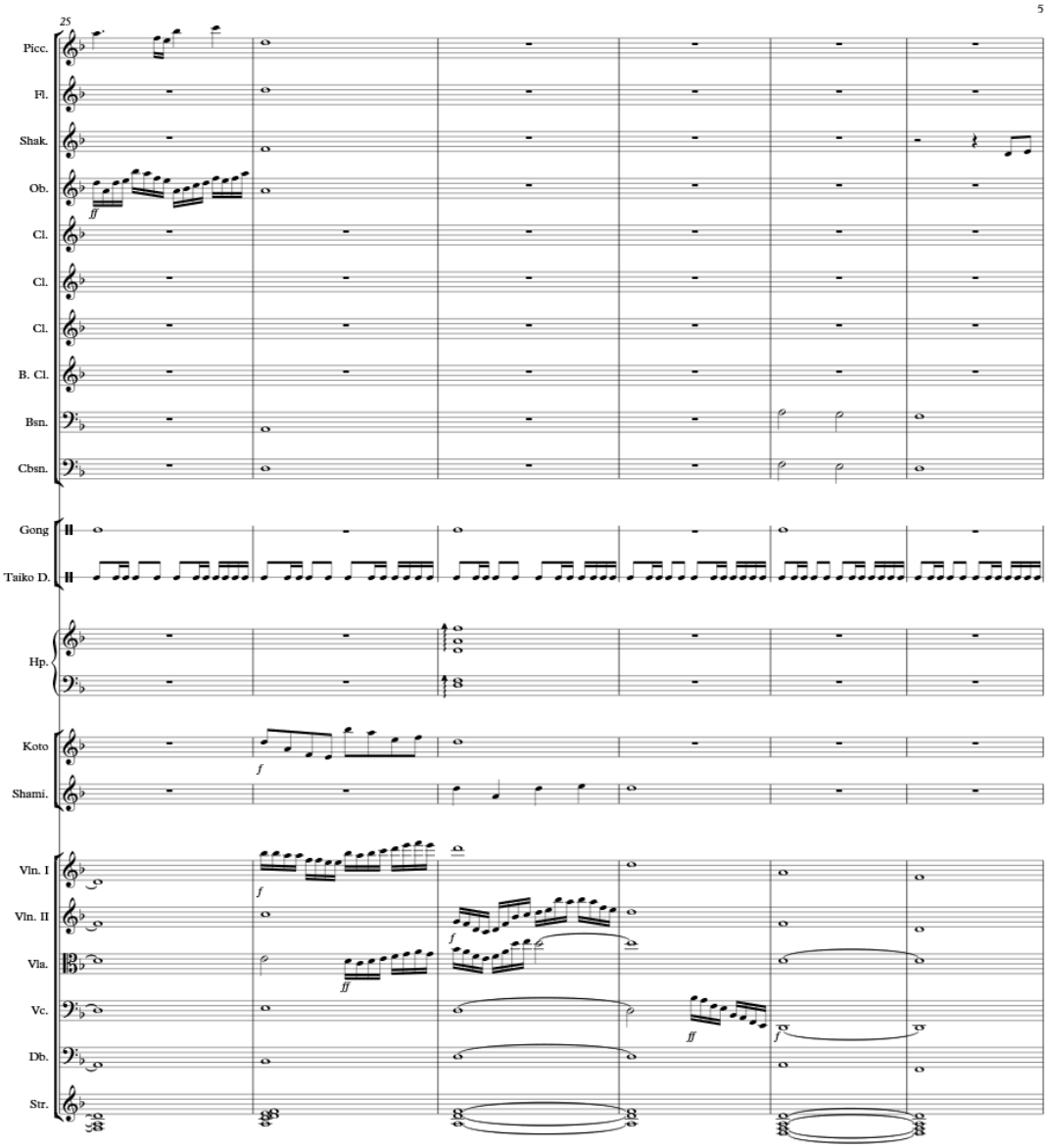
Christopher Knapp

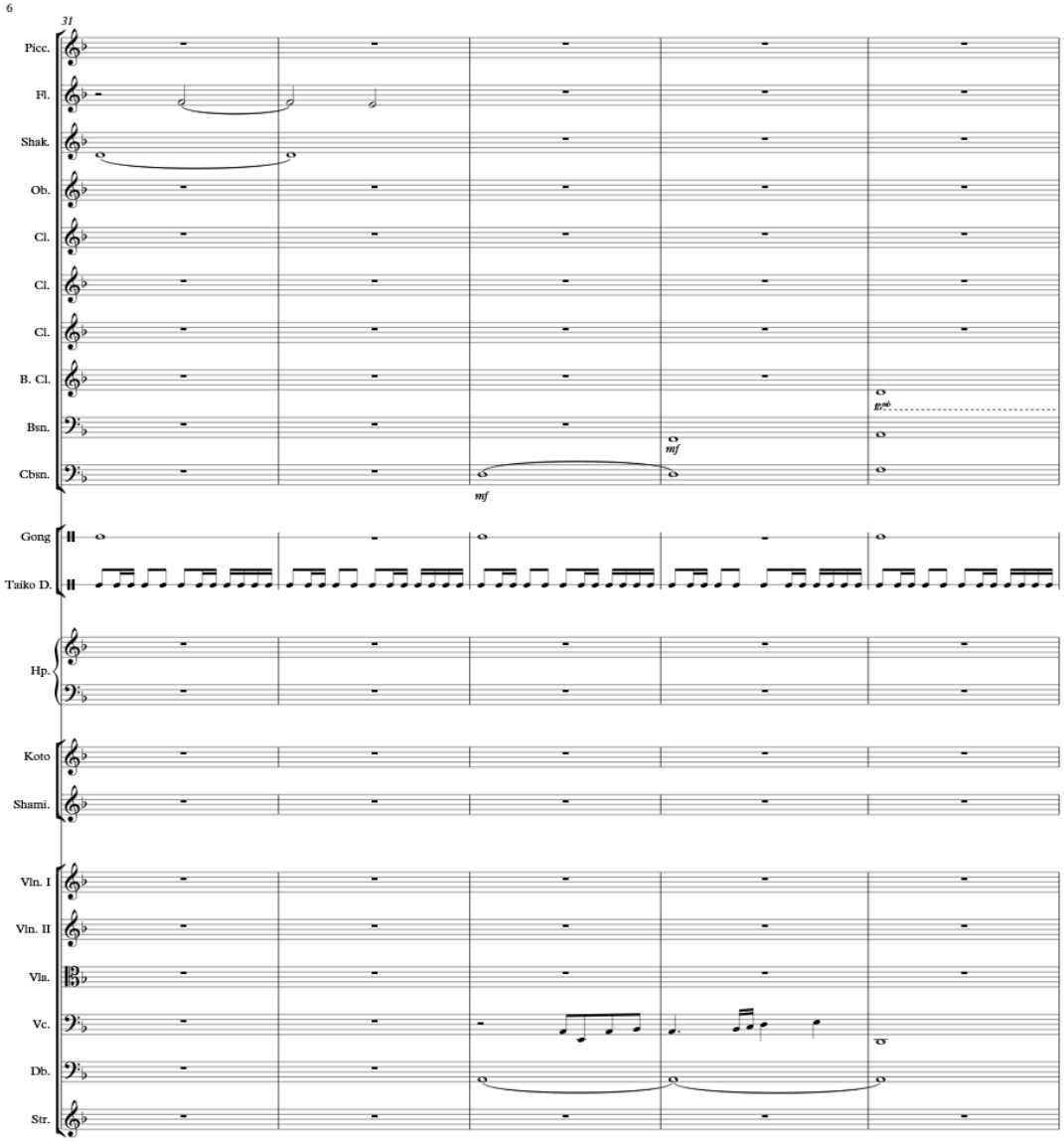


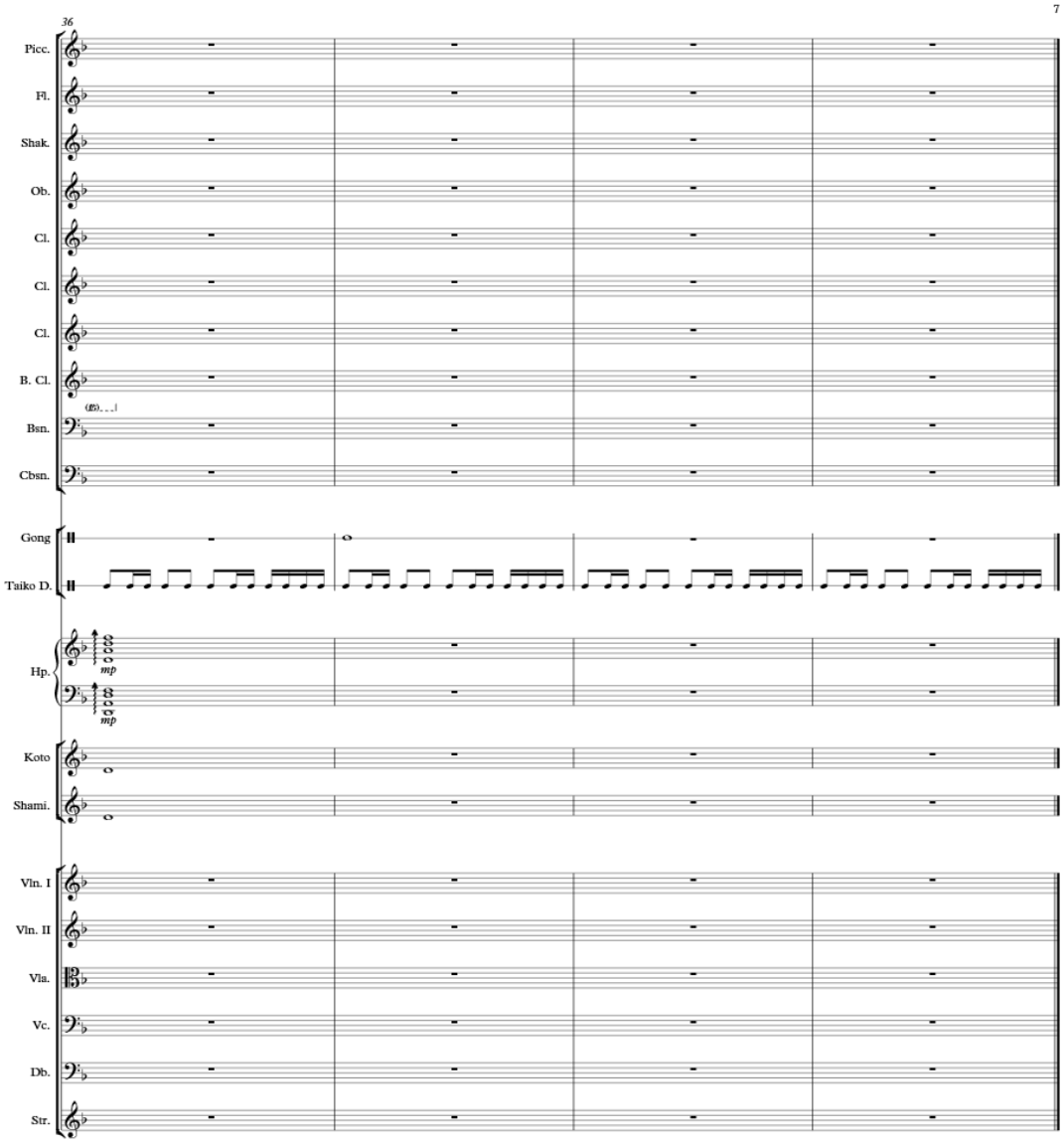

7
Picc.
Fl.
Shak.
Ob.
Cl.
Cl.
Cl.
B. Cl.
Bsn.
Cbsn.
Gong
Taiko D.
Hp.
Koto
Shami.
Vln. I
Vln. II
Vla.
Vc.
Db.
Str.


13
Picc.
Fl.
Shak.
Ob.
Cl.
Cl.
Cl.
B. Cl.
Bsn.
Cbsn.
Gong
Taiko D.
Hp.
Koto
Shami.
Vln. I
Vln. II
Vla.
Vc.
Db.
Str.
mp
f
mf
ff


19
Picc.
Fl.
Shak.
Ob.
Cl.
Cl.
Cl.
B. Cl.
Bsn.
Cbsn.
Gong
Taiko D.
Hp.
Koto
Shami.
Vln. I
Vln. II
Vla.
Vc.
Db.
Str.
f
f
f
ff
ff
mp
f


25
Picc.
Fl.
Shak.
Ob.
Cl.
Cl.
Cl.
B. Cl.
Bsn.
Cbsn.
Gong
Taiko D.
Hp.
Koto
Shami.
Vln. I
Vln. II
Vla.
Vc.
Db.
Str.
ff
f
f
ff
f
ff

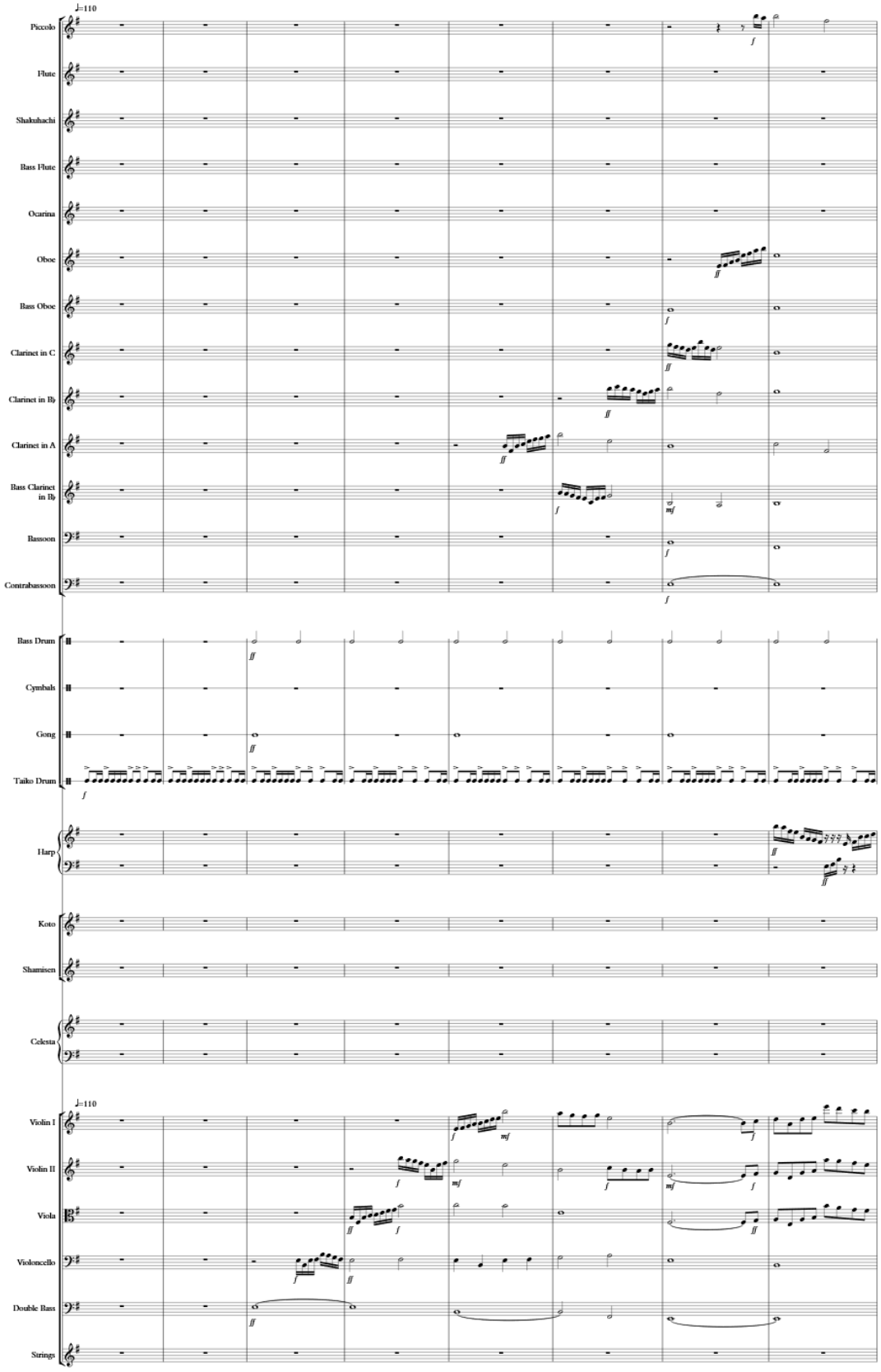

31
Picc.
Fl.
Shak.
Ob.
Cl.
Cl.
Cl.
B. Cl.
Bsn.
Cbsn.
mf
mf
Gong
Taiko D.
Hp.
Koto
Shami.
Vln. I
Vln. II
Vla.
Vc.
Db.
Str.

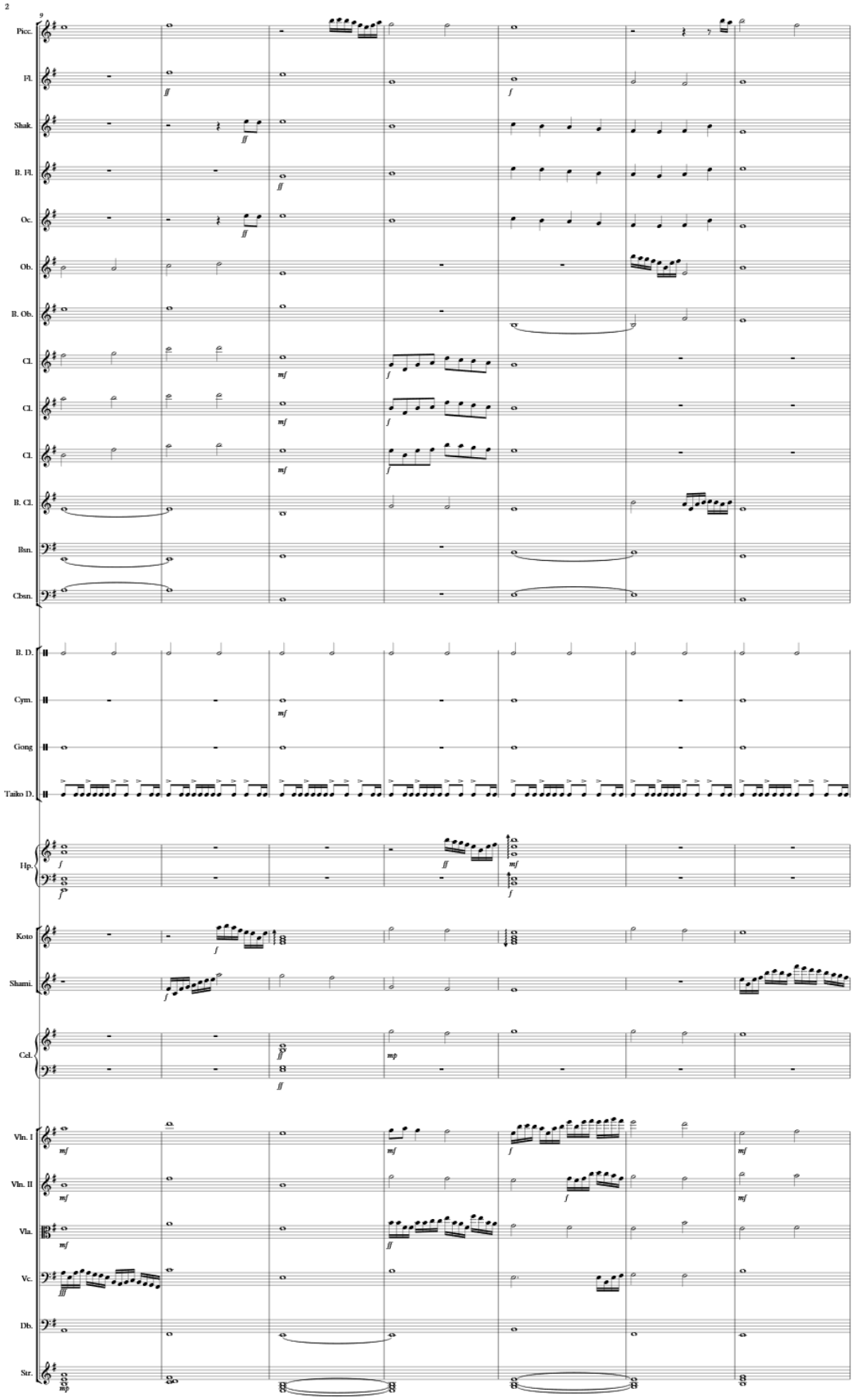

36
Picc.
Fl.
Shak.
Ob.
Cl.
Cl.
Cl.
B. Cl.
Bsn.
Cbsn.
Gong
Taiko D.
Hp.
mp
mp
Koto
Shami.
Vln. I
Vln. II
Vla.
Vc.
Db.
Str.

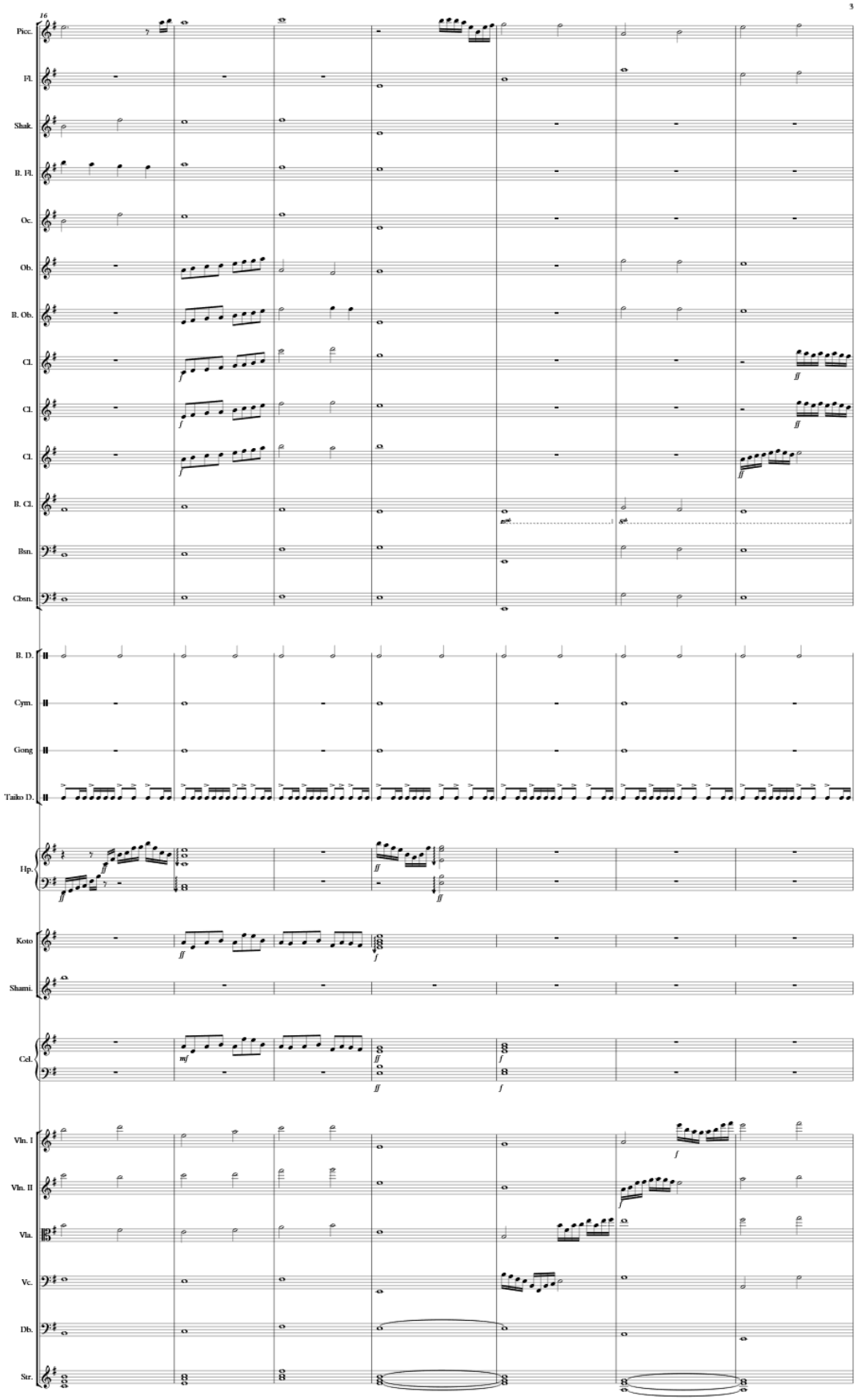
Hikari Hook

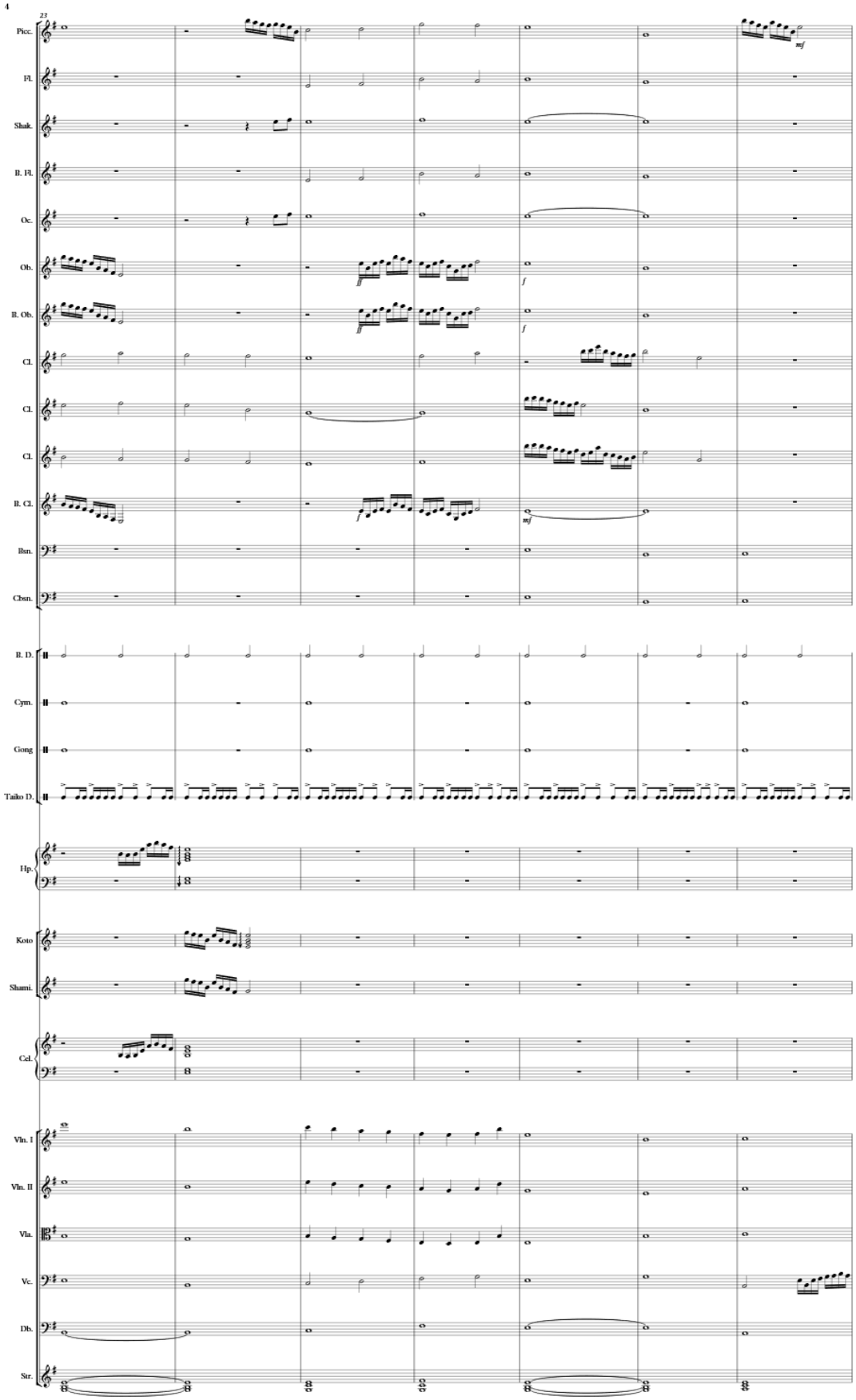
Level 3

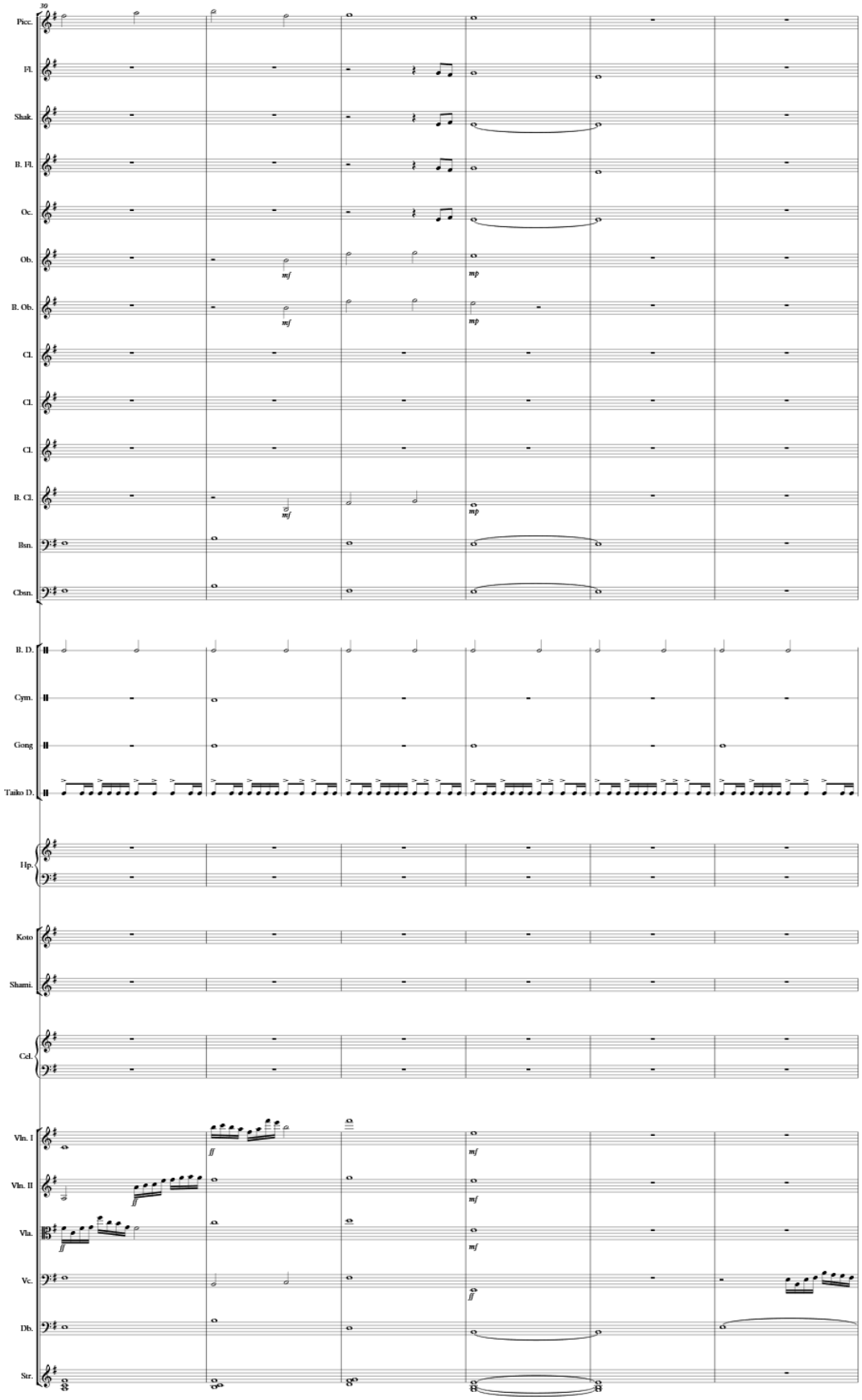
Christopher Knapp

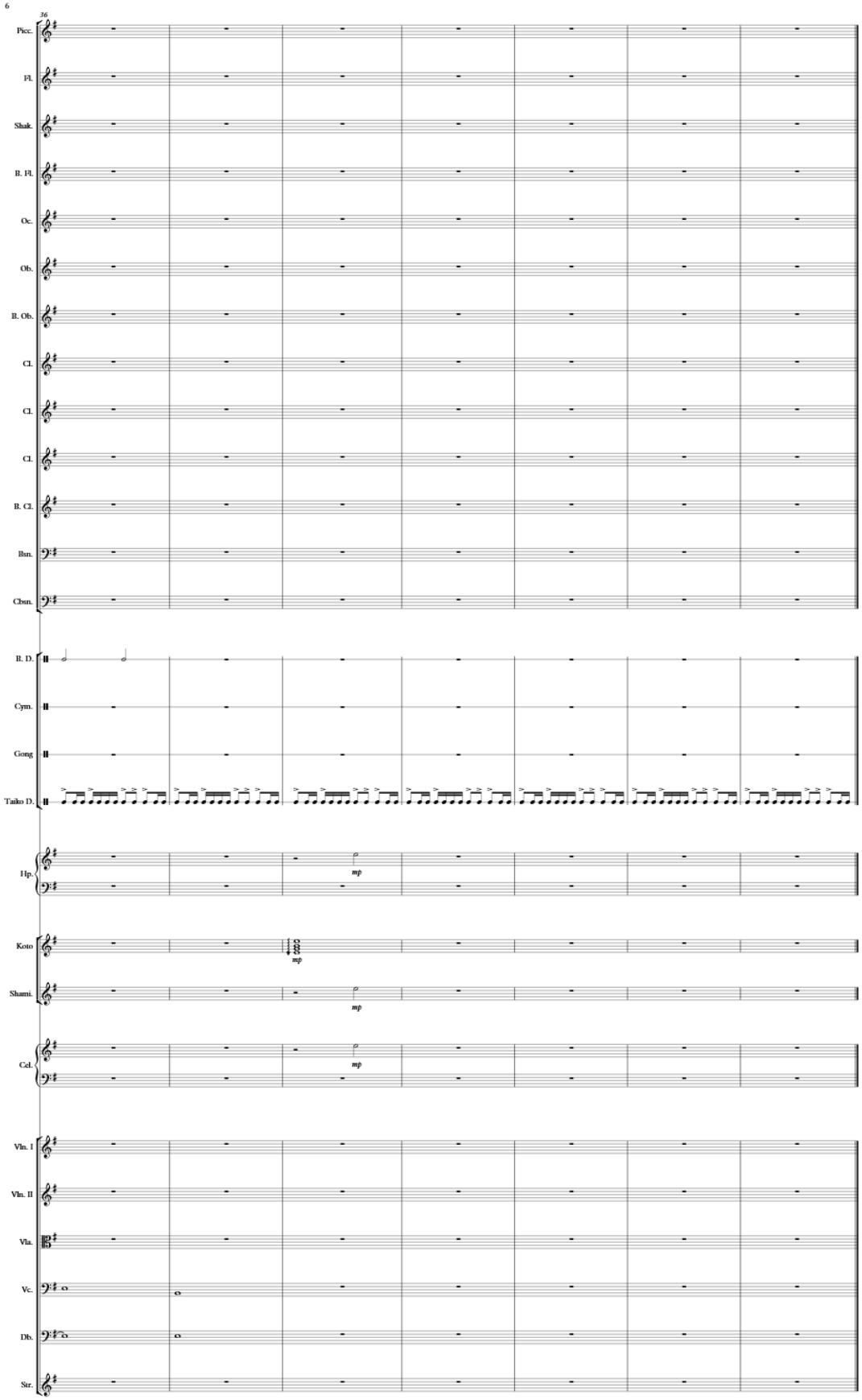



9
Picc.
Fl.
Shak.
B. Fl.
Oc.
Ob.
B. Ob.
Cl.
Cl.
Cl.
B. Cl.
Bsn.
Cbsn.
B. D.
Cym.
Gong
Taiko D.
Hp.
Koto
Shami.
Cel.
Vln. I
Vln. II
Vla.
Vc.
Db.
Str.


16
Picc.
Fl.
Shak.
B. Fl.
Oc.
Ob.
B. Ob.
Cl.
Cl.
Cl.
B. Cl.
Bsn.
Cbsn.
B. D.
Cym.
Gong
Taiko D.
Hp.
Koto
Shami.
Cel.
Vln. I
Vln. II
Vla.
Vc.
Db.
Str.


23
Picc.
Fl.
Shak.
B. Fl.
Oc.
Ob.
B. Ob.
Cl.
Cl.
Cl.
B. Cl.
Bsn.
Cbsn.
B. D.
Cym.
Gong
Taiko D.
Hp.
Koto
Shami.
Cel.
Vln. I
Vln. II
Vla.
Vc.
Db.
Str.
mf
f
ff
f
mf
f

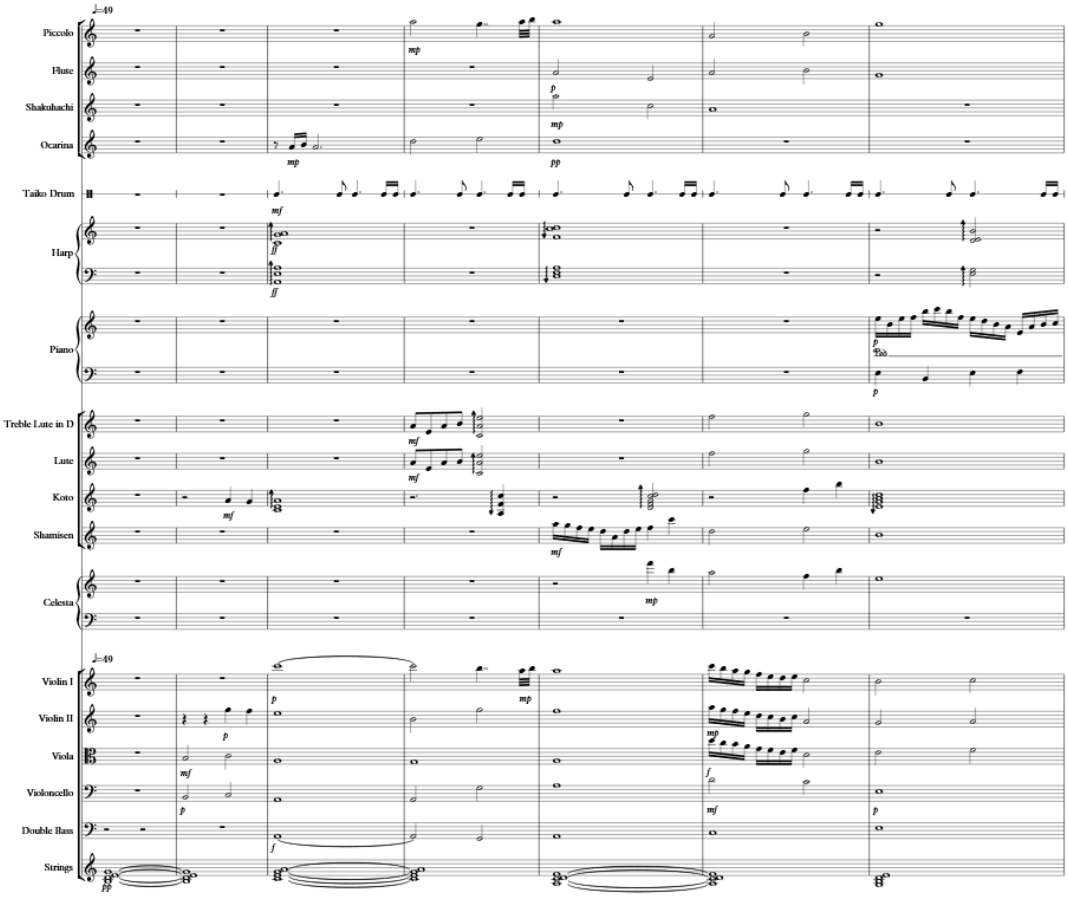

39
Picc.
Fl.
Shak.
B. Fl.
Oc.
Ob.
mf
mp
B. Ob.
mf
mp
Cl.
Cl.
Cl.
B. Cl.
mf
mp
Bsn.
Cbsn.
B. D.
Cym.
Gong
Taiko D.
Hp.
Koto
Shami.
Cel.
Vln. I
ff
mf
Vln. II
ff
mf
Vla.
ff
mf
Vc.
ff
Db.
Str.

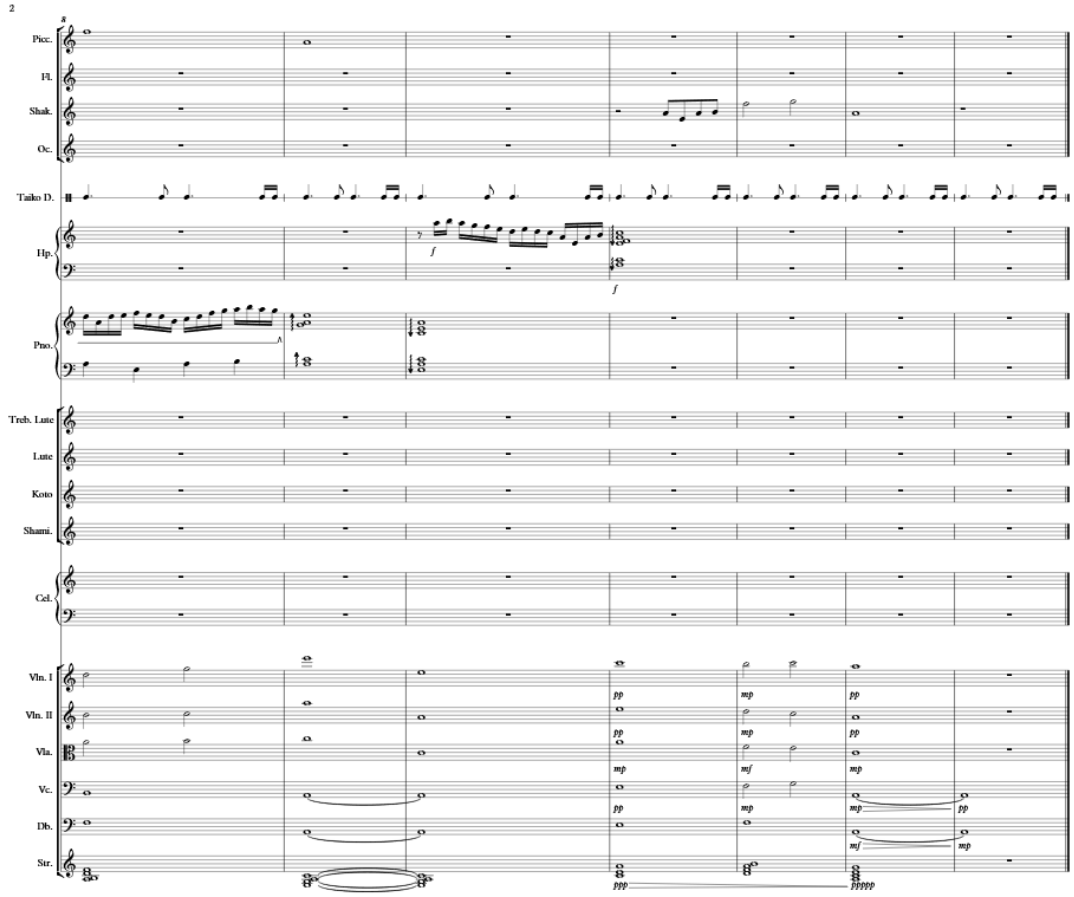

36
Picc.
Fl.
Shak.
B. Fl.
Oc.
Ob.
B. Ob.
Cl.
Cl.
Cl.
B. Cl.
Bsn.
Cbsn.
B. D.
Cym.
Gong
Taiko D.
Hp.
mp
Koto
mp
Shami.
mp
Cel.
mp
Vln. I
Vln. II
Vla.
Vc.
Db.
Str.

Hikari Hook

Finding

Christopher Knapp




Picc.
Fl.
Shak.
Oc.
Taiko D.
Hp.
Pno.
Treb. Lute
Lute
Koto
Shami.
Cel.
Vln. I
Vln. II
Vla.
Vc.
Db.
Str.