Hikari Michi: Studying Immersion in Games Using the Oculus Rift and Leap Motion Controller

---

Christian Manuel, Stone Cleven, Adam Degenhardt, Nick Mollica

November 12, 2014

A Major Qualifying Project Report:
submitted to the Faculty of the

WORCESTER POLYTECHNIC INSTITUTE

in collaboration with
OSAKA UNIVERSITY

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

Approved by:

Professor Robert Lindeman, Advisor

Abstract

Game developers are always searching for new, creative ways to immerse players into their games, whether it be through gameplay, input, or output. As aspiring game developers ourselves, we pursued this project with the goal of building a game that made an enriching and engaging experience through its gameplay, input, and output. In this paper, we will go into detail about all aspects of our project which includes the gameplay, mechanics, hardware, software, design process, art, and sound. For this game, we made use of the Leap Motion hand and finger movement capturing device and the Oculus Rift virtual reality head mounted display alongside the C4 Game Engine. We will also look at the user studies we performed when we had volunteers playtest an early version of our game. Through this project, we hope to provide players with a fun, immersive game play experience and hopefully be a source of inspiration for virtual reality game developers in the future.

Acknowledgements

We would like to thank Worcester Polytechnic Institute, Osaka University, the Takemura Lab and everyone in it, our advisors: Professors Lindeman and Kiyokawa, Terathon Software, and the members of the C4 Engine forum for making this project possible.

Contents

List of Figures

List of Tables

1 Introduction

Video game hardware is becoming more advanced with the releases of more powerful and sophisticated consoles and software. Developers are always aiming to create the most immersive experience that they can for players. Better immersion provides for a more fun, richer experience for those playing the game. Developers have used creative input devices, such as plastic instruments used in *Guitar Hero* or *Rock Band*, to provide a new and unique way to interact with the game and provide a whole new experience.

In relation to that, better immersion can also be achieved through output and graphical feedback to the player. Commercially successful, publicly available virtual reality equipment is starting to appear, and a VR game market will soon follow. We decided to design a game that would make full use of a unique input device, all the while capturing the player's full attention using head mounted virtual reality. During our stay at the Osaka, Japan MQP project site, we collaborated with Osaka University faculty and students and made use of the resources available to us to create a unique, immersive virtual reality gaming experience for users.

This report will describe the game we made for our MQP and the process we went through to make it. The Introduction section will outline the game concept and the goals we had when building it. The Hardware and Input section will talk about the input devices we used and how they control gameplay. We will discuss the software we used to develop our game under the Software section. In Related Work and Research, we will discuss the research we did directly related to our project, as well as any similar works we drew inspiration from. The Design Process section includes a discussion of our design goal for the game and how we pursued them. Under Game Mechanics and Gameplay, we will discuss in depth the inner mechanics of the game and the user experience. The Programming section will include an overview of the game code's structure, and details about some of the more interesting and complicated programming problems we faced. The game's art and theme will be discussed in Art and Themes. The Sound section will cover artistic and technical topics related to the game's sound. Project Management will include a description of the project management techniques and tools we used. User Studies will discuss development testing with outside resources. The Post Mortem section includes what we can see, in hindsight, did and didn't go well during the development of the game. Lastly, we will outline the predicted timeline and dates we had for our project at the beginning, and the actual timeline and dates of development.

1.1 Project Goals

The main goal of our project was to create a unique and immersive virtual reality gaming experience for players using the Oculus Rift and Leap Motion. We planned to make full use of all of our technical and artistic skills, while gaining valuable new knowledge and skills. Since none of us had previously developed for virtual reality, this provided us with a unique opportunity to work with a new technology and was a great learning experience.

1.2 The Game

The main idea for the game is a three-dimensional, first-person action game. The core concept is that the player is "surfing" along a path of solid light that is dynamically created in front of them. Movement is automatic on the path and the player's speed is determined by a number of factors including vertical movement, speed-increasing pickups, and speed-reducing obstacles. The player controls the direction and height of the path by using their hand in conjunction with the Leap Motion. Using the head orientation tracking function of the Oculus Rift, the player can look around the world for objectives and obstacles.

The goal of the game is to complete each level as quickly as possible. The player's best time on each level is recorded and visible on the main menu. Each level is laid out like an obstacle course, with a variety of beneficial pickups and obstacles that can make the player lose speed or fail the level. The path that the player creates persists in the world until the level is completed, so the player must be careful not to run into the additional obstacle presented by his/her path.

2 Hardware and Input

The game is playable on desktop computers using the Oculus Rift and Leap Motion as input. The stereoscopic view required to use the Oculus Rift is a feature built into the game software, so users do not need the Rift SDK to use it. However, the Leap Motion software needs to be installed to enable the computer to recognize the device.

2.1 Computer

Our project was developed on many different computers, each with different system specifications. The performance varied little from computer to computer, but we identified the computer which seemed to be ideal for smooth gameplay. Table 1 shows the specifications used in the laptop that showed the best performance.

Table 1: Hardware specifications of recommended computer for best performance of the completed game

Model	Sager
RAM	8 GB
Graphics Card	GeForce GTX 485m processor
Processor	Intel i7
OS	Windows 7 - 64 bit

2.2 Oculus Rift

The Oculus Rift is a virtual reality head mounted display (HMD), developed by Oculus VR, that allows the user to see games in stereoscopic 3D. Some specifications of the Rift DK1 are listed in Table 2.

Table 2: Some hardware specifications of the Oculus Rift. [6]

Resolution	1280x800 (640x800 per eye)
Panel type	LCD
Video input	DVI/HDMI
3D Input type	Side by Side + optical distortion
Field of view	110 degrees diagonal
Weight	220 grams
Head Tracking	Yes

The screen of the Rift Development Kit 1 has a 7 inch diagonal viewing area with 1280 x 800 resolution (720p). This resolution is split between both eyes, yielding 640 x 800 per eye. The lenses that come installed with the Rift are for users with 20/20 or farsighted vision. This can be adjusted with extra lenses that are included with the kit for nearsighted users. The display is a LCD panel which runs at 60Hz via DVI-D Single Link or HDMI 1.3+ cable.[8]

The Rift has head tracking features that track the user's head orientation, along with a field of view of 110 degrees diagonal. Inside the Rift, there is a three-axis gyroscope that senses angular velocity, a compass, and a three-axis accelerometer that senses all accelerations, including gravitational accelerations. All three of these measurement tools work together to provide precise, real-time head tracking. This is a very good feature for looking around in games and was a feature we designed around. For this project, the Rift allows the user to look around their environment. The Rift is also very good at enhancing player immersion because it gives the player a greater sense of being inside the virtual world.[8]

2.3 Leap Motion

The Leap Motion is a motion tracking device that can detect the user's hand and individual finger orientation and position. The Leap Motion Controller tracks all ten fingers up to 1/100th of a millimeter and has a wide 150 degree field of view and a Z-axis for depth. This lets the user move their hands in 3D motions, similar to the real world, without losing track of their hands. It tracks hand movements at a rate of over 200 frames per second, giving accurate real-time hand positions. The Leap Motion does have a limited range of detection (about 2 ft.), but we found that it meets the requirements for this game. More technical specifications can be seen in Table 3.

Table 3: Some hardware soecifications of the Leap Motion Controller. [7]

Height	0.5 inches
Width	1.2 inches
Depth	3 inches
Weight	0.1 pounds
Included Cables	24" and 60" USB 2.0 (microUSB 3.0 connectors)
OS Required	Windows 7 or 8 or Mac OS X 10.7 Lion
Processor Required	AMD Phenom II or Intel Core i3, i5, i7 processor
Memory Required	2 GB RAM
Other Requirements	USB 2.0 port and Internet connection

The Leap Motion controller uses optical sensors and infrared light to detect movement. The sensors are directed along the y-axis-upward when the controller is in its standard operating position. The effective range of the Leap Motion Controller extends from approximately 25 to 600 millimeters above the device. The controller is able to do its best detection and tracking when the controller has a clear, high-contrast view of the hand's silhouette. The software combines the sensor data with an internal model of the human hand to help cope with challenging or difficult tracking conditions.[3]

3 Software

A variety of software applications were used to make the game. Autodesk Maya was the sole program used for model creation, rigging and animation. Photoshop was used for concept art, texturing and creation of other texture maps. To interface the game with the Leap Motion Controller, we used the native interface Leap Motion API available through the Leap Motion website. We also used the Oculus Rift SDK 1 to enable our computers for Rift compatibility. For sound creation, a variety of software was used including: Finale Printmusic, Cakewalk's Music Creator 6 Touch, and Audacity.

3.1 C4 Game Engine

We used the C4 engine to build the game. The C4 engine is compatible with the Oculus Rift SDK and Leap Motion C++ API, both of which were required for the game. We chose C4 because it was readily available to us and met the requirements for our game. For technical details about the engine, see the Programming section.

3.2 Art and Animation

Autodesk Maya, a 3D computer graphics software currently owned by Autodesk, Inc., was used to create all of the models for all 3D art assets. Additionally, Autodesk Maya was used for rigging and animations for the player character and game objects. Adobe Photoshop, a raster graphics editor developed by Adobe Systems, was used for concept art, texturing, and other texture maps.

3.3 Oculus SDK

There are no specific hardware requirements to use the Oculus SDK, but it is recommended that computers that use it have a modern graphics card. A recommended benchmark test is to run Unity at 60 frames per second with vertical sync and stereo 3D enabled. If the application runs without dropping frames then the system configuration is sufficient for Oculus Rift Development. More recommended guidlines are shown in table 4.[1]

Table 4: Some computer hardware specifications for the Oculus Rift SDK. [1]

Windows	7, 8, or 8.1
MacOS	10.8+
Linux	Ubuntu 12.04 LTS
Processor	2.0 GHz processor
RAM	2 GB
Graphics Card	Direct3D10 or OpenGL3 compatible card

The Oculus Rift's internal coordinate system uses Euler angles to determine the orientation of the user's head. Euler angles describe the three angles used to describe the orientation of an object: pitch, roll, and yaw. Figure 1 shows how the coordinate system relates to the user's head orientation.

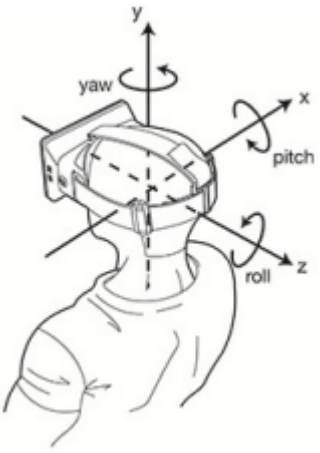


Figure 1: Oculus Rift Coordinate system [1]

The pitch describes rotation around the x-axis, the yaw is rotation around the y-axis, and roll is rotation around the z-axis. The Oculus Rift coordinate system uses left-handed notation, meaning the x-positive goes to the right and the z-positive goes towards the player.

3.4 Leap Motion API and System Architecture

The Leap Motion system is able to recognize and track hands, fingers, and finger-like tools or objects. The device operates in close proximity with high precision and tracking frame rate and reports discrete positions, gestures, and motion.

The Leap Motion software runs as a service on Windows or Daemon on Mac and Linux. It connects to the Leap Motion Controller device via the USB bus. Leap-enabled applications access the Leap Motion service to receive motion tracking information. The Leap Motion SDK had two varieties of API for getting Leap Motion data: a native interface and a WebSocket interface. Since our game is a local client, we focused primarily on the native interface API. Using these APIs, you are able to create Leap-enabled applications in several programming languages, including: Javascript, Unity/C-Sharp, C++, Java, Python, and Objective-C. Our game only needed to use the C++ libraries.

The Leap's native application interface is provided through a dynamically loaded library. This library connects to the Leap Motion service and provides tracking data to whichever application it is linked to. The library can be linked directly in C++ and Objective-C applications, or through one of the language bindings provided for Java, C-Sharp, and Python. More information on how the process works can be seen in figure 2.[2]

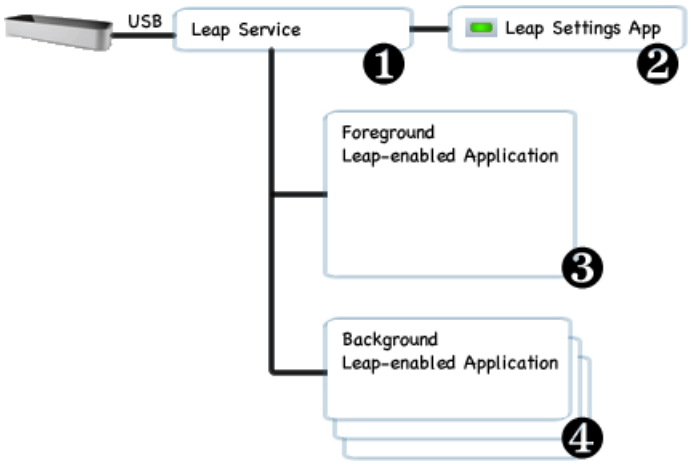


Figure 2: Native Application pipeline process [2]

In the first step, the Leap Motion service receives data from the Leap Motion Controller over USB. Here, it processes that information and sends it to running Leap-enabled applications. By default, the service only sends tracking data to the foreground application. However, applications can request that they receive data in the background. This request can be denied by the user. By default, the C4 game application, when running, is enabled to receive any information from the Leap Motion Controller, whether it is in the foreground or background.

At step two, the Leap Motion application runs separately from the service and allows the computer user to configure their Leap Motion installation. The Leap Motion application

is a Control Panel applet on Windows that comes included with the SDK.

During step three, the foreground Leap-enabled application receives motion tracking data from the service. Any Leap-enabled application can connect to the Leap Motion service using the Leap Motion native library, as stated above. When a Leap-enabled application loses the OS focus, the Leap Motion service stops sending information to it. Applications intended to work in the background can request permission to receive data even when in the background. When in the background, the configuration settings are determined by the foreground application.[2]

The Leap Motion system uses a right-handed Cartesian coordinate system for positions, as seen in figure 3. The origin is centered at the top-middle of the device. The x- and z-axes lie on a horizontal plane, with the x-axis running parallel along the long edge of the controller. The y-axis is the vertical 3D axis, with positive values increasing upwards. The z-axis values increase towards the user.

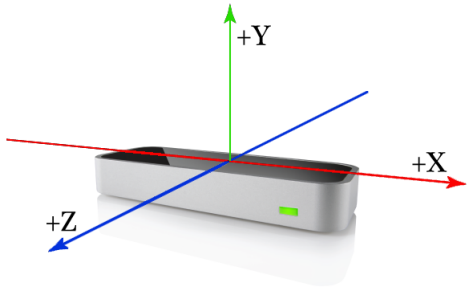


Figure 3: Leap Motion right-handed coordinate system [3]

The Leap Motion controller organizes all the information it finds about the hands, fingers, and tools in its field of view in a set, or a frame, of data. Each Frame object contains lists of tracked entities, like hands, fingers, and tools, as well as recognized gestures and factors describing the overall motion in the scene. The Frame object is the root of the Leap Motion data model [3]. Its in these frames that we draw the data necessary to change the player's hand model position.

3.5 Sound

The Leshy SFMaker in-browser sound creation tool [9] was used for many game-like, synthy, and/or interface-focused sound effects. For music, Finale Printmusic 2011 was used in composition, while Cakewalk's Music Creator 6 Touch served the purpose of applying

virtual instrument patches and various effects. Audacity and the Moo0 Audio Converter v1.32 provided conversion utilities to the necessary .wav encodings.

4 Related Work and Research

To help us design a game to meet our goals, we looked at some of the research done in virtual reality, as well as some games with Oculus Rift compatibility. We noted the important rules attributed to successful VR simulations and the science behind VR. There has been a lot of research done in the field of virtual reality including everything from purely visual immersion to full body simulations. For the purposes of our work, we primarily focused on the work that was directly related to graphical output using head mounted devices and input using the player's hand and head motions.

Each of the different games we looked at taught us something different about important features that went into making an enjoyable game using the Oculus Rift. In some games, it was the mechanics; others had high quality graphics and impressive environments. It is important to note that a majority of the games sampled were not designed specifically for the Oculus Rift. Rather, they were ported after it became available for development purposes. Only a few games were Rift specific. This is important to note because it is possible to see the difference in mechanics between Rift ported games and Rift specific games.

4.1 Virtual Reality

Virtual reality is an immersive multimedia technique that applies attributes of perception of the physical world to computer-simulated environments. Sensorimotor actions are important to complete immersion within the simulation, and it is here that VR can be limited. If the user is unable to physically move around and interact with virtual content as if it was in the real world, immersion may be broken[10]. At the same time, if the user walks around, the controls must go with them. There is also a problem of inadvertently colliding with real world obstacles, causing disruption in the gameplay. It may be a safety hazard as well.

That being said, virtual reality is more than just interacting with 3D worlds. Giving the user sensory perceptions within the world enhances immersion and leads to a greater, more unique experience. Human senses, in general, tend to be dominated by what is being seen, rather than what is there in reality[11]. By dominating the user's field of vision with a convincing representation of a physical world, we are able to fool them into believing they exist within a space that may be larger than the one they currently occupy [10].

The combination of a sense of immersion and interactivity is called telepresence [10].

This is the sense of being elsewhere, created through a game or simulation. Computer scientist and online publishing pioneer, Jonathan Steuer, defined telepresence as "the extent to which one feels present in the mediated environment, rather than in the immediate physical environment" [12]. He also proposed that there are two main components when dealing with immersion: *depth of information* and *breadth of information*. Depth of information is referring to the amount and quality of data that the user receives (graphical output, controler vibration, sounds, etc.) and breadth of information refers to "the number of sensory dimensions simultaneously presented" [12]. For the immersion to be effective, both criteria need to be met.

The virtual environment also needs to adjust in real-time as the user explores the environment. If there is 3D sound in the environment, the user needs to be convinced that the sound's orientation shifts naturally as he or she navigates the environment. Sensory stimulation must be consistent for a user to feel immersed in the virtual environment. Any lag time between the user's actions and the virtual environment's reflections, also known as *latency*, will cause a user to become aware of being in an artificial environment and will destroy the sense of immersion [12]. Specifically, when working with the Oculus Rift, any latency between the user and graphical output can cause disorientation and possibly motion sickness.

To meet our project goal, we needed to achieve both a depth of information and a breadth of information. Through the game's large environmental geometry, the high resolution of the Oculus Rift, and high quality sound effects, we achieved satisfactory depth of information. Allowing the player to use both their head for looking and their hand for controlling character movement, as well as providing visual and audio feedback through player actions, let us fulfill the breadth of information requirement. Because of the restriction of mobility the player is limited to with our hardware, we needed to utilize the maximum amount of space that was available to the player without having the player walking around. Basically, this limited the user to sitting in a chair or standing. In conjunction with this, we needed a convincing illusion of the game space's existence. The game allows the player to have a sense of movement without physically walking from their spot by having the game character move automatically. This also gives a reason for the player to remain stationary. Even though they are not walking, the player is making use of other movements involving their hands and head to give visual feedback. To give a better sensation of being within the game world, the player is able to "interact" with it by moving their hand via the Leap Motion Controller. This moves the hand model seen on the screen. When the onscreen

hand mimics the motion of the player's hand motion, telepresence is created.

4.2 Rift Reality

When developing a virtual simulation for any head mounted device, there are many different practices to keep in mind to make an enjoyable simulation. Oculus VR, the company behind the Oculus Rift, has released documentation detailing recommended practices when developing software for the Rift. We used this information to improve the design of our game to enhance immersion and prevent discomfort. In particular, our game:

- Doesn't force the player to look in a direction they aren't moving in
- Doesn't zoom or move the camera for any reason other than the player character's movement
- Was tested by users to see how it affects them
- Contains no impacts that move the camera without the user's control
- Doesn't use any HUD elements
- Doesn't use any input devices that the user will have a hard time finding without seeing
- Doesn't require the user to strafe, back-step, or spin

Our game does feature a lot of acceleration, which can cause some motion sickness. Most of the other best practices in the document are details that we kept in mind during implementation [13].

4.3 Oculus Rift Games

To learn more about what makes an Oculus Rift game successful, we played several games that make use of the Rift. We designed our game around the things that we learned did and didn't work in these games. The games included:

- *AaaaaAAaaaAAAaaAAAAaAAAAA!!! for the Awesome*
- *HAWKEN*
- *Minecrift* (a *Minecraft* mod for Rift support)

- *Proton Pulse*
- Several small Rift demos

4.3.1 *AaaaaAAaaaAAAaaAAAAAaAAAAA!!! for the Awesome*

Figure 4: One of the levels in *AaaaaAAaaaAAAaaAAAAAaAAAAA!!! for the Awesome*.

AaaaaAAaaaAAAaaAAAAAaAAAAA!!! for the Awesome is a first-person arcade style game where the player is constantly falling downward due to gravity. Players must navigate around the buildings and land safely on the last platform at the bottom by deploying their parachutes. Figure 4 shows one of the many levels featured in the game.

AaaaaAAaaaAAAaaAAAAAaAAAAA!!! for the Awesome works very well on the Oculus Rift. The game plays well to the Rift's strengths. Depth perception made the space we were falling into feel much more real. Seeing large buildings rush past us on the side was impressive as well. Our inability to see the real world made the feeling of falling much more convincing. The game's only weakness on the Rift was small menu text that was hard to read. This would prove to be a recurring problem in many of the games we played. The game has some jokes, and we noticed that smiling while wearing the Rift can push it out of place [14].

4.3.2 *HAWKEN*

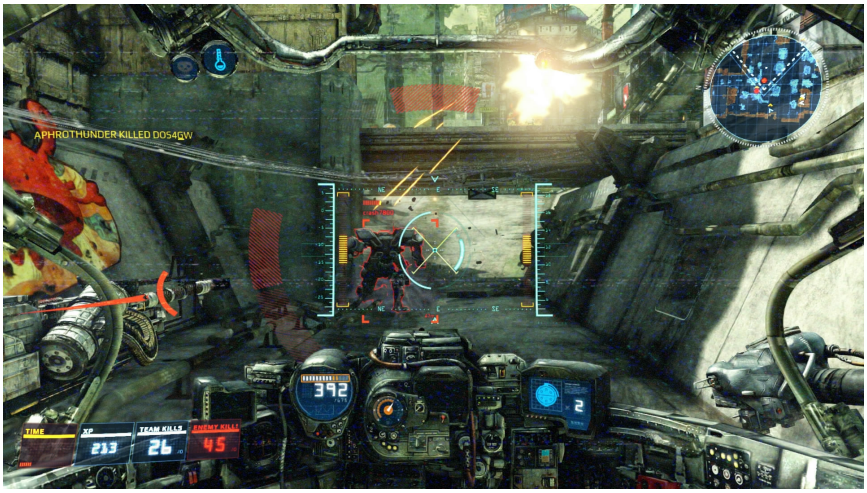


Figure 5: The player's perspective in the game HAWKEN

HAWKEN is a multiplayer mech combat game where the players must destroy enemy mechs in order to win. HAWKEN's use of the Rift was less impressive. Rift support for the game was still in development, so some features didn't work correctly. The main offender was the UI. On a standard screen, the game displayed important UI elements in the corners. Figure 5 shows the player's perspective and UI elements. With the Rift's more rounded display, these elements were completely out of view. Worse, they moved with the player's head, so there was no way to see them. The Rift's low resolution screen made it hard to see distant and small objects, making a crowded battlefield difficult to parse. As with the previous game, menu text was hard to read. It wasn't all bad though; depth perception made slow-moving rockets easier and more fun to aim and dodge [15].

4.3.3 *Minecrift*

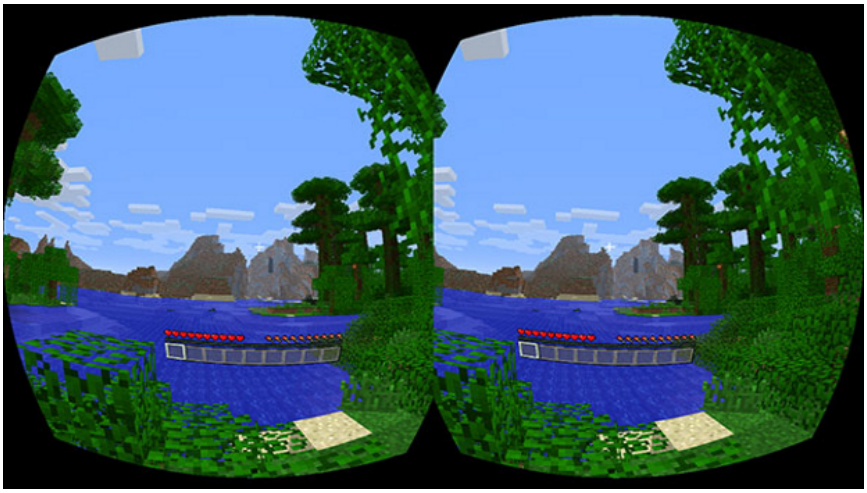


Figure 6: The Oculus version of the popular game, *Minecraft*

Minecrift is a mod for *Minecraft* that adds Oculus Rift support. In contrast to *HAWKEN*, *Minecrift*'s developers clearly put a lot of work into getting the UI right, or rather, allowing the player to make it right. Figure 6 shows how *Minecrift* looks through the Rift. It has plenty of customization options to make the UI comfortable and useful. Minecraft has a pretty simple UI anyway, which made this easier. While the menus had a lot of text, it was always large enough that it was easy to read. The game itself was a good fit for the Rift. It had great tracts of land that look really good with depth perception. The game's blocky aesthetic means that little is lost visually on the Rift's low resolution screen. We noticed that game objects close to the character felt closer to the player than they do on a standard screen, eliciting stronger emotional reactions [16].

4.3.4 *Proton Pulse*

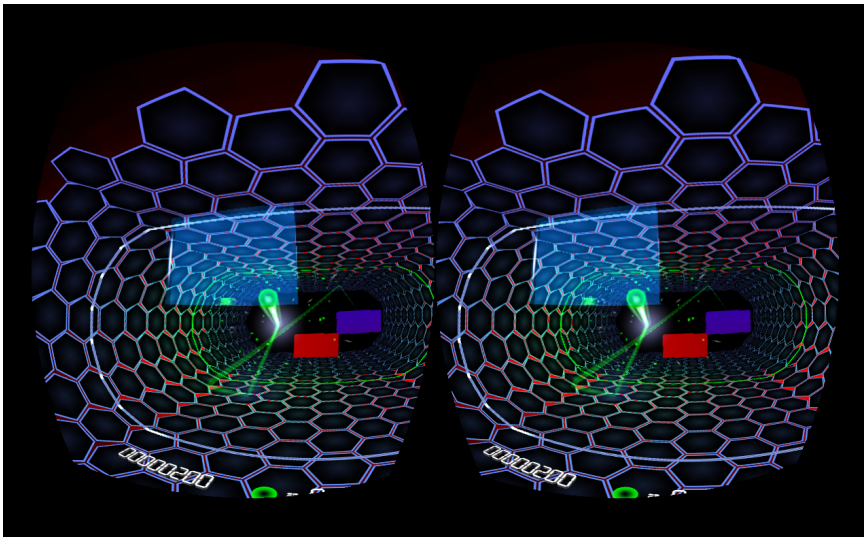


Figure 7: Player perspective of *Proton Pulse*

Proton Pulse is the only game we played that was specifically designed for VR. It is a 3D ball and paddle game where the player controlled the paddle by moving his/her head. The game didn't teach us much that we didn't learn from the other games, but it was the game with the best answer to the question "Why VR?". Almost everything in the game depended on VR to work. Without depth perception, it would have been impossible to tell where the ball and blocks were. Using head orientation to control the paddle was surprisingly intuitive as well. Some levels had background effects that rotated, which felt strange [17].

4.3.5 Oculus Demos

Other than these games, we played a few small demos made specifically for the Oculus. From these, we learned that large objects near the player's view look much more impressive than they do on a standard screen and that suddenly changing the player's view without input from them is disorienting. As obvious as it might seem, we also found that it's hard to find keyboard keys while wearing the Rift.

4.4 Summary

From our research, we learned that some game elements work well with the Rift and some should be avoided. Good elements include: large objects near the player, moving quickly through a space, slow moving projectiles, large open areas, characters and objects the player is meant to form an emotional connection with, and head-movement-based controls. Elements that should be avoided include: HUD elements, especially with text, jokes funny enough to make the player smile, rotating backgrounds, sudden view changes without input, and keyboard-based controls.

5 Design Process

When designing the game, our primary focus was to build it around the Oculus Rift and Leap Motion controller as unique output and input devices, respectively. We needed to create a game that would be able to fully utilize the functionality of both the Rift and Leap Motion, while at the same time, not being too difficult for the player to control.

5.1 Game Mechanics

Game mechanics are the core of any video game and are what the game should be designed around. Game mechanics refer to the set of rules and methods for interaction with the game state. Many styles and theories differ as to how ultimately important mechanics are to a game. In general, the study and process of game design, or ludology, are efforts to design mechanics that let people playing the game have an engaging, though not necessarily fun, experience. In virtual reality, specifically, the ultimate goal is not to have a fun experience, but a truly immersive and engaging one. Through an engaging experience, users can have many different levels of fun as a side effect. Something as simple as sitting at a VR desk, interacting with the virtual objects on the desk can prove a fun experience, provided it is engaging enough.

Our initial design was to have the game be from a first-person point of view, since this is typically how virtual reality games/simulations are done. The player is seeing through the eyes in the in-game character. Because of the limited mobility that the player has while playing, and for complete immersion, we needed to give the player a way to traverse the game's landscape without moving in the real world. To solve this problem, we have the in-game character moving forward automatically, with direction being controlled by the player.

The design of the game was primarily focused on creating a large virtual environment to give the player a sense of being in a much larger space when playing the game.

5.2 Levels

When it came to level design, our approach was to have the player learn how to play the game without explicitly telling them how everything worked. The goal in designing the levels was to have their difficulty increase linearly as the levels go on, with Level 1 being the easiest and Level 4 being the hardest. Each level is designed around a specific mechanic that the player is encouraged to master in order to complete the level as quickly as possible.

During the first two levels, the player is introduced to different mechanics and items. These levels act as a sort of an informal tutorial for the player, to allow the player to familiarize himself/herself with the mechanics and items. Afterwards, during the last two levels, the mechanics and items are combined to test the players skill at using them.

The levels are designed to be quick and completed within about 1 minute. Depending on how well the player performs, this time can be cut in half. Each level is intended to be short as not to tire out the player. It is possible to complete each level without the use of some of the items through alternate paths, but in order to complete the level as fast as possible, the player needs to explore other routes. The levels were designed in a way that the player could return to them after completion and after gaining more experience playing, and complete the levels in a shorter time using alternative paths to the goal.

While we wanted to make the world as open as possible, to give the player a sense of a large open environment, the levels themselves are fairly linear, with obstacles blocking the player's access to certain areas. To limit the amount of geometry and the respective resources to load and render it, the outer boundaries of the levels are blocked from player view by other environment geometry.

5.2.1 Level 1

The mechanics incorporated into this level are:

- Speed rings
- Static environmental obstacles the player must maneuver around
- Speed boost item
- Going uphill

Level 1 was designed to be a very easy, open level without many challenge obstacles. When the player first starts off, they are in a wide open area with the rings directly in front of them. The main goal in this level is to have the player gain an understanding of the turning mechanics, as well as introduce the ring mechanics and speed item mechanic. Near the end of the level is an upward slope that introduces the player to the mechanic of going diagonally and how their speed decreases as they continue to go uphill. The player is able to overcome this if they collect the speed boost item near the entrance of the upward slope. Figure 8 shows an overhead view of the entirety of level 1, with the start being near the bottom of the picture and the end goal near the top left region.

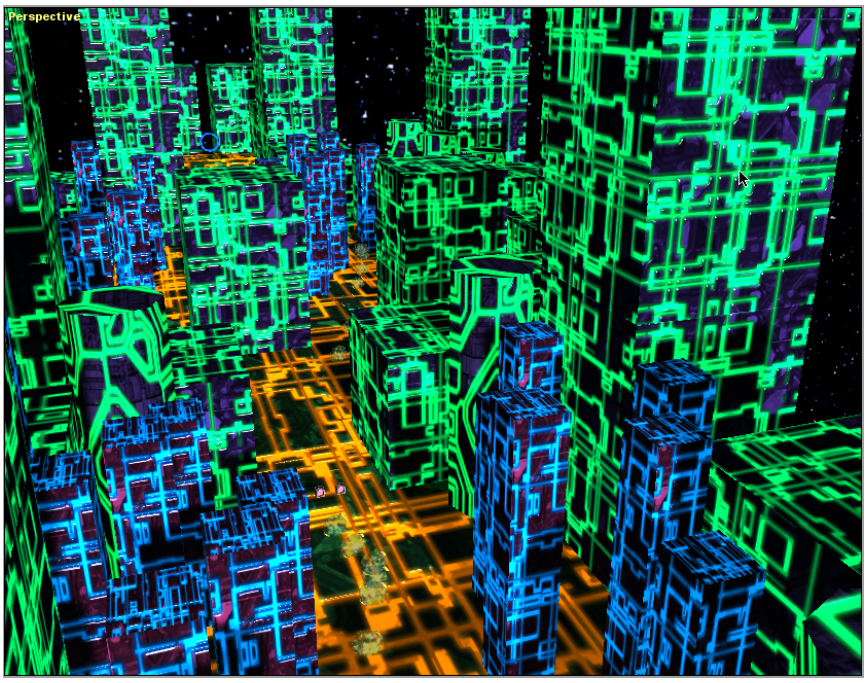


Figure 8: Overhead view of Level 1 from start to finish

5.2.2 Level 2

The mechanics incorporated into this level are:

- Speed rings
- Static environmental obstacles the player must maneuver around
- Dynamic environmental obstacles the player must maneuver around
- Ring boost item

This is the first level where the player encounters moving geometry. The goal here is to encourage the player to go along with the downhill curve of the path and increase their speed. This will make it much easier to get past the moving obstacles. The player is also introduced to a new item, the Ring Boost item. These items are positioned so that the player can use them to avoid the moving geometry. A complete view of the level can be seen in Figure 9 and Figure 10.

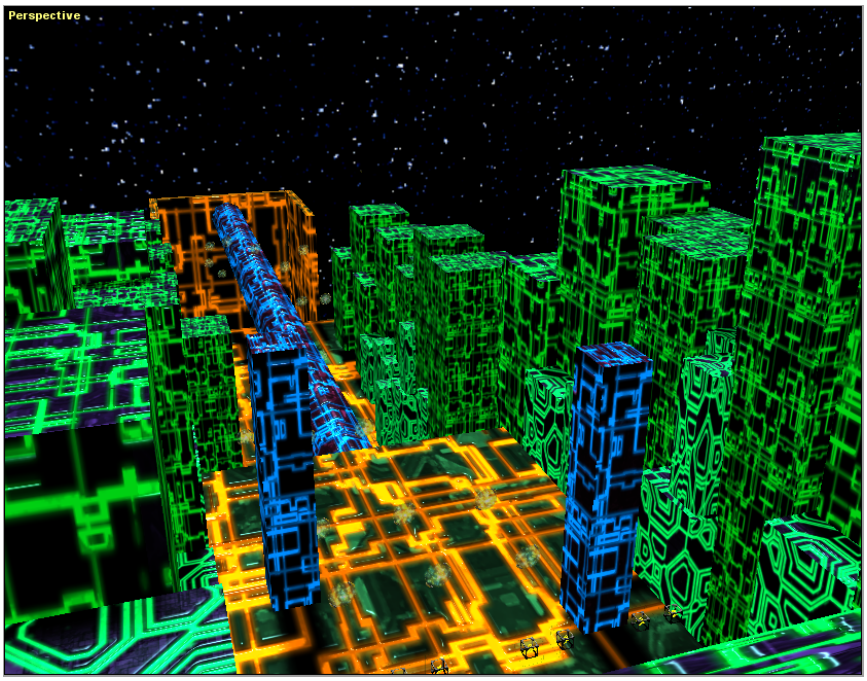


Figure 9: Overhead view of Level 2 from the starting position.

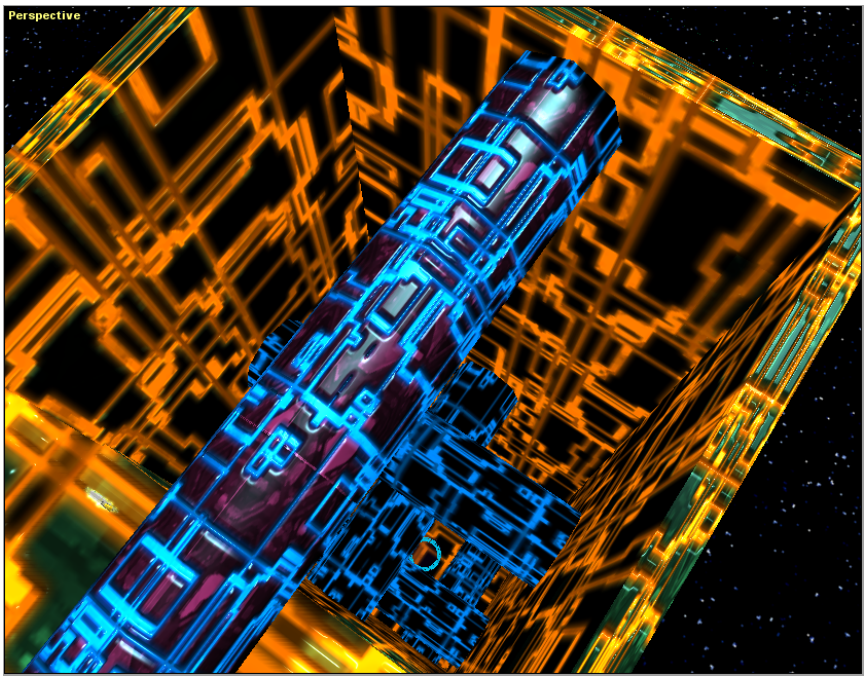


Figure 10: Overhead view of the downward passage in Level 2 leading to the end goal.

5.2.3 Level 3

The mechanics incorporated into this level are:

- Speed Rings
- Static environmental obstacles the player must maneuver around
- Dynamic environmental obstacles the player must maneuver around
- Speed boost item
- Ring boost item
- Downers

Level 3 is the first level to introduce"Downers". The Downers are obstacles that decrease the players velocity when hit. In this level, the Downers are placed in such a way that it is very difficult to avoid them. One must have good skill with maneuvering in order to avoid them completely. The level was designed this way to introduce new players to the concept of downers without greatly impeding their progress. The overhead view of level 3 can be seen in Figure 11.

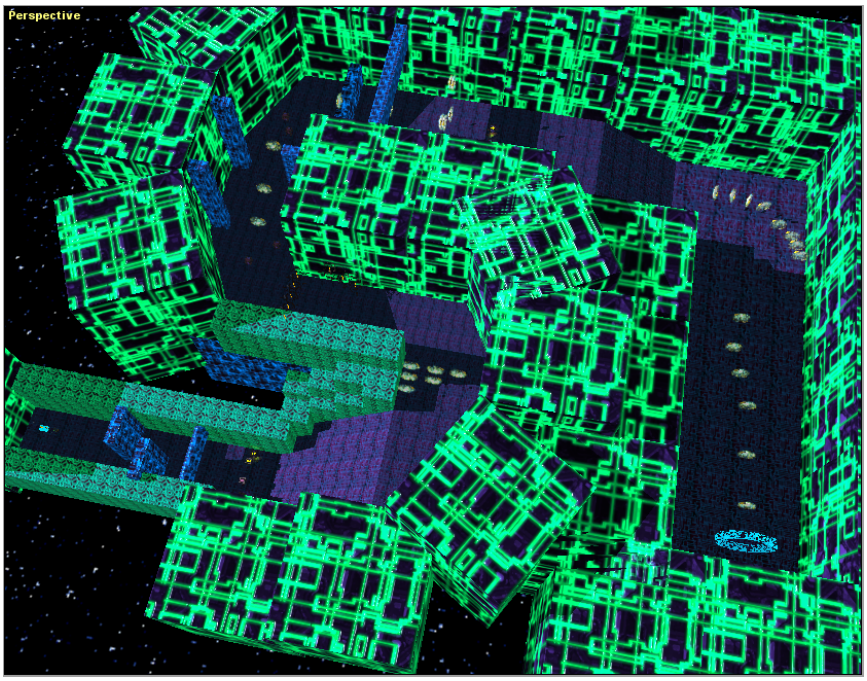


Figure 11: Overhead of Level 3 in its entirety

5.2.4 Level 4

The mechanics incorporated into this level are:

- Speed Rings
- Ring boost item
- Downers
- Static environment obstacles the player must maneuver around
- Speed Boost Item
- Going downhill and uphill

Level 4's main section is a huge tower that the player must spiral up around. There is a path of ring boost powerups and rings arranged so that the player can get to the top very quickly if they hit every ring. At the top of the tower, the player flies through a gate into a vast, open cityscape far below them, with the goal ring at the other end. The inner most part of level 4 can be seen in Figure 12 and the outer most part in Figure 13.

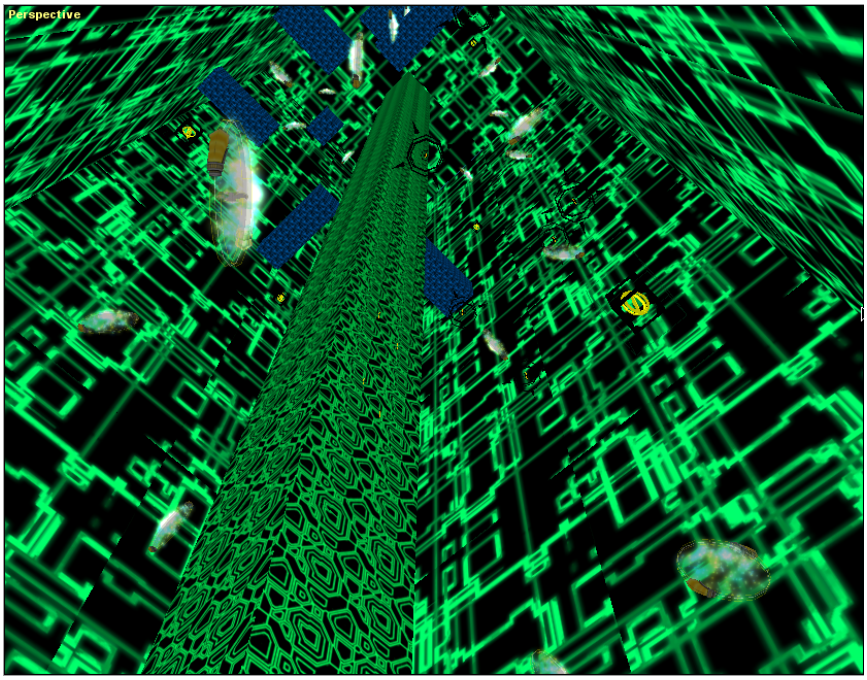


Figure 12: Upward view of the corridor in Level 4 leading to the outside.

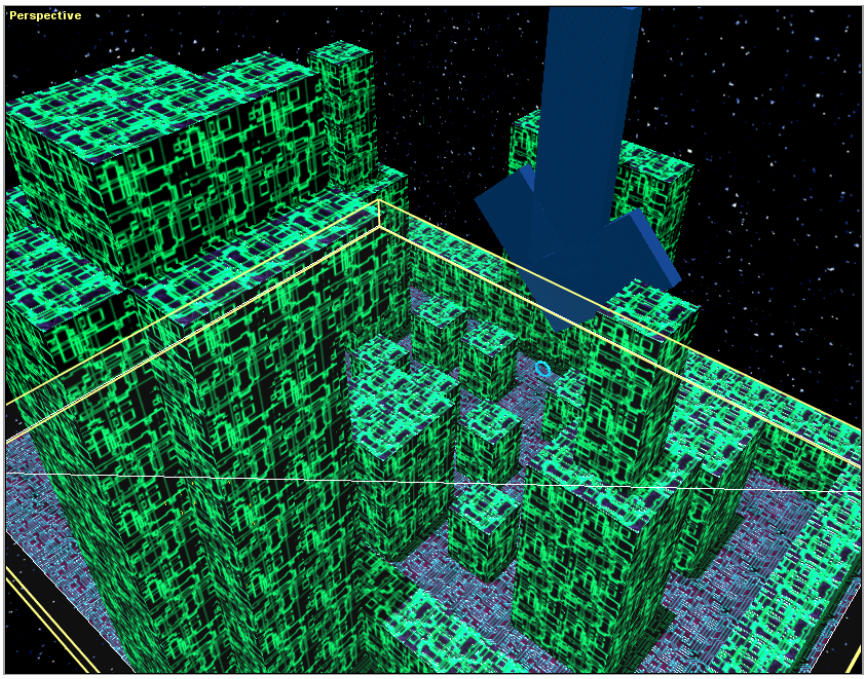


Figure 13: Overhead view of the outside of Level 4

6 Game Mechanics and Gameplay

Often, there can be some confusion when it comes to the difference between a game's mechanics and its gameplay. One could define gameplay as the combination and interaction of the game's many elements. This differs from game mechanics, which we previously defined as the rules designed for interaction with the game state (*See section 5.1 on page 19*).

6.1 Objective

The goal, or object, of a game slightly differs from the gameplay itself. The objective of the game is to reach the end of the level as quickly as possible.

6.2 Mechanics

The player's hand controls a light path that is continually constructed in front of them. A neutral position means the path is level and straight forward. Moving the hand up, down, left or right creates the appropriate incline, decline, or left/right curvature. Rolling the hand left or right causes the path to roll the same way.

Several game elements can raise or lower the player's movement speed. Moving upward slows the player down, while going downward speeds them up. The player will lose speed when going around a sharp turn if the path isn't rolled so that it banks. There are rings that give the player some speed if they passes through them. Conversely, some obstacles, called "downers", reduce the player's speed. Finally, consumable powerups can give the player a temporary speed boost.

The player completes a level when they flies through a goal ring at the end of the path. They fail the level if they crashes into any solid geometry or their own path. On completion or failure, the player is returned to the menu where they can choose to attempt the level again or play something else.

6.3 Rings

If the player (specifically the point where the camera is located) passes through a ring, their speed is increased. Rings have a diameter of 185cm. They are somewhat challenging to hit without any complications, and are often used in conjunction with other challenges.

6.4 Powerups

There are two kinds of powerups that the player can pick up. Once picked up, these can be used to give the player character temporary benefits. The player can only carry one powerup at a time. If they touche a new one while already holding a powerup, the new one replaces the current one. To use the powerup, the player makes a grab gesture.

6.4.1 Speed Boost

One type of powerup is the speed boost. Figure 14 shows the in game model for the powerup. When used, this sets the player's speed to twice their current speed (and at least a set moderately fast speed in case their current speed is particularly slow) for a few seconds, then returns it to what it was at the start of the speed boost, plus any additional speed accrued from passing through rings during that time. This can be used to climb hills or go through downers "for free".

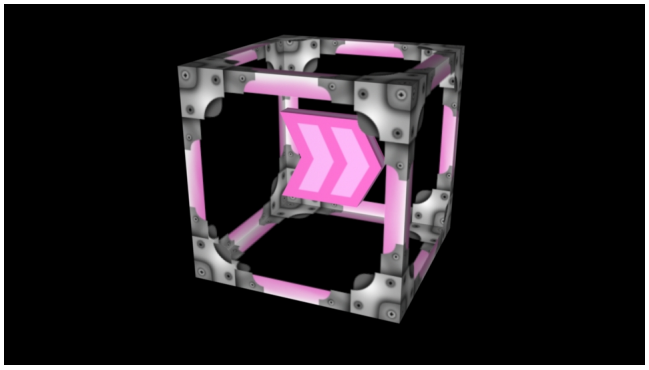


Figure 14: Speed Boost Powerup

6.4.2 Ring Boost

The other type of powerup is the ring boost. For 5 seconds after use, rings give the player a speed boost like the one the Speed Boost powerup gives, but with shorter duration.

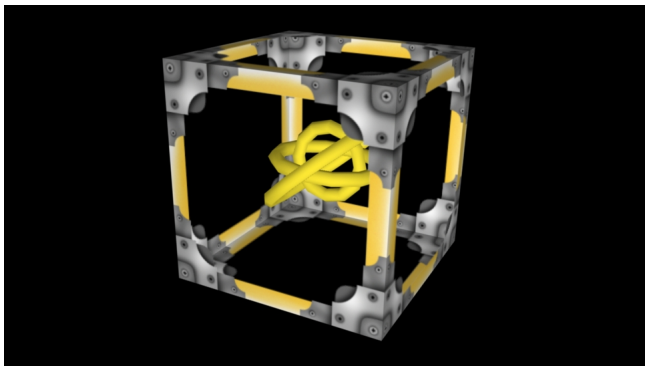


Figure 15: Ring Boost Powerup

6.5 Obstacles

There are two types of obstacles the player should avoid: Downers and geometry. Downers look like a floating gadget. If the player hits a downer, he loses some speed. Downers have a very distinct appearance and sound. Their sound is spatial, so the player can use it to detect the location of a downer even if they can't see it. Geometry obstacles include any other level geometry and the player's light path. If they hit any of these, they fail the level and are returned to the menu.

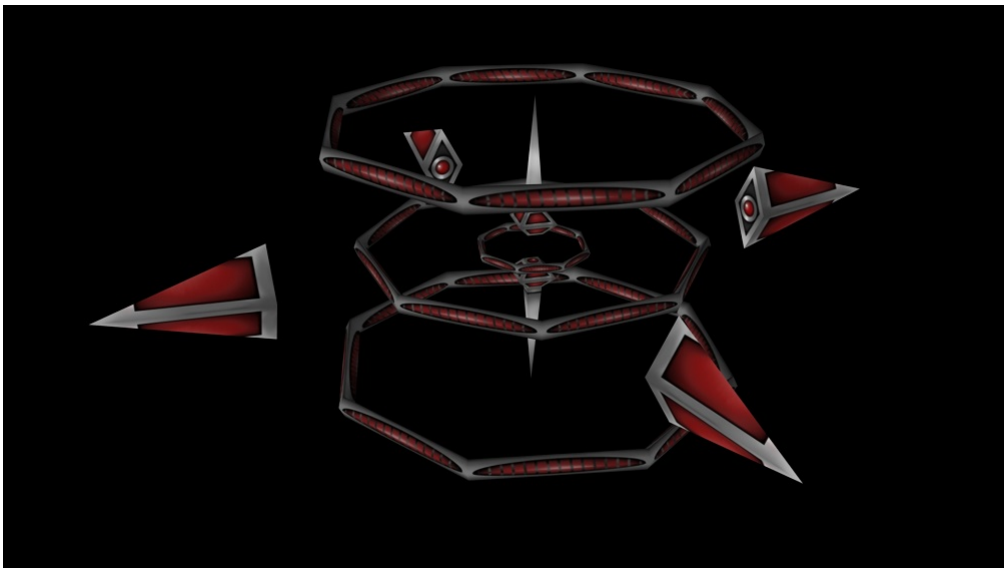


Figure 16: Downer

7 Programming

7.1 Code Structure

Most of the game's mechanics are defined in the player, hand, and light path controllers. For more information on controllers, see the Controllers section below. The three controllers depend heavily on and contain pointers to each other.

7.2 C4 Engine

The C4 Engine consists of a collection of managers and each manager serves a different purpose. Some of the high-level managers depend on low level managers, which are the only parts of the engine that interact with the OS. This hierarchy is designed to make the high-level functionality of the engine completely platform-independent[5]. Games and applications are built as DLLs that the engine can link to and run. This structure provides a clean separation between game code and engine code and allows a game to be built without compiling any engine code. C4 uses mutual exclusion, or a mutex, to ensure that only one instance of the engine is running at one time. This helps prevent any race conditions the engine may encounter during runtime. However, any number of tool modules can be used at once within the one C4 engine application[5]. With this, it's possible to change a specific tool or application quickly and efficiently. A more detailed illustration of the overall architecture, including all the low-level and high-level managers, can be seen in figure 17.

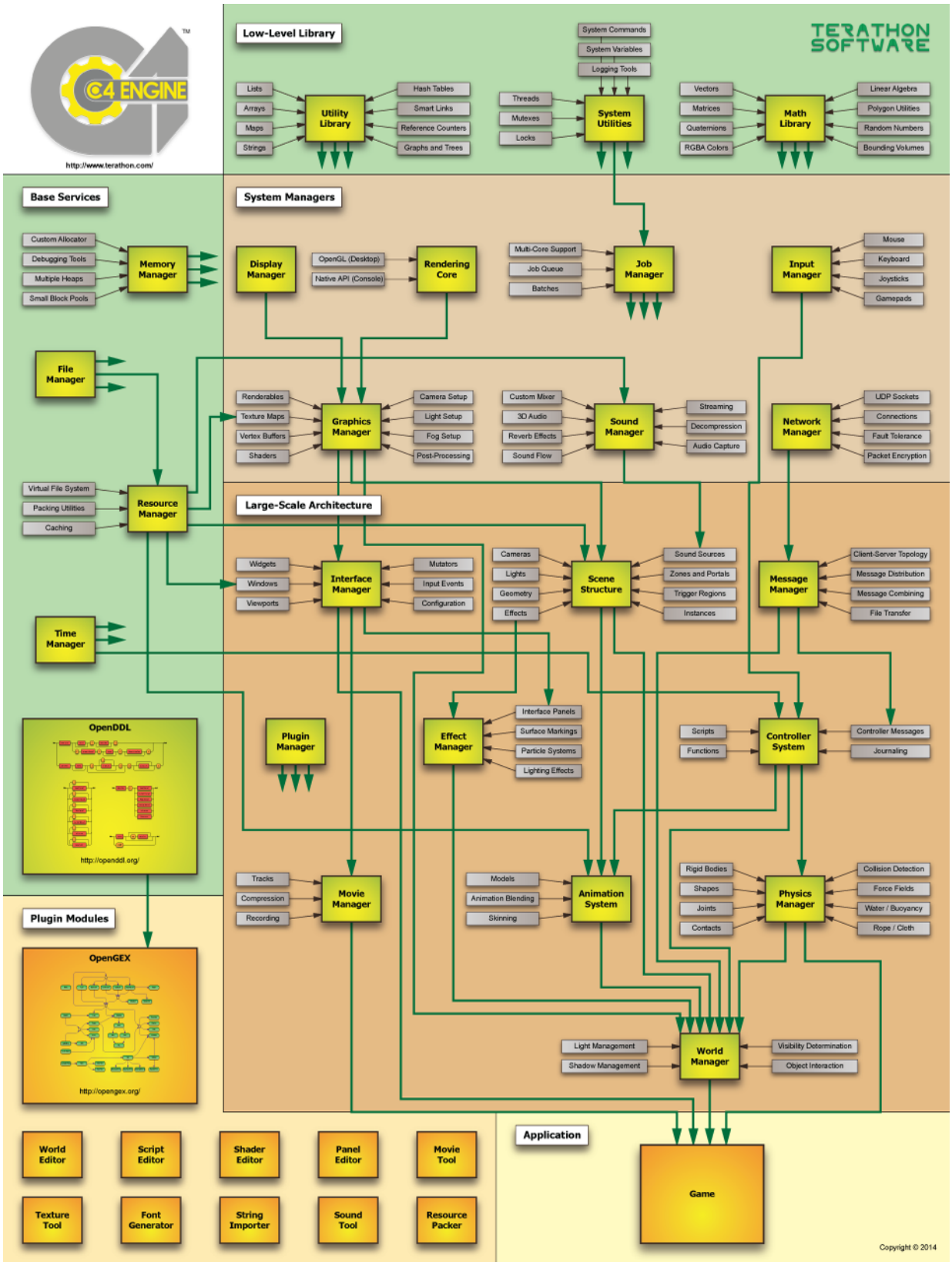


Figure 17: C4 Engine Architecture[4]

7.2.1 Nodes

Single scenes rendered in the C4 Engine are defined as *worlds* and are represented in the World class. All items in the world are organized into a scene graph having a tree structure. Each item is represented by a single *node* defined in the Node class. The Node class serves as the base class for a large hierarchy of different classes used to represent a variety of different types of objects in the scene graph. Nodes can have any number of subnodes as children of that node and all nodes, except the root node, have one parent supernode. Some examples of various nodes include:

- Lights
- Cameras
- Geometries
- Sources
- Zones
- Portals

7.2.2 Objects and Controllers

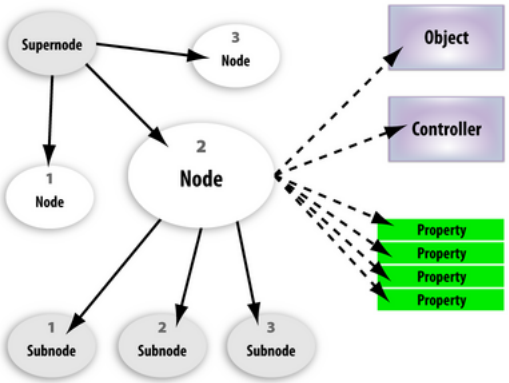


Figure 18: Relationship between nodes, objects, controllers, and properties[5]

Figure 18 illustrates the relationship between nodes and their objects. For most node types, there are corresponding types of objects. A node is able to reference only one object corresponding to its primary type, but an object can be referenced by an unlimited number

of nodes. This is how C4 does object instancing. Any instance-specific information is stored inside a Node subclass and all information is shared among all instances stored in an Object subclass [5].

All the nodes in a scene graph can have a *controller* attached to them. A controller, represented by the Controller class, is used to manage anything about its target node that changes over time. A controller could be used to move a geometry node in some way, or even modify the brightness of a light source. There are several types of controllers built into the core engine and applications can define their own type of controller by subclassing from the Controller class.

7.2.3 Controllers

In C4, nearly every object in the game that does something has a controller attached to it. A controller object is a subclass of the C4 Controller class and does exactly what it sounds like: controls the object it is attached to. Controllers have a function called Move that is called every frame. This is where the bulk of the game's mechanics are coded. Figure 19 shows the relationship and dependencies between each of the game's main controllers.

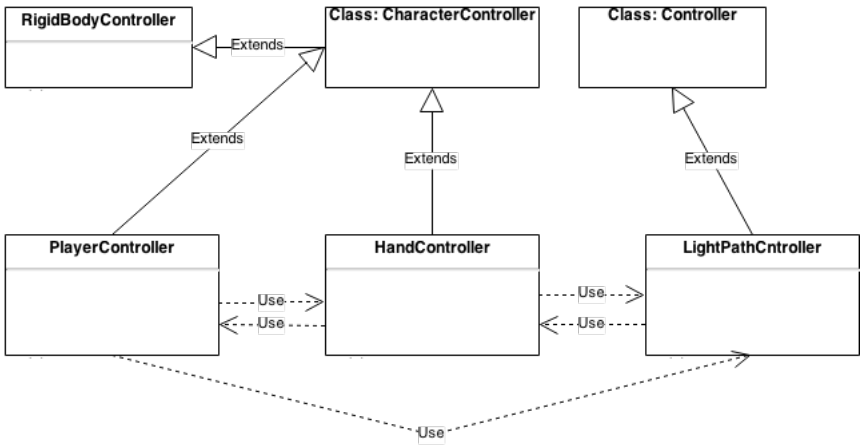


Figure 19: Relationship between controllers

7.2.4 Player Controller

The PlayerController class is the controller of the main player. PlayerController extends the C4 CharacterController class, which in turn implements the C4 RigidBodyClass. The CharacterController class class manages a basic character model and implements a small amount of functionality that is commonly needed for basic characters. The RigidBody-

Controller class manages a rigid body in a physics simulation. Extending the CharacterController class allows the player model to implement animations and the RigidBodyClass gives the player model physics and rigidbody collision detection. This controller uses the LightPathController class to receive information about how the player model changes position and orientation along the dynamically created path. From the HandController class, it receives information for when it is time to "roll" and when a item is used.

The player controller's main purpose is to control the movement of the player character. Each frame, the player's position and orientation must be calculated. The player controller uses a spline (see Spline Interpolation in section 7.3) made of points on the light path to find the player's position along the path. Orientation comes in two parts: horizontal orientation and roll. The player is rotated horizontally to face the front of the light path, whose location is easily retrieved from the light path controller. The player character is then "rolled" to match the roll of the section of path he is currently standing on. This is more difficult, as there is no way to look "back" along the path to learn how the section of path the player is standing on was rolled. Instead, the hand controller (where roll information comes from originally) constantly reports its roll to the player controller, which stores it until the player reaches the point on the path that was generated when that piece of roll data was reported. The player controller is also responsible for handling interactions and collisions with other objects, such as rings, powerups, downers, and geometry. It contains all the code for the powerup mechanics.

7.2.5 Hand Controller

The HandController class extends the CharacterController class, giving it animation, rigidbody collision detection, and physics. Since the hand object in the game has no weight and is moved based on the player's hand position, physics was not necessarily needed. We did, however, need to enable rigidbody collision detection to let the hand interact with the powerup and environment geometry.

The hand controller's main job is to retrieve data from the Leap Motion controller, (see Leap Motion section below) change the in-game hand's position and orientation to match that of the player's physical hand, and report this data to the player and light path controllers. The hand moves along with the player character; if the player doesn't move their physical hand, the in-game hand will not move relative to the player character. The hand controller also checks for a grab gesture, indicating that the player is using a powerup. It handles some interactions and collisions with other objects, usually referring them to the

player controller.

The hand controller, similar to the player controller, also handles the animations for the hand model. There is a default, reoccurring animation that runs each level. In the animation file the hand controller reads from, there are multiple "scenes" in a file. This means that in a single file, animation frames are mapped to the model that are depicting different actions.

7.2.6 Light Path Controller

The LightPathController class is the class that handles the behavior of the path the player model moves along. Unlike the HandController and PlayerController classes, this class extends the C4 Controller class. The Controller class is the general mechanism through which dynamic nodes are handled in a world. Many controllers extend this class because it is so general. The LightPathController class uses data from both the HandController class and the PlayerController class. As the player's hand is being monitored by the HandController class, it is sending information to the light path controller as to how it should behave. When the light path makes any change, that information is sent to the PlayerController class to let the player model know how to properly behave.

The light path in the game is made of a series of box geometries. Only the foremost box has a light path controller attached to it. The rest of the path is "dead" geometry. The light path controller's job is to extend its box forward into space until it detects that the player is trying to turn or roll the path an amount greater than a certain (small) threshold. It then creates a new box facing the new direction with a new light path controller attached to it, passes all of its relevant data to the new controller, and destroys itself.

7.3 Spline Interpolation

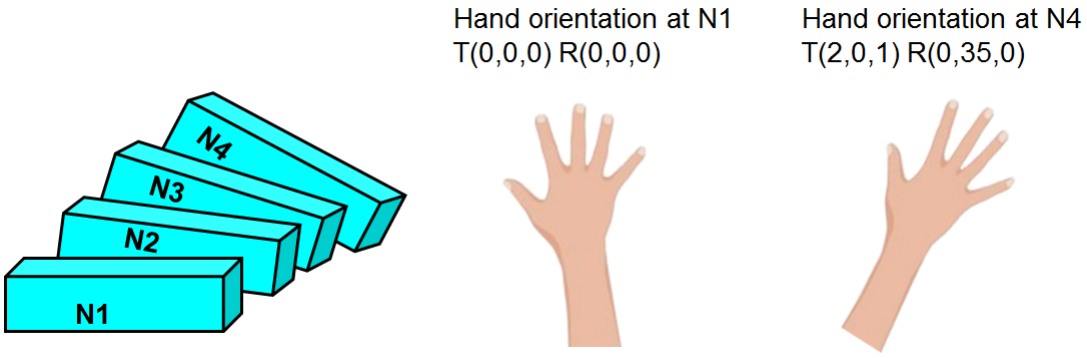


Figure 20: Simple diagram of how the light path behaves based on the player's hand position and orientation

A lot of code, time, and effort went into making the player's movement feel right. The light path that the player moves along is essentially a series of points. Figure 20 shows a simple example of how the spline behaves. In an early version of the game, the player moved in a straight line from one point to the next. When the player reached a point, he would turn instantly to face the next one, which is very jarring and not smooth at all. We used spline interpolation to fix this problem. Spline interpolation is a mathematical technique that finds a smooth curve that passes through all points in a series of points. In between each point (i.e. between points N1 and N2, N2 and N3, etc.), the path between the points are interpolated and more block geometry is added between the two points. This will smooth out the path instead of creating a stair-like path.

This was a natural fit for our problem because, as mentioned before, the light path is a series of points. This would have been a relatively easy solution, except for the fact that new points are frequently added to the light path. In some conditions, adding a new point at the front of the light path would change the spline at the player's position, causing him to "jump" a short distance in some direction. This was even more jarring than the original problem. To fix this, points are added to the front of the spline in short, regular intervals to create a "buffer" of points between the front of the path and the player. The player model then interpolates it's position and orientation between each of these points to give a smooth transition. Points a certain distance behind the player are deleted to save memory.

7.4 Oculus Rift

We developed our game for the Oculus Rift Development Kit 1. Development Kit 2 was released during development, but we stayed with DK1 because Rift support in C4 was still early in development and only supported DK1. From a coding perspective, working with the Rift was surprisingly easy. All we had to do was get the player's head orientation (which was very easy in C4) and orient the game camera the same way. The only problem we had with the Rift was that C4 was only compatible with an older version of the Rift SDK, which we had to figure out.

7.5 Leap Motion

Working with the Leap Motion controller was also very easy. C4 didn't have native support for it, but it didn't need to. The Leap Motion API includes libraries built in C++ and it integrated into our project very well. Any C++ program is able to get a "frame" of data from the Leap with one line of code. A frame contains hand and finger position information, which was everything we needed for our game. We used the Skeletal Tracking API Beta, which provided very accurate and reliable tracking data.

8 Art and Theme

8.1 Concept Art and Artistic References

We used concept art to give us a better idea of how the game may look or work. We started with concept art of how gameplay would work. Figure 21 shows an early concept of how gameplay would look from a third-person perspective. Some of the initial mechanics thought about in the early stages of gameplay design involved the player aiming and firing projectiles at enemies or targets. The figure shows what the gameplay may have looked like. This concept included robotic enemies that generate a noticeable light for easy visibility, and other important items that significantly contrasted with the environment to be easily recognizable on the Rift. The robotic enemy idea was scrapped when we decided to pursue a first-person perspective using the Oculus Rift. A third person perspective would not have worked out well and it would have been too complicated to try to aim a cursor with one hand while steering with the other.

Figure 21: Simple concept thumbnail

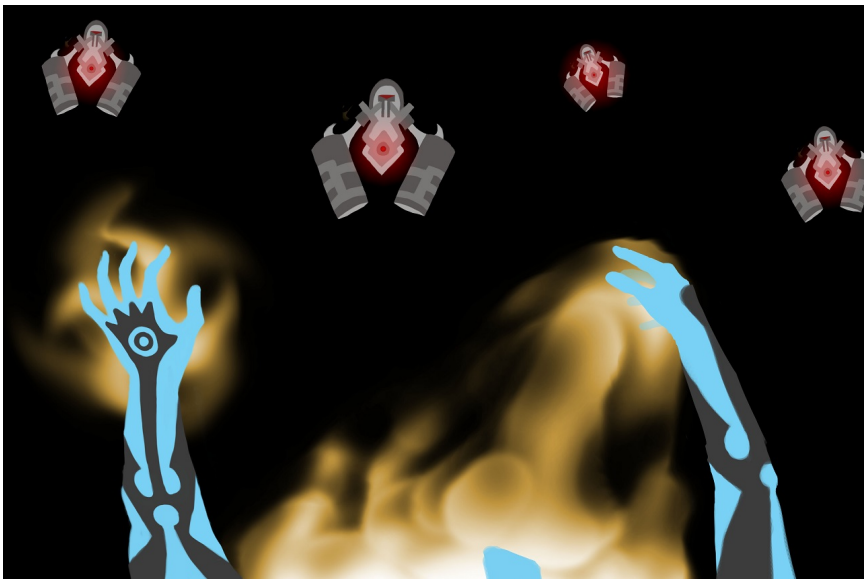


Figure 22: First-person gameplay mockup

Figure 22 shows a similar concept to the previous figure in keeping with the shooting mechanic. The player would use their right hand to steer the light path while using their left hand to aim at enemies. This is a simple mockup idea of gameplay from a first-person perspective. Again, this idea was scrapped after we ran into limitations with the complexity and hardware. The Leap Motion is only able to detect hand motion up to about 2 ft. and quick, rapid hand movements are not detected well. In the end we went with just one hand for detection.

Concept art was also drawn up depicting our idea for the how the environment would look. Figure 23 depicts an early version of the visual style of the game's environment.

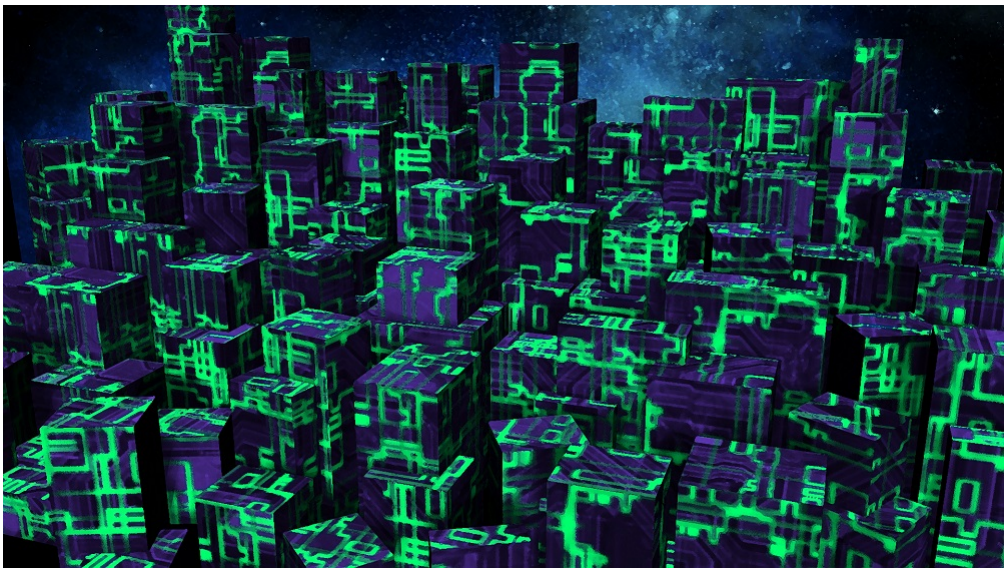


Figure 23: City environment

Overall, artistic inspirations for the character, objects, and environment come from:

- The Protoss race from the *StarCraft* game series for color and stylistic design (figure 24)

Figure 24: A Protoss Zealot from *StarCraft*

- *Megaman Battle Network* (BN), *Megaman X*, and the original *Megaman* series of games for stylistic design (figure 25)

Figure 25: Megaman and Bass from the *Megaman* series

- *Metroid* for stylistic design (Figure 26)

Figure 26: Samus from *Metroid*

- *Digimon* for stylistic design
- *World of Warcraft* for stylistic design and basic animation
- *Mario Kart* series for item design
- *Tron* (2010) for environment design
- *The Matrix* movie series for environment design

8.2 Character Design

The player character's concept went through two passes in design before being finalized. While making the concepts, we had to be thinking towards the future, with questions including:

- How can this be modeled for low-poly optimization?
- How much significance in the texture files should each bit of geometry receive?
- Where will joints be placed for this to animate well?

8.2.1 Character Concept 1

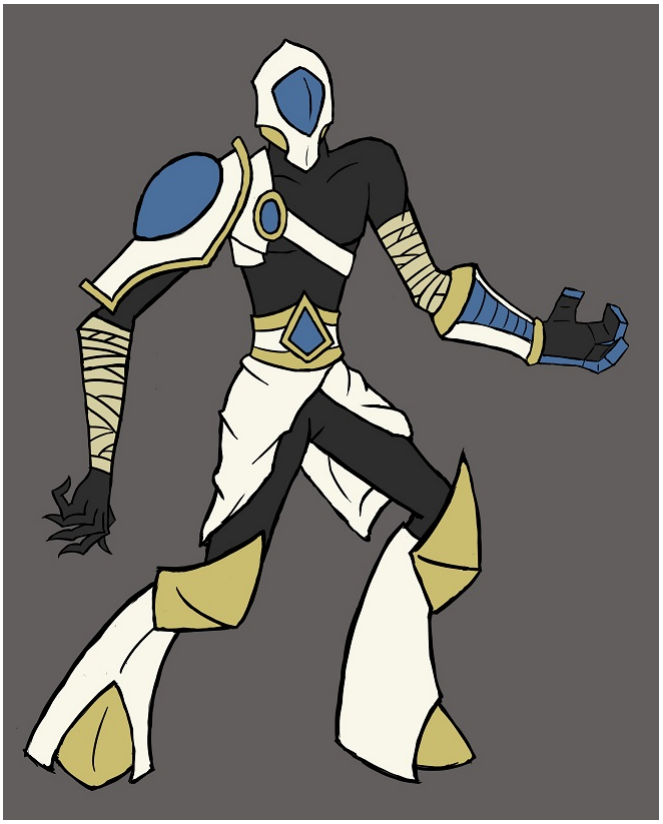


Figure 27: Character Concept 1st Pass

During construction of the first design, as seen in figure 27, the character had loosely fitting leggings, akin to hakama tucked into leg wraps, but these lose their bagginess below the knees instead of above them. This design was also going to have knee guards. However,

some critique noted that the character looked like a ninja in part because of the leggings. After researching more, this look too greatly resembled a ninja which is not what we were going for, so we removed the baggy leggings and knee guards. The design for the legs was instead changed to large greaves covering the legs from the knees down. The new appearance resembles the simple leg designs of characters from the *Megaman* series, especially Bass from *Megaman & Bass*, with the guards added at the top by the knees.

A different concept for the legs was they would be long needles or stakes, similar to the Steamlords in *Metroid Prime 3: Corruption*. Beginning either at the knees or the lower joint of double-jointed legs the character could be given, the needles would act like skates for the character to glide along the path.

Early in the design phase, we believed our game would have a first-person shooter feature, so the designs had the player character's two hands be noticeably different from each other: the slim, always path-creating right arm and the bulky, projectile firing left gauntlet.

8.2.2 Character Concept 2

Figure 28: Character Concept 2nd Pass

After we finalized some of the more important design and aesthetic choices in our game, the player character had to go through another design pass. At this point, the character concept received a slimmer, simplified, and somewhat androgynous look which many characters in the *Megaman Battle Network* series use. As with *Megaman BN* characters, cloth features were also removed in the new design, being that cloth is often difficult to animate.

The overall design in figure 28 was also made to be more symmetrical to make better use of low poly limitations. With an almost completely symmetrical design, UV mapping for the character's texture could allow for greater definition in certain areas, especially the blue gems in the belt, shoulders, gauntlets, helmet visor, and pauldrons.

The most obvious change with the new design is that the character's arms are no longer attached, but float alongside the player as a pair of gauntlets. This was done to allow for

easier animation for the player character in-game. If the original design had been kept, it would have been much more difficult to prepare and code the character's body to be animated in real time to follow the player's input with the Leap Motion. With this new design, the player's arm can be animated independently to follow the input.

In the old concept, the player's organic hand had a regular five fingers while the other hand only had a bulky and almost mechanical three. In the new concept we took the middle ground and gave each hand three fingers, but did not make them blocky and mechanical-looking. Additionally, fewer fingers meant fewer polygons in the hand model, making for easier weighting and animation. Since the new character overall is largely organic, the hands were chosen to appear organic, matching the character's main body's color. The character's finger movements are not reflected exactly by the Leap in-game, so we determined the character having fewer fingers would not affect immersion.

Additionally, the player character's status as freedom fighter was decided and the character's helm was changed slightly to look like a knight's pointed helm. The bandaged arm was removed not only because of the free-floating gauntlets, but also because its appearance more closely carried the resemblance of a bandit, not as much a freedom fighter or knight.

Other changes included giving the sides of the belt more geometry, changing the pauldron's design overall to be less plain, changing the pauldron's strap to connect to the added pauldron instead of looping back around the body, the removal of the brooch-like gem from the pauldron's strap on the chest and changing the cloth around the waist to tassets.

8.2.3 Character Model

Figure 29: Player character's full model

The player character's model is broken into eleven sections (pairs counted as two): Feet, Lower Body, Kneeguards, Belt, Upper Body, Pauldrons, and Gauntlets. Figure 29 shows the character model as it would appear in game. All models were made using Autodesk Maya. The polygon count for the character's main body is just under 1800 while each gauntlet is under 300.

8.3 Animation

Animations for the player character and items were also done in Autodesk Maya.

8.3.1 Character Animation

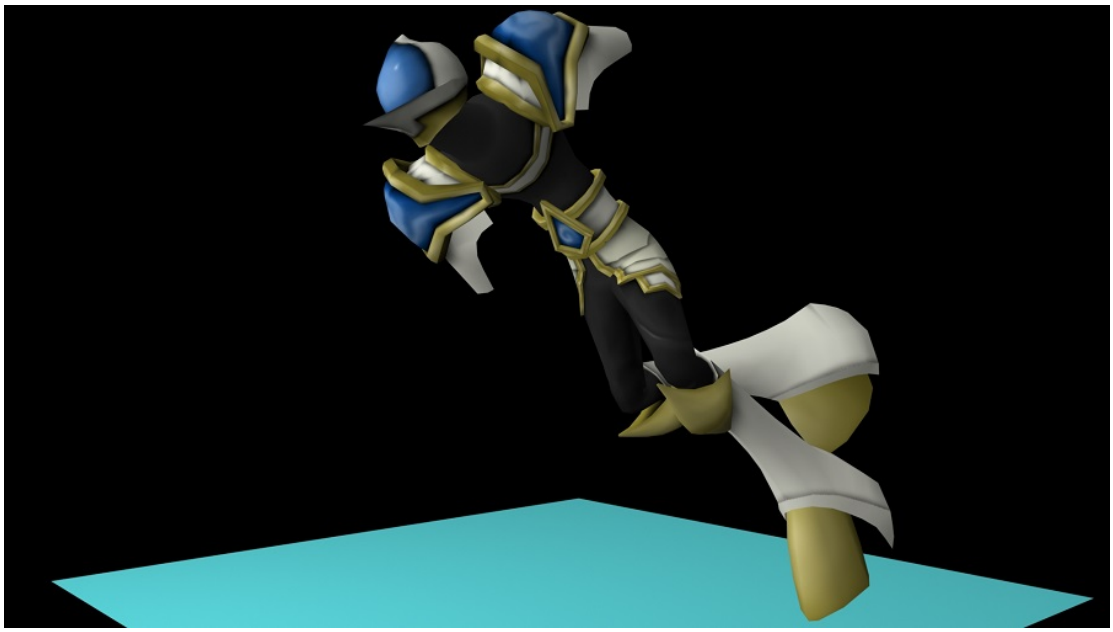


Figure 30: A keyframe of the player character's skating animation

Reference for the animation was taken from YouTube videos of a Sprint Speed Skating Championship and some athletes training for sprint speed skating. The model in figure 30 shows an animation keyframe similar to how a skating athlete would behave. The Sprint Speed Skating Championship athletes acted well as a reference for general speed, appearance, and minor varying movement as the athlete skated while the latter acted as a great reference for the limits of how far a skater's limbs and bodies bend in their skating form.

8.3.2 Gauntlet Animation

Animations for the hand are basic, consisting of an idle cast animation and a crush animation. The formers inspiration comes mostly from the various cast animations found in *World of Warcraft* while the latter was developed naturally while weighting the hand for animation. Animation for the gauntlet's inorganic majority is very subtle so it does not affect how the player sees its movement in game while they control it.

8.4 Items and Objects

Items and objects initially went through a short pass at design before being modeled. The individual item/object designs each went through smaller passes during development and received critique before being finalized. The item concepts shown in figure 31 show some early concepts for their design.

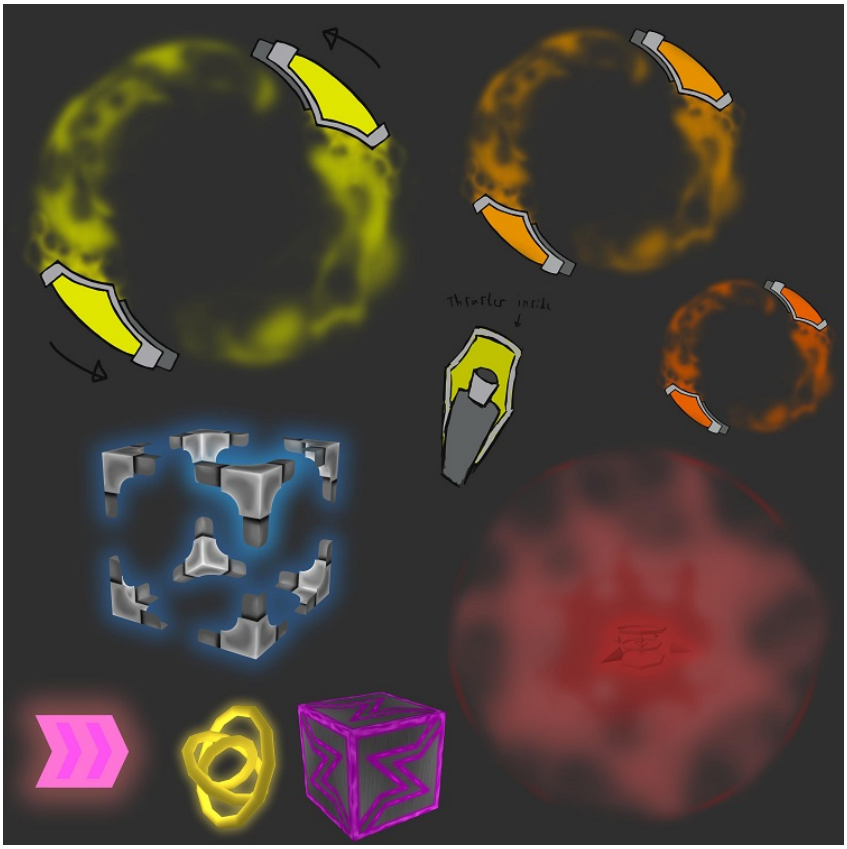


Figure 31: Early concept art for items and objects

8.4.1 Item Cubes

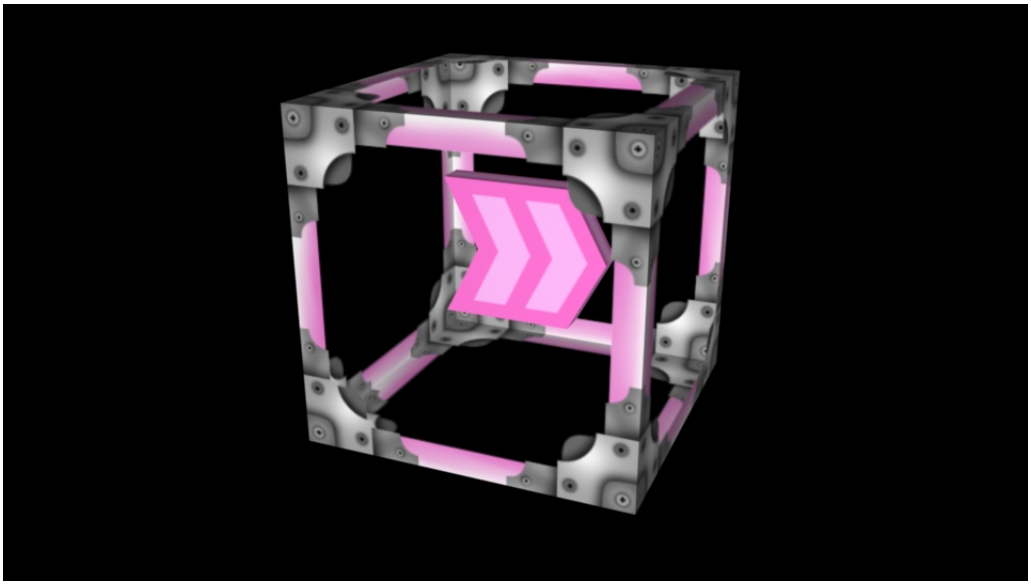


Figure 32: Speed Booster Cube Model

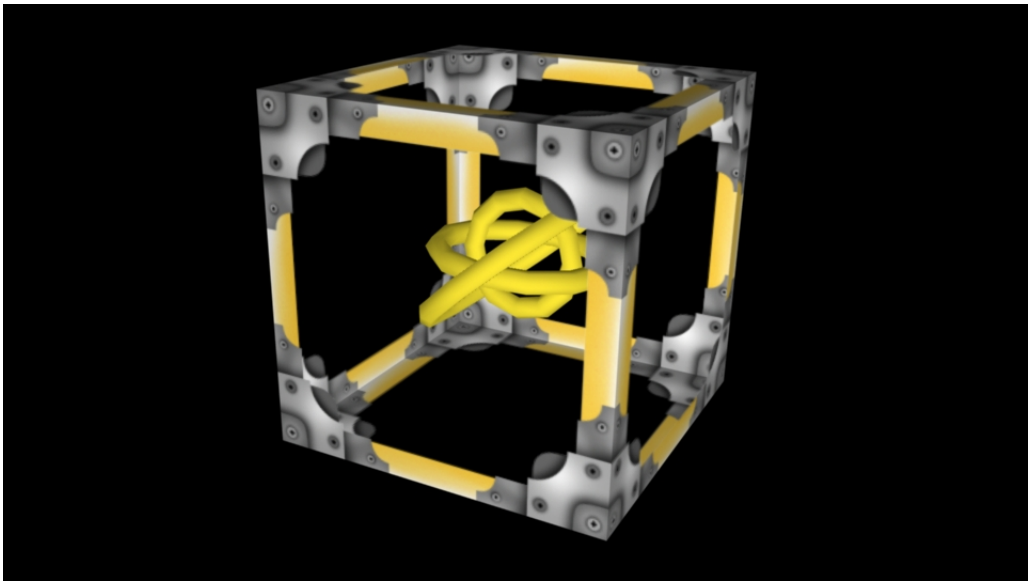


Figure 33: Ring Booster Cube Model

The Item Cubes' general concept and design takes inspiration from the *Mario Kart* series' item cubes and as such, look similar. *Mario Kart* item cubes, however, are always a varied blue or purple and have a "?" in their core to conceal what it holds, but our game only

has two useable items instead of the numerous possibilities found in *Mario Kart*, so our Item Cubes show what they contain with simple geometry while the cubes themselves have two distinct colors. The Speed Booster in figure 32 shows the item's core geometry to be a simple arrow resembling a fast forward icon on remote controls. The cube's distinct color is a pinkish purple. The Ring Booster in figure 33 shows the core's geometry as a set of three rings spinning in a gyroscopic fashion. The cube's distinct color is yellow. Because our game has plenty of objects the player can interact with by flying through, we went with the *Mario Kart* cube design to not deter the player from trying to run into our Item Cubes.

Originally, the texture for the actual cube edges looked wispy but the engine had trouble reading these detailed textures with a transparency map, so we had to simplify the texture to a more standard lightsaber look on the edges. The item cores of each cube were also going to be 2D planes to represent what they hold in a stylized way, but the fact that they were 2D was very obvious in our completely 3D environment so the current 3D cores had to be made.

8.4.2 Ring Generators

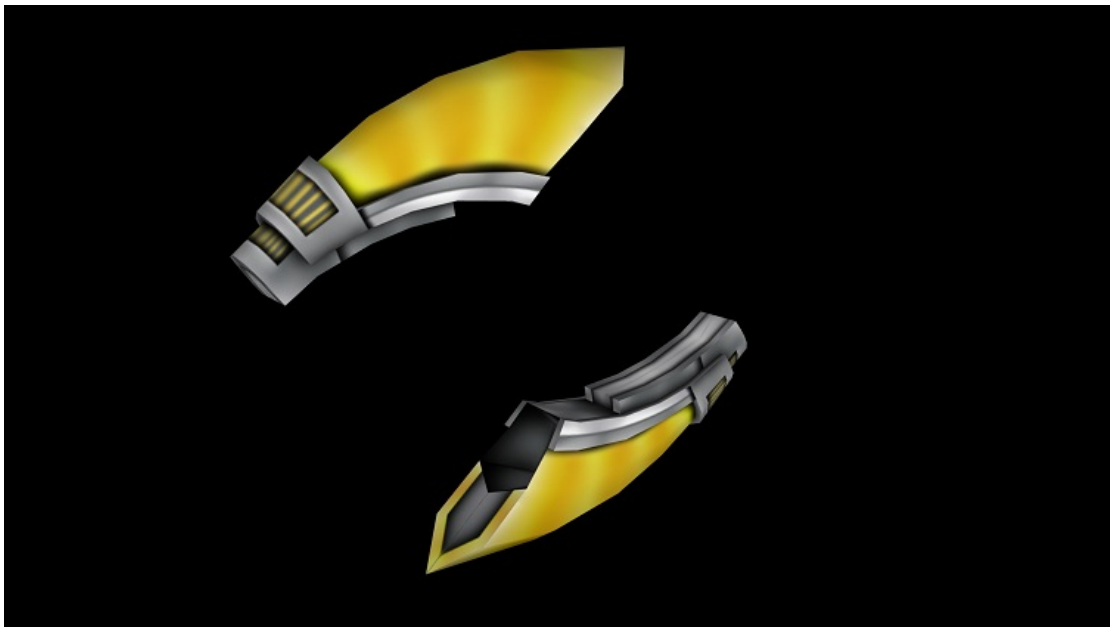


Figure 34: Ring Generators Model

The Ring Generators' appearance consists of a pair of yellow and grey thrusters spinning a circle.

We designed the Ring Generators to look inviting enough for the player to want to try flying through them, giving them a noticeable bright color and smooth appearance. Figure 34 shows the in-game model for the ring generator design.

Originally, the Ring Generators were just floating Rings with a Matrix-like texture, but after receiving critique that they were too plain, we changed the design entirely. The new Ring Generators act as tiny thrusters, always spinning in a circle and naturally forming a ring. Particles were supposed to be utilized to better create the natural shape of a ring, but difficulties with the engine prevented this.

The old design for the Ring Generator was actually re-purposed into the goal ring for our levels' finish lines.

8.4.3 Crate

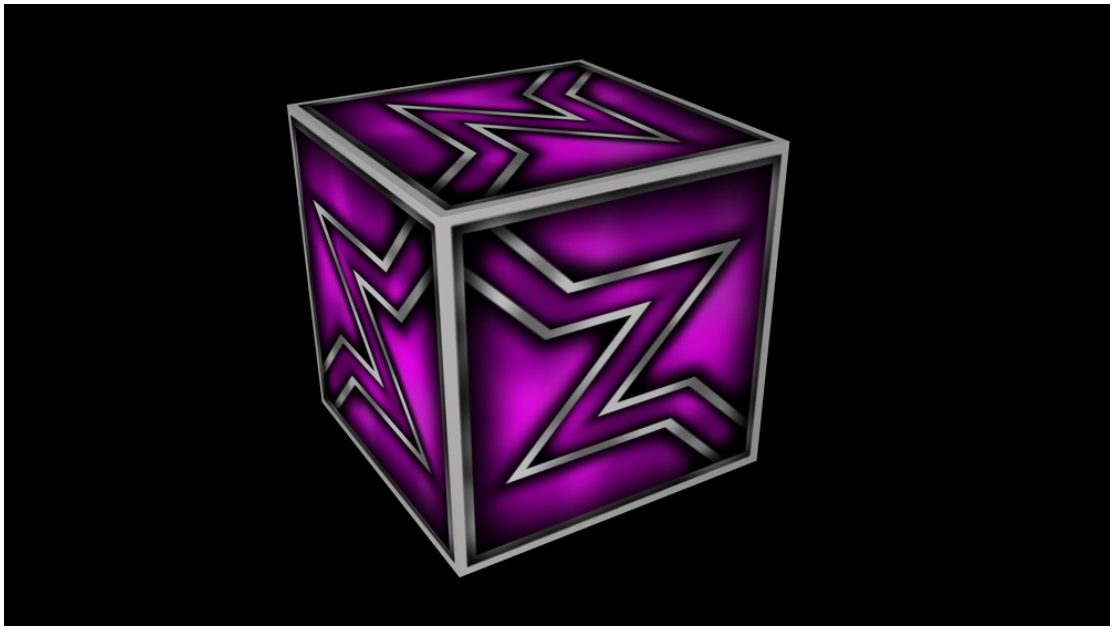


Figure 35: Crate Model

A Crate's appearance consists of a silver and violet box with a Z -like pattern on every side. This is shown in figure 35. Similar to our logic with our Item Cubes, we tried to design the Crate's look such that they would be something the player might want to try to hit, but not feel like they are an obstacle which could end the game if the player collided with

them. Pyramidal geometry would be both smaller targets and harder to see, as well as look somewhat dangerous because of their spiky nature. Therefore, we decided the object best for playing with the game's physics would be a neutral, 6-sided cube. We decided they would be the color violet because the Crate would be able to stand out against the environment and violet is far enough from the darker colors and red to be less threatening.

8.4.4 Downer

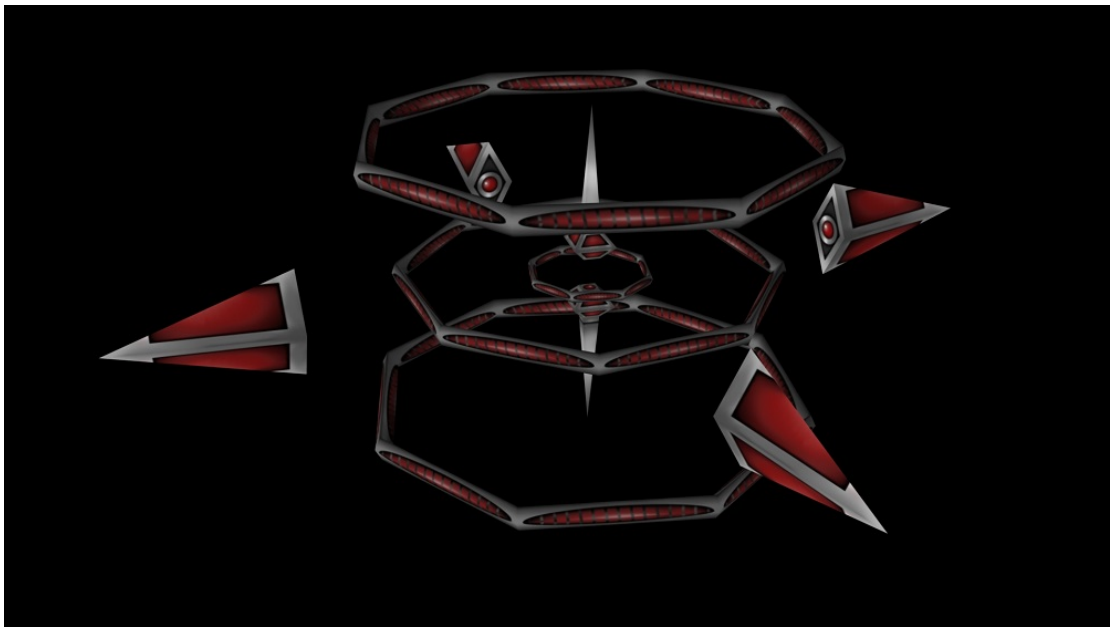


Figure 36: Downer Model

Figure 36 shows the in-game model for the Dower object. The Downer's appearance consists of a grey and red, floating, spiked, and spinning mechanism.

Originally, the Downer itself was going to be much smaller and emit a large red field several times its size, but as with the Item Cubes, the engine had trouble reading detailed textures with a transparency map. To prevent the whole object from looking like a plain, floating, transparent, red sphere, we instead up-scaled the Downer's spiky center contraption to a size closer to the player and dropped the red field concept. Even without the field, the Downer still stands out from other items due to its dark and red color scheme, still aided by its spiky, moving geometry.

8.5 Environment

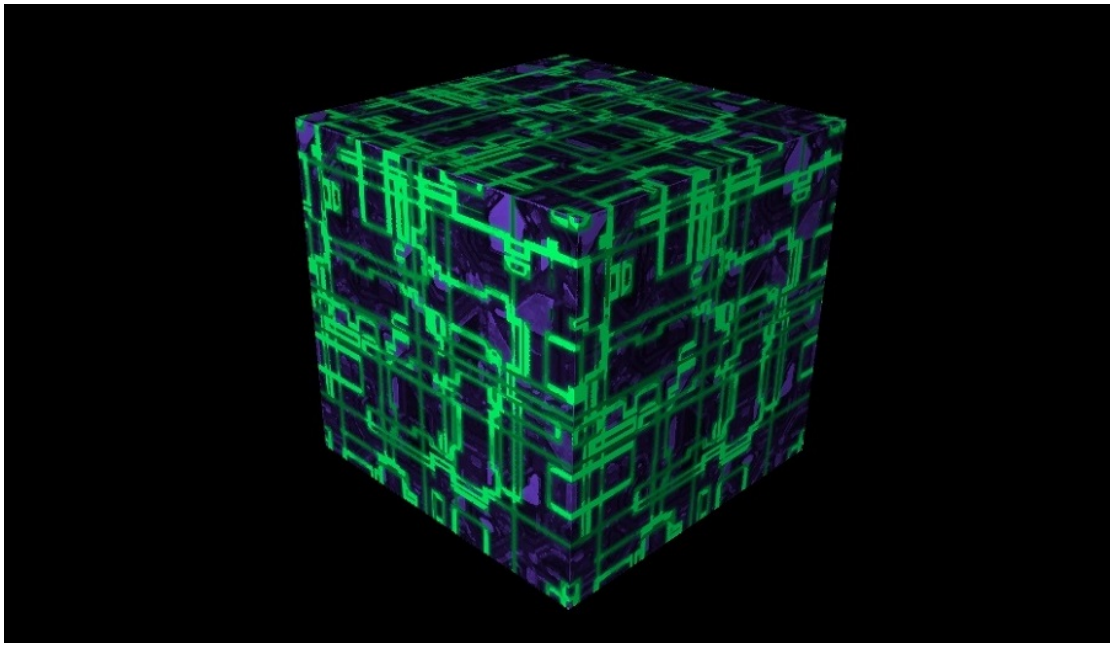


Figure 37: A lone building block of the city environment

The general environment takes significant inspiration from *Tron* (2010) and *The Matrix* movie series. Our environment geometry makes heavy use of some motherboard or circuit board-like textures with bright colors in sharp patterns on top of mechanical foundations. All pieces were modeled with simple geometry for modular level design and construction. An example of this is seen in figure 37.

8.5.1 Skybox

The skybox was something we had not created on our own before, so we spent a decent amount of time troubleshooting and testing. Initially, we planned to have the skybox's sides depict a cityscape in the distance similar to the immediate environment's geometry. After trying this however, the buildings looked flat and didn't give the desired 3D impression. We then decided to go for a simpler skybox consisting mainly of a starry sky. This looked good at first, but after several tests it became apparent the texture, when blown up to sky box size, looked pixelated. After more testing, better textures were made until the pixels weren't so obvious.

8.6 Menu UI

The Menu art's inspiration comes from various sci-fi media where a device can project a menu with a touch interface. The light blue generally matches the player's path colors but at the same time is a neutral color often seen in sci-fi media.

8.7 Color and Texture

8.7.1 Player Color and Textures

The player's colors have to contrast from the mostly dark, but patterned bright environment pieces. For this, the main body is pitch black, white is the base color for armor and yellow is used for its edges and guards. These color choices resemble the Protoss race from *StarCraft*. Between design passes, the character's general color scheme did not change. The alternate player colors change the blue to red and the gold armor edges to silver. These palette changes complement each other while also being different enough to distinguish when close or at a distance.

Textures on the player make use of a simple technique to imitate depth where there is actually a flat surface, which involves careful use of shadows and shading. Small details were also added with this technique from the concept to 3D realization, such as the plated-look on the tassets, and the extra edges on the belt, chest strap, and greaves.

Texture files sizes for the player character's different pieces range between 256x256 and 512x512 pixels. Texture files of resolutions any larger would be unnecessarily large and only hurt the game's performance.

Figure 38: The texture file for the player character's lower body, optimizing the model's symmetry

8.7.2 Object Color and Textures

Early on, it was decided that like other games, red is generally a threat while brighter colors, especially yellow, are beneficial. This is the colod scheme used in the player model texture, as seen in figure 38. Objects like the Downers are dark and red while the Rings are primarily yellow.

8.7.3 Environment Color and Textures

It was decided that the sky should be dark and not draw attention from the player as they navigate the bright environment. Since early concepts it was decided to make the sky starry and simple. City geometry for the environment was made to stand out against this sky. A full list of environment geometry textures can be found in Appendix A.

8.8 Lighting

Lighting was an important feature to implement correctly to give the game the desired appearance. The C4 Engine comes equipped with five distinct types of light sources that

can be used to illuminate a scene. There are some differences among these types as to whether the light is a point source or an infinite directional light and whether the light projects a texture map onto the scene[18]. These lights include:

- Ambient Light
- Infinite Light
- Point Light
- Cube Light
- Spot Light

The game uses Ambient and Infinite lighting for environmental effects, and Point lights for smaller detail effects. We used each of these sources in combination with each other to achieve the desired look of the game.

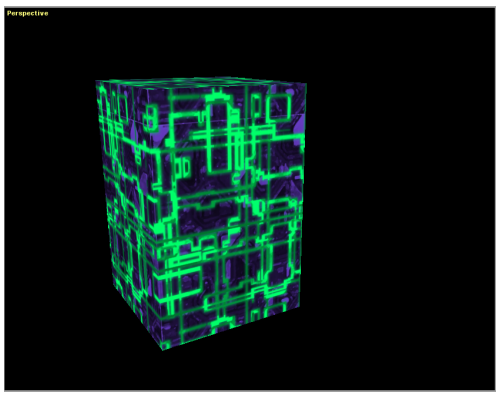
8.8.1 Ambient Light

An ambient light source is one that affects all objects in the world equally. All objects in a scene are lit on all surfaces with a specified brightness and color. This light source simulates light that has been reflected many times and seems to illuminate everything. From every angle and every direction, the objects are lit equally. Ambient lights are used to provide a basic view of the objects in the game without shadows. Ambient lighting can be combined with *ambient occlusion* to represent how exposed each point of the scene is, affecting the amount of ambient light it can reflect.

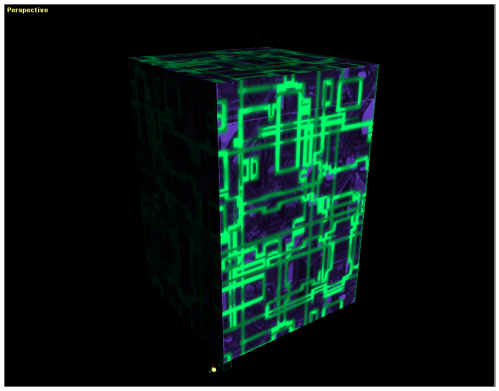
Ambient occlusion refers to a rendering technique that is used to calculate how exposed each point in a scene is to ambient lighting. For example, the inside of a cave would be more occluded, therefore darker, than the outside, and would only become more occluded the deeper the inside goes. This applies to any object that is being occluded, such as an object surrounded by other objects or perhaps within shade. When combined with ambient light, ambient occlusion produces a diffuse, non-directional lighting source throughout the scene, casting no clear shadows, but with enclosed and sheltered areas darkened.

The ambient light within our game was set to a low intensity to give the environment a darker look. The goal was to simulate a nighttime city-like world with glowing environmental features. The low ambient light darkened the textures of the world and allowed the

structures to be illuminated and cast shadows without the scene coming off as too bright. This concept can be seen demonstrated in figure 39.



(a) Cube with default white ambient light



(b) Cube with near black ambient light and source light

Figure 39: Comparison between an environment with white ambient light in a scene (a) and an environment with near black ambient light (b)

8.8.2 Infinite Light

Infinite light is a good source when illuminating a large landscape, typically in an outdoor setting. The infinite light source illuminates a scene with a custom constant defined brightness from a single direction. When the light is placed in the world, the orientation of the light is what is considered when calculating illumination and shadows. The position of the light source isn't taken into account[18]. Multiple infinite light sources can be added to a scene to provide multiple shadow angles, although there is typically only one infinite light source added per world.

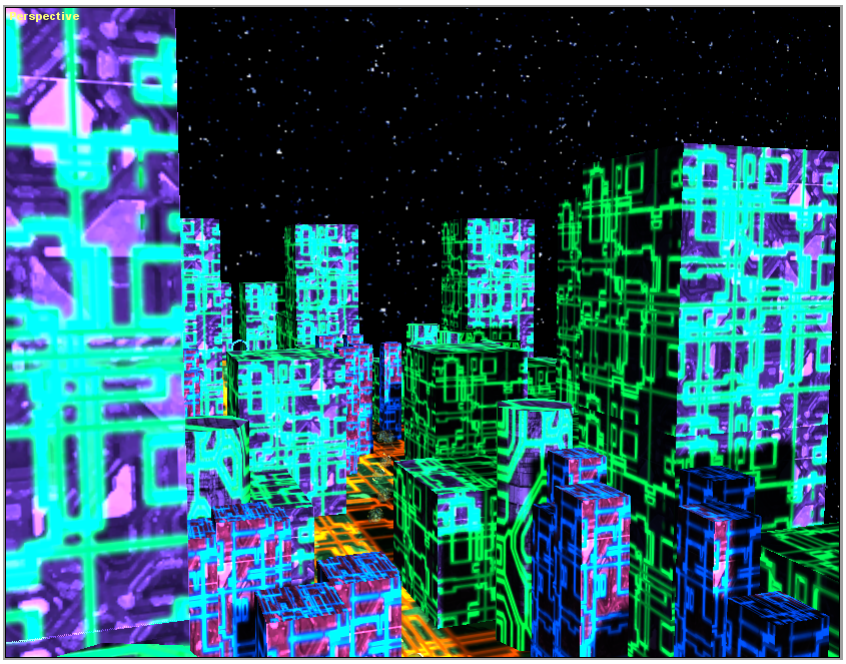


Figure 40: Example of the game environment with an infinite light applied

C4 infinite light uses a technique called *cascaded shadow mapping*, which renders shadows into a scene. Shadow mapping hardware on the GPU is used to perform per-pixel depth comparisons. This causes correct shadows to be applied to any object within the cascade ranges of the camera. The shadow map created by the infinite light source is completely dynamic and updated every frame[18]. This same technique is used in the environment in figure 40.

To create the dark world environment, we set the ambient light of the world to near black with a single, infinite depth light. The depth light was positioned at such an angle to allow light to be cast on much of the individual environment geometry, all the while casting shadows and occluding other parts of the geometry. This helps the environment look less monotonous even though the geometry is very basic and mostly the same.

8.8.3 Point Light

A point light shines light from a single point out to a set radius, beyond which the intensity of the light will gradually reduce until it reaches zero. Point lights can cast shadows supplied by up to six two-dimensional texture maps arranged in a cube. Point lights fade and then disappear as the camera moves farther away from them. All of these properties can be set in the world editor.[18]

We used point lights to illuminate the speed boost and ring enhancement items the player can collect. The point lights give extra light to the items which, along with their color and texture, helps distinguish them from the rest of the scene. The point lights attached to the items glow pink or yellow, depending on whether they are the Speed Boost or Ring Boost, respectively. They are more noticeable as a result without adding too much extra light to the scenes they are in.

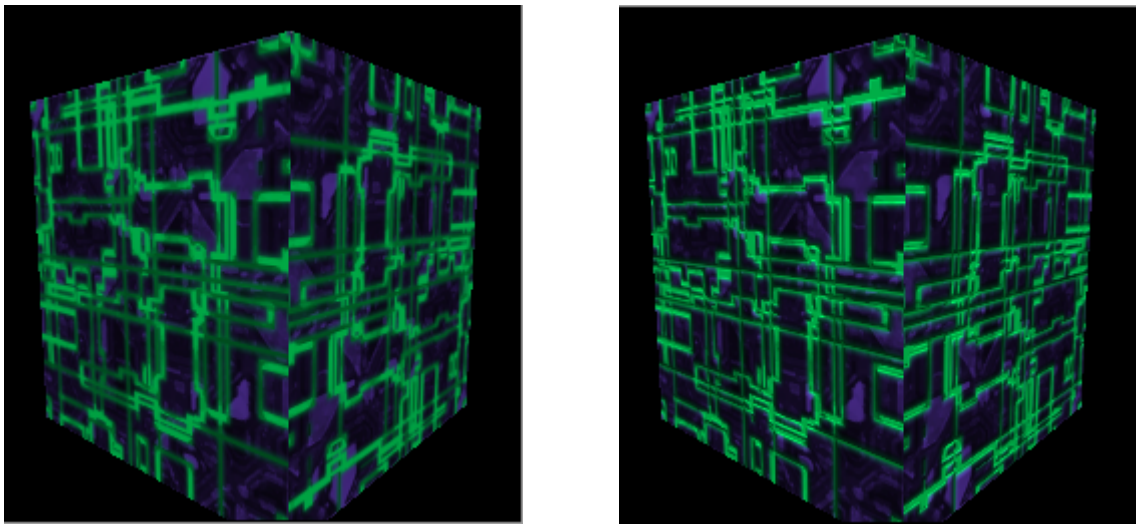
8.8.4 Spot Light

A spot light is similar to a point light, but it only projects light in a cone instead of a sphere. Like a point light, a 2D texture map can be provided as the light's shadow[18]. A common use for spot lights is when simulating a flashlight or street lamp. We used a spot light to illuminate the space around the player character on the path. Multiple spot lights were also used in certain parts throughout the map environments, to diversify the atmosphere.

8.9 Texture Mapping

8.9.1 Bump Maps

A bump map is another name for a Normal Map, which is a texture that encodes normal vectors so that a flat surface appears to be bumpy. This gives a feeling of depth to objects without the need of adding any extra geometry to models. Bump maps make use of the alpha (transparency) channel to contain information that affects different aspects of the material. If the channel has any parallax data, then the parallax mapping calculation is applied in the rendering process [19]. Since the textures we were using in the game were mapped to flat objects, we did not put any parallax data in the bump maps. For our texture normal maps, we decided to have the emission lines (the green or blue lines) look indented in the geometry, as opposed to making them look extended. Figure 41 shows how bump maps appear when applied to an existing texture.



(a) Cube without bump/normal map texture (b) Cube with bump/normal map texture

Figure 41: Comparison between a material without bump maps applied (a) and with bump maps applied (b)

The normal maps are calculated from the alpha transparency information stored in a separate file. This separate image is a copy of the outline of a normal texture except in gray scale. The areas that are to be indented are darker than the areas that are supposed to stick out. Firgure 42 shows the normal map that is calculated from the alpha transparency map. We used bump maps to give a sense of depth to the environment geometry, as well as to the Speed Boost item.

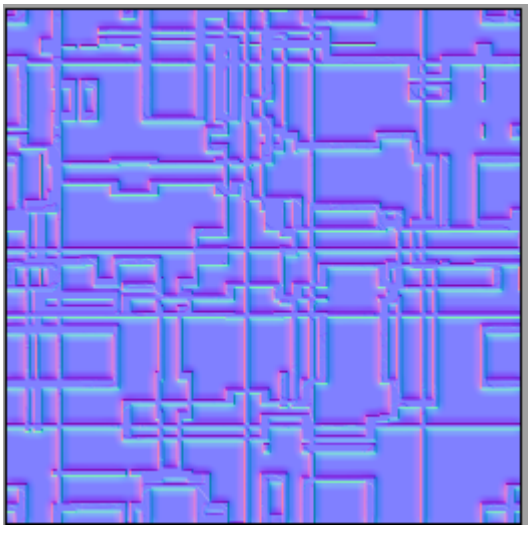
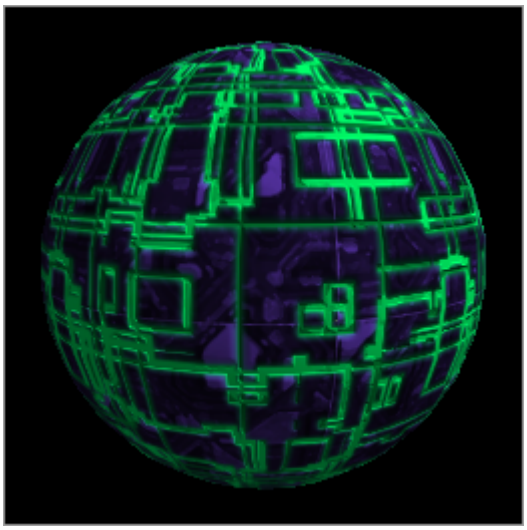


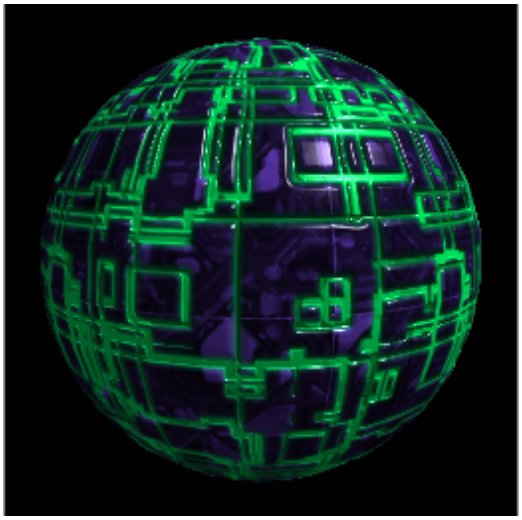
Figure 42: Normal Map calculated from alpha transparency texture

8.9.2 Specular Maps

Specular maps in the C4 engine refers to textures that control the specular reflection of color across the texture. Specular reflection is a mirror-like reflection of light where a surface where the light is coming from a single direction reflects it in a single outgoing direction. The Specular Maps plugin in the C4 world editor modulates the specular reflection. When this is combined with the bump map, the light reflects off the "outermost" sections of the texture. The sections where the light is reflected appear to be outlined in white (or whichever color the specular reflection is set to) and give off the effect of reflection. Figure 43 compares the differences between a bump texture with and without specular reflection applied.



(a) Bump Sphere without specular reflection



(b) Bump Sphere with specular reflection

Figure 43: Comparison between a material without specular reflection applied (a) and with specular reflection applied (b)

8.9.3 Emission Maps

Emission maps, when referring to the C4 Engine, are texture maps that, when applied to a material, add extra bloom or glow to the texture. The emission map controls the emission color, which is the color of light that the surface appears to emit by itself. The emission color is added to the ambient light that the surface reflects if this attribute is used in a material. Emission color is multiplied by the color read from the emission map. Emission maps are useful when there is a texture that needs to be seen in areas with little to no light. To save resources, emission maps can give off the appearance of an object

emitting light without actually doing so. This proved useful for our textures. We used an emission map of the foreground lines to give off a glowing effect. These textures can be seen when shadows are cast over the environment, and when combined with radiosity give off a glowing illusion. When we created emission maps for the environment geometry, we decided to have the highlighted green lines (sometimes blue or orange) be in the foreground and shine through shadowed areas. Figure 44 compares the differences between a texture with and without an emission map applied to it.

(a) Cube with emission map texture

(b) Cube without emission map texture

Figure 44: Comparison between an environment with white ambient light in a scene (a) and an environment with near black ambient light (b)

When all of these elements are combined together, we achieved the result we were aiming for. The textures give the objects depth, shadows, reflection, and a bit of realism that makes for convincing environments, as opposed to flat environments. Figure 45 shows the results of textures applied with normal maps, emission maps, and specular reflection.

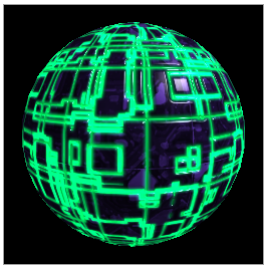


Figure 45: Final texture product with specular reflection, bump maps, and emission maps.

9 Sound

9.1 Sound Effects

Almost all sound effects in the game were created through a browser-based waveform-manipulation sound effect creation program. One primary reason for this was that recording capabilities and professional sound effect software, were inaccessable. Also, the distinctly digitized sound of the sound effects produced by that program pairs fairly well aesthetically with the largely science-fiction visual design.

The process of creating the sound effects generally involved pressing the random button in the program until the output was perceived as being close to a sound that would fit a specific sound effect in the game, then taking the current sound and carefully adjusting parameters such as starting frequency, number of steps, and the combination of waveforms used.

Regarding design choices, one relatively consistent theme between almost all of the sound effects is a distinct rising or falling effect, often matching the direction of adjustment to the player's speed; for example, passing through rings or using a speed boost will trigger sound effects that rise in pitch over their duration, while colliding with a downer will trigger a sound effect where the pitch falls. The epitomization of this concept is the 'ambient' looping sound generated by the light path and originating at its front; both the pitch and speed of the path's sound are proportional to the current speed of the player, providing direct feedback in regards to the current speed without having to clutter the display with HUD elements.

9.2 Music

9.2.1 Musical Inspirations

Musical sources with distinct styles that influenced the design of the game music include the *Sonic the Hedgehog*, *Mario Kart*, and *F-Zero* game series, all of which notably emphasize the effect of speed on a player's experience (through racing elements) and thus represent appropriate inspirations for the music here.

9.2.2 Content Organization

Originally, one menu track and one gameplay track were created. The structure of the gameplay track included four distinct sections of equal length and similar structure. As

the gameplay content moved toward its current shape of four relatively brief levels, the decision was made to split up the gameplay track into four tracks, one for each section, so that players would be more likely to hear all of the audio content within normal play.

9.2.3 Menu Music Design

Since interaction with the game menus is a more relaxed setting than main gameplay, design choices there include a more laid-back tempo, some softer instrument choices (mostly synthetic), and a simple, repeating three-chord structure which is always either holding the tonic or ascending back to it. The piece was extended by having the overall structure repeat once, with the second time including a backbeat similar to those used in title music from the *Sonic the Hedgehog* series.

The menu music is in the key of Eb major, partially due to a vague notion, shared around the choral music community, that the Eb major chord feels the most peaceful.

9.2.4 Gameplay Music Design

A particularly crucial element of the gameplay music instrumentation is the rather frantic bass guitar line, which provides the melody for the first half of the first track, uses that melody as a countermelody in the second half of that track, and then offers various countermelodies throughout most of the rest of the gameplay music, while always staying within the intended chord structure. The bass's frantic pace specifically supports the high-speed gameplay. The only part where the bass cuts out is in the piano feature in the first half of the third section, which is intended to serve as something of an interlude in the context of playing through the levels in sequence, and thus avoid having all of the gameplay music feel too similar in pace. The use of synth trumpet for the melody in the second half of the first track is particularly notable for being inspired by music from *F-Zero*, especially tracks like Mute City and the cup completion fanfare.

The first track of the gameplay music is in the key of D (dorian), one half-step below the menu music; this ostensibly creates a sense of dropping into gameplay and then rising back to the menu interface (which makes sense since the menus represent the more high-level interaction with the game). The latter three tracks are based in different keys (D major, F major, and G minor, respectively), but they were chosen in a way that the four tracks can flow smoothly together (since they were originally one track).

9.2.5 Balancing

Volume balancing was fairly simple, with the primary focus being to make sure that the melody instrument in a section of a piece is usually the loudest at that time.

Some deeper logical thinking was required for setting pan values. First, giving each instrument a unique pan value is important so that they can clearly be recognized independently on a spacial basis. This, however, opens up another potential problem, particularly in the gameplay music; a reasonable left-right balance at all times is greatly preferable. (As an example of the opposite, a segment where all instruments are noticeably left of center will rapidly become irritating and feel somewhat unnatural.) The approach taken to solve this was to place the most persistent instrument(s), such as drums and bass, in/near the center and then move outwards, going back and forth between left and right, to place the more incidental instruments, like the trumpet and lead guitar.

10 Project Management

10.1 Asset List

We compiled a list of all of our game assets using the Google Spreadsheet application included in Google Drive. This allowed us to easily share and modify any assets across the entire team. We tracked several aspects of each asset, including file name, status, completion percentage, and priority. The spreadsheet was programmed to automatically calculate our overall progress based on the estimated percentage of completion for each asset. This gave us a good sense of progress, although the completeion estimation was only speculation on our part. The complete asset list is included in appendix C.

10.2 Asana

Asana is a free project management tool. We used it to track who was working on what tasks, and when they should be done[20]. Figure 46 shows a screen shot of the Asana workspace and some tasks that are recorded.

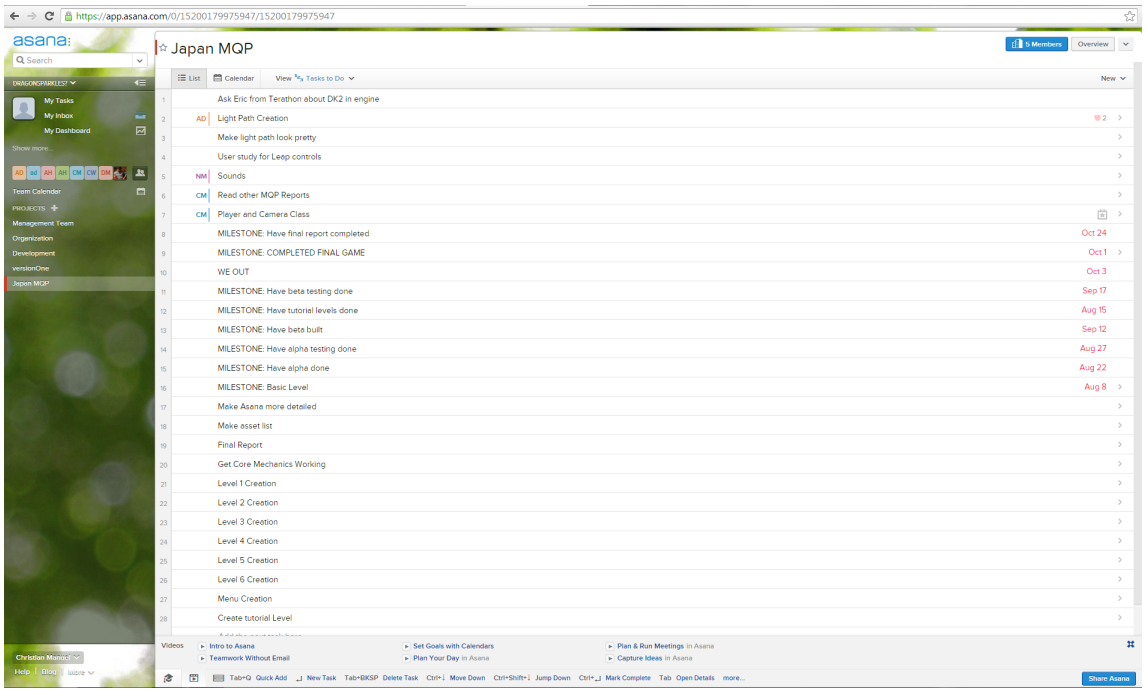


Figure 46: Asana online project management application

Initially, we started off with an outline of long-term goals. We recorded major dates such as by when to be finished with the initial design process, when to have all concept art

done, when to have all core mechanics implemented, and when our project presentation will be. As days went by and the project progressed, we regularly updated the task list with whatever tasks came up. These tasks could range from small fixes to large problems. Each task that was added was given a time in which it needed to be completed, as well as a priority as to the importance of the task. Tasks with higher priority needed to be completed, at the very most, by the time that was specified, or at the very least as soon as possible. Specific members can be assigned to work on tasks that are added or a group of members can work on a task. Every member that is assigned to a task is continually updated as to the status of the task; whether any changes have been made, whether it has been completed, or whether it has been deleted. When a task's status has changed, the users assigned to the task are notified via the email address that they sign-in with.

This was a very useful tool in keeping the team organized in what we needed to do and how to go about doing it. Each member was aware of the task that he was assigned to, so there was no need for delay in progress due to not having any work to do. Everyone was always busy with something. It made communicating with the team members easier as well because we did not have to allocate time to have a group meeting to discuss every single task that came up. This saved us a lot of time.

11 User Studies

During beta testing, when we had all core mechanics implemented and levels built, we did some very informal user testing using ourselves and people from the lab as subjects to see if the game's movement controls were intuitive. The language barrier with the Japanese students made it an even better chance to test how intuitive our mechanics were. The goal was to have the players figure out how to play the levels by their actions, with us only telling them the bare minimum.

We had the players play through each level at least once to get feedback on each level. We watched their hands and asked them for feedback. Based on these tests, we changed the in-game hand movement to only reflect the movement and gestures that have an effect. The survey used for testing is included in appendix B.

From their initial responses, we learned that the game was too hard from the first level. We learned that the skill curve was not what we wanted as far as level progression goes; the first level was too difficult and the second level was too easy. Almost no one completed the 3rd level. Seeing these results, we changed the obstacles and items in the levels to better try to accommodate the new players.

Later testing showed improvement with first-time player performance and better navigation of the game.

12 Post Mortem

12.1 What went right

Overall, we were very pleased with how the game turned out. To us, it is a fun game to play, and that was the general consensus of the testers as well. We feel that the concept of the game, especially the combination of the Oculus Rift and Leap Motion controller, is very compelling. We were very pleased with the graphical quality of the game. The C4 engine enabled us to use high quality textures and shading effects without being too resource intensive, providing for smooth gameplay. It looks very good on the Oculus Rift and the controls are smooth and easy. The sound effects are good quality and accomplish what they were designed to do. We accomplished most of the goals with the game that we set out to do. This is an immersive, unique experience that lets the player really feel like they are in the game world.

The tools and hardware we chose to use worked well for us. Both the Rift and the Leap were easy to work with and were easy to integrate with C4. While we had some problems with C4, (see below) its asset pipeline was easy to use. Many game projects struggle with their pipeline, but we didn't have any significant problems with it.

12.2 What went wrong

During our work on the game, several factors made us less efficient than we could have been or took away time we could have used to work on the game. This led to some planned features of the game not being included in the final version, such as levels 5 and 6 and multiplayer mode. We also didn't have as much time to test and refine the existing levels as we would have liked.

The biggest source of inefficiency was working with the C4 engine without any prior experience. The engine is very powerful, but it has a steep learning curve. Reference documentation (function, class descriptions, etc.) for the engine was very thorough, but tutorials and guides to engine features (such as collision detection, lighting, networking, etc.) were few and did not go into great detail. This meant that in order to use many of the engine's features, we had to either ask about them on the C4 Engine forum, reverse-engineer example games, or just experiment until it worked. These are perfectly good techniques, but they take time. Most of the time we spent working on the project went to troubleshooting C4's less documented areas.

We did not prepare anything related to the project before arriving in Japan. Because of this, and because we took some time adjusting to life in Japan, we had a slow start on the game. Once we did start, we spent more time designing the game than we needed to.

While working on our game, each team member was also working on a side project with a team of students from Osaka University. The side projects were due before we left WPI, (which is different from previous years) so working on those took up a lot of time we could have been using on the game. The side projects took almost all of our time for about two weeks.

13 Conclusion

Our goal for this project was to make Hikari Michi the most immersive, enjoyable game that we could, and we feel that we met that goal, despite the challenges we faced in the process. Working with two unfamiliar Input/Output devices and a new engine was difficult, but we were able to complete most of the goals we had planned for our game. Through our Major Qualifying Project in Japan, we learned much about the expanding fields of immersion and virtual reality, and gained great experience in the various pipelines of interactive game development.

References

[1] Oculus VR, LLC, "Oculus developer guide, sdk version 0.4," October 2014.

[2] "Leap Motion System Architecture Overview," July 2014. https://developer.leapmotion.com/documentation/skeletal/cpp/devguide/Leap_Architecture.html.

[3] "Leap Motion API Overview," July 2014. https://developer.leapmotion.com/documentation/skeletal/cpp/devguide/Leap_Overview.html.

[4] Terathon, "C4 Engine Architecture," November 2014. http://www.terathon.com/architecture.php.

[5] "Programming Introduction to the C4 Engine," January 2014. http://www.terathon.com/wiki/index.php/Programming_Introduction_to_the_C4_Engine.

[6] "Lights and Shadows," August 2012. https://www.kickstarter.com/projects/1523379957/oculus-rift-step-into-the-game.

[7] "Leap Motion Product," October 2014. https://www.leapmotion.com/product.

[8] Michael Antonov, Nate Mitchell, Andrew Reisse, Lee Cooper, Steve LaValle, Max Katsev, "Oculus vr sdk overview," October 2013.

[9] Leshy Labs LLC 2013, "Leshy SFDesigner," August 2014. http://www.leshylabs.com/apps/sfMaker/.

[10] Evan A. Suma, Zachary Lipps, Samantha Finkelstein, David M. Krum, Mark Bolas, "Impossible Spaces: Maximizing Natural Walking in Virtual Environments with Self-Overlapping Architecture,"

[11] Enrico Gobbetti and Riccardo Scateni, "Virtual Reality: Past, Present, and Future,"

[12] Jonathan Strickland, "How Virtual Reality Works."

[13] A. D. T. F. N. M. P. H. Richard Yao, Tom Heath, "Oculus vr best practices guide," January 2014.

[14] D. Games, "*AaaaaAAaaaAAAaaAAAAaAAAAA!!! For the Awesome*," 2011.

[15] A. Games, "*HAWKEN*," 2014.

[16] "*Minecrift*," 2014.

[17] P. Pixels, "*Proton Pulse*," 2013.

[18] "Lights and Shadows," October 2014. https://www.terathon.com/wiki/index.php.

[19] Terathon, "C4 Engine Glossary," August 2014. http://www.terathon.com/wiki/index.php/Glossary.

[20] "Asana," October 2014. https://asana.com/product.

Appendix A: All Environment Textures

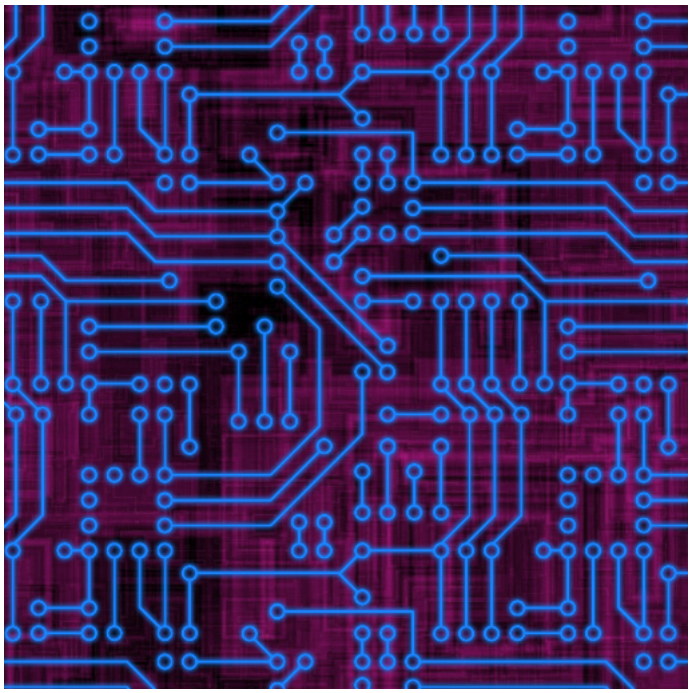


Figure 47: Blue version of a circuit board pattern

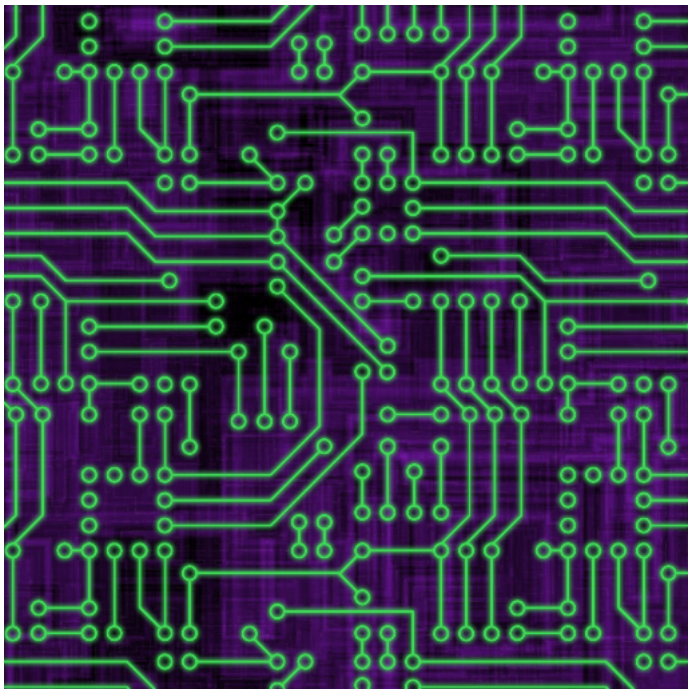


Figure 48: Green version of a circuit board pattern

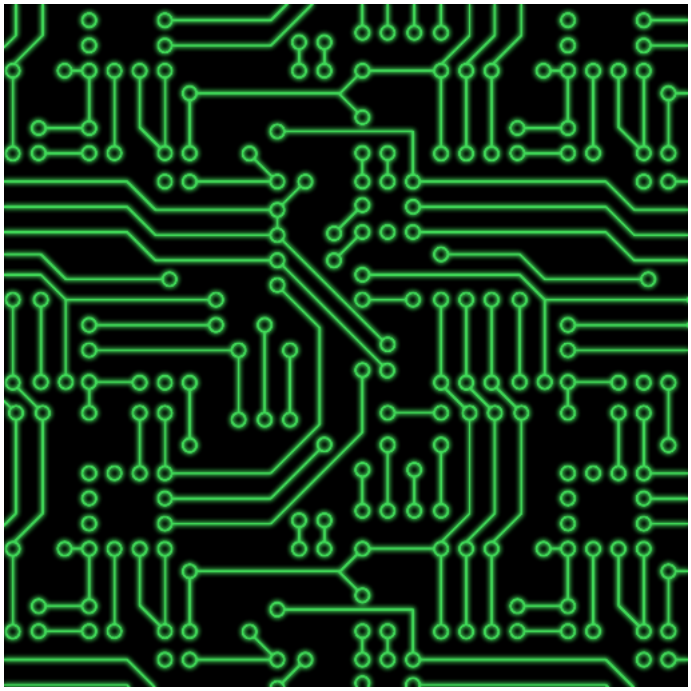


Figure 49: Emission map for green version of a circuit board pattern

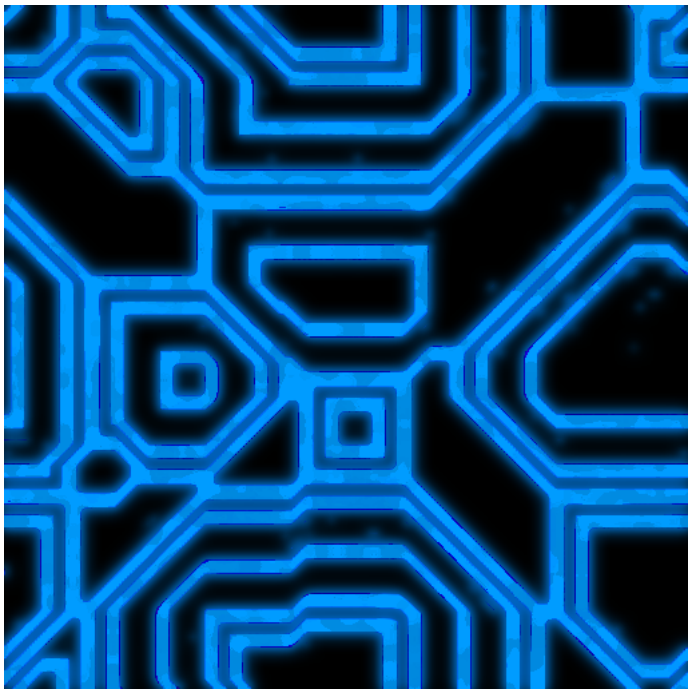


Figure 50: Emission map for blue alternate environment texture

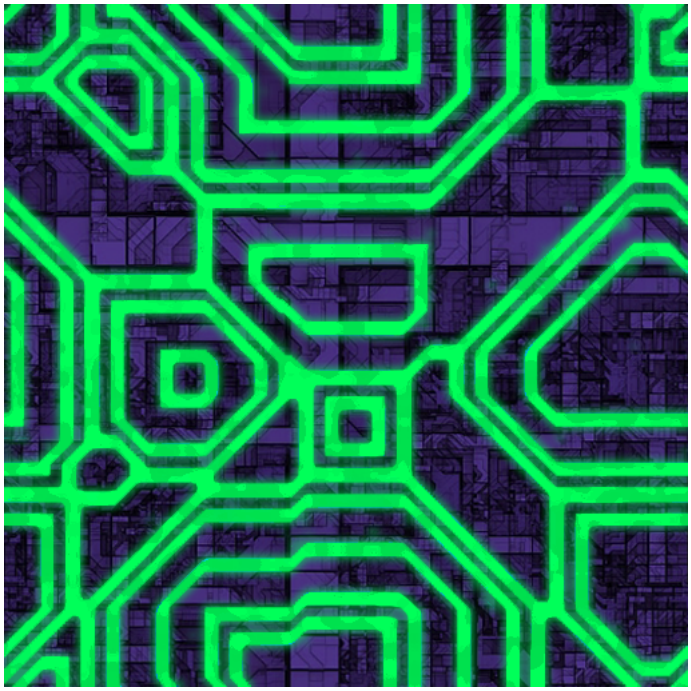


Figure 51: Alternate green environment texture

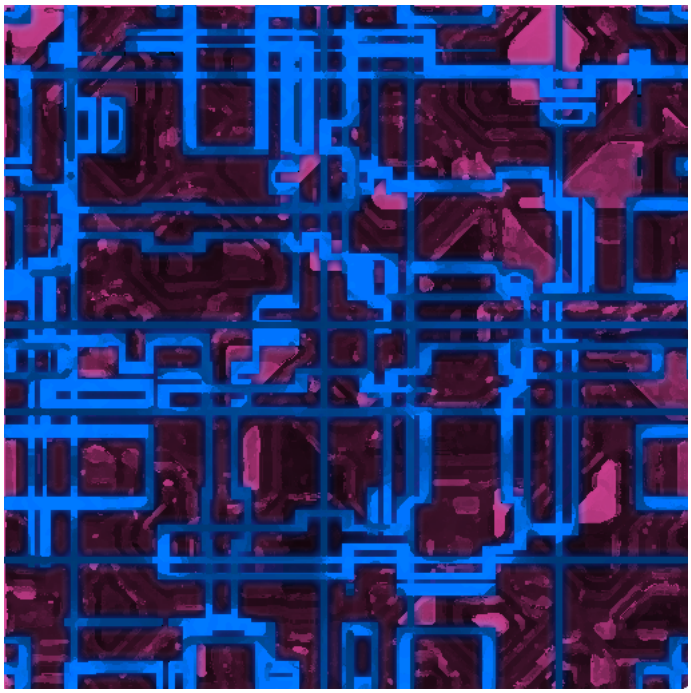


Figure 52: Blue version of the environment texture

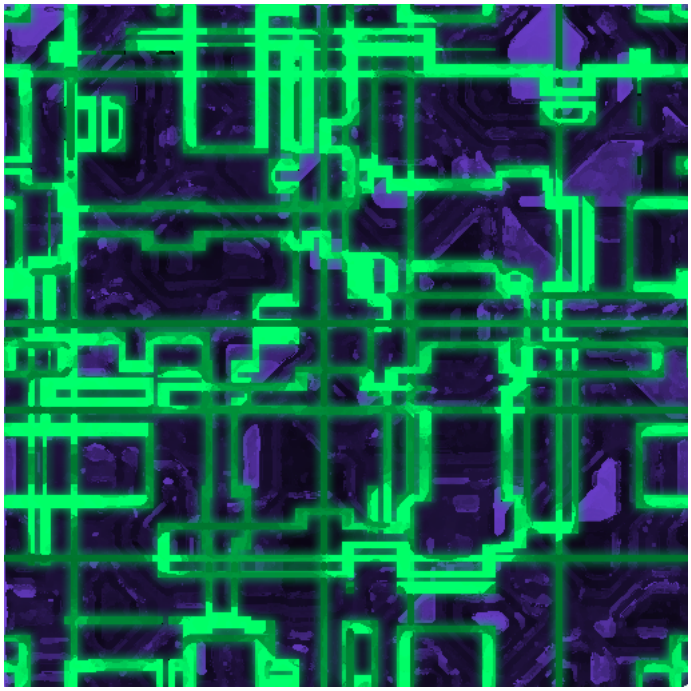


Figure 53: Green version of the environment texture

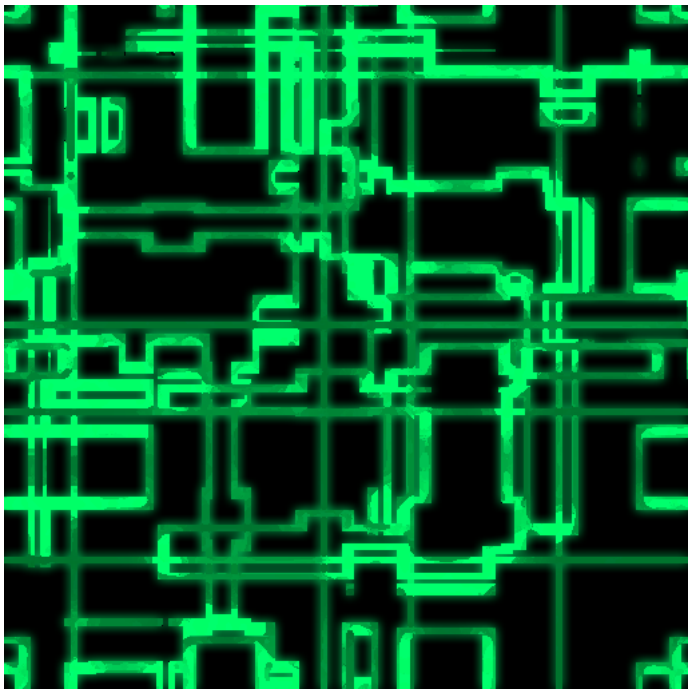


Figure 54: Green emission map for the green environment texture

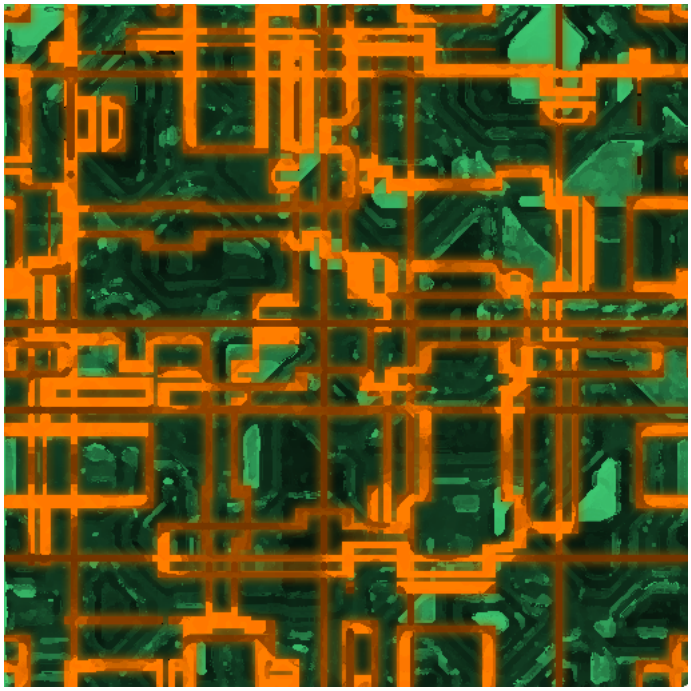


Figure 55: Orange version of the environment texture

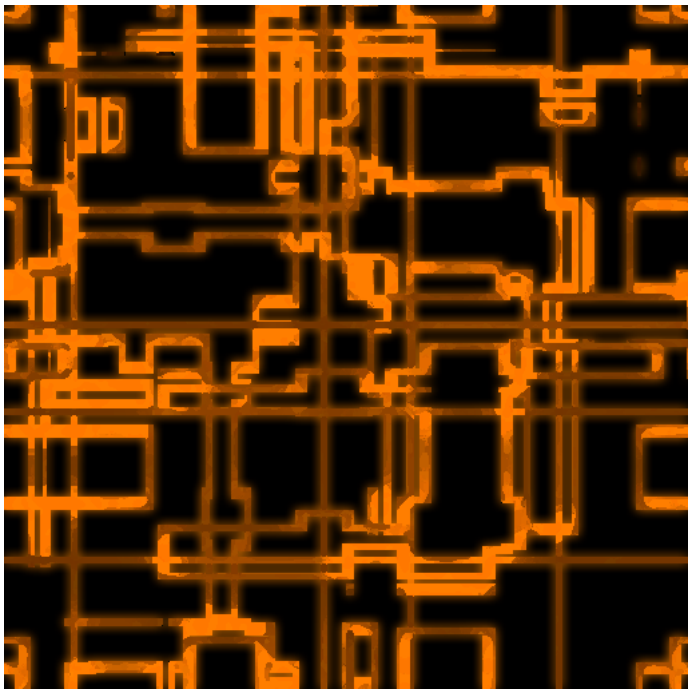


Figure 56: Emission map for the orange environment texture

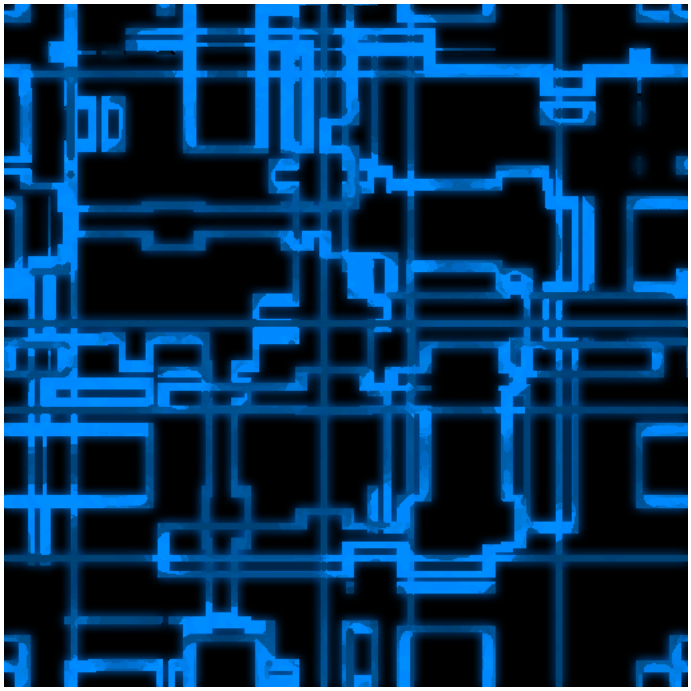


Figure 57: Emission map for the blue environment texture

Appendix B: User Study Survey

User Survey

Mark the box that best matches your opinion. 1=strongly disagree, 5=strongly agree.

	1	2	3	4	5
I found the game fun and enjoyable to play.					
The controls were intuitive and felt responsive.					
I did not become nauseated from playing the game.					
The game provided a high level of challenge.					
The difficulty increased appropriately with each successive level.					
The menus were easy to navigate.					
The game ran smoothly, without obvious glitches/bugs.					
The sound assets were well balanced in volume.					
The sound assets were high quality.					
The visual elements (assets, framerate, resolution) were high quality.					
The graphics and sound fit the gameplay experience well.					
I am an experienced game player.					
I have substantial previous experience with the Oculus Rift.					
I have substantial previous experience with the Leap Motion.					

Additional Comments:

Appendix C: Asset List

Models	File name	Status	Percent Done	Date Completed	Priority	Completion Percentage	Notes
player model		finished	100.00%		High	100.00%	
hand model	Gauntlet.dae / gauntlet.mdl	finished	100.00%	8/8/2014	High		
Large building		finished	100.00%		Medium		These mostly make up a level's walls
Small building		finished	100.00%		Medium		For more detailed areas
More buildings		finished	100.00%		Low		For variety, if there's time
Downer model		finished	100.00%		High		
Speed up pickup		finished	100.00%		High		
Ring expander pickup		finished	100.00%		High		
Small Ring model	Ring_Yellow.dae / ringSmall.mdl	finished	100.00%		High		
Medium Ring Model	Ring_Yellow.dae / ringMedium.mdl	finished	100.00%		High		
Large Ring Model	Ring_Yellow.dae / ringLarge.mdl	finished	100.00%		High		
Crate Model		finished	100.00%		Low		
Item container Model		finished	100.00%		Medium		
Main menu billboard		finished	100.00%		Medium		
Light Path Model		finished	100.00%		Medium		
Mine Model		finished	100.00%		Low		
Hand-held speed up		finished	100.00%		Medium		
Hand-held ring expander		finished	100.00%		Medium		

Textures	File name	Status	Percent Done	Date Completed	Priority	Completion Percentage
Player Model texture		finished	100.00%			88.89%
Hand Model Texture	Gauntlet.tga / gauntlet.tex	finished	100.00%			
Small Building Texture		finished	100.00%			
Large Building Texture		finished	100.00%			
Other Building Textures		finished	100.00%			
Downer Texture		finished	100.00%			
Booster item texture		finished	100.00%			
Ring expander item texture		finished	100.00%			
Light Path Texture		finished	100.00%			
Mine Texture		finished	100.00%			
Small ring	Ring_Diffuse.tga / Ring_Diffuse.tex	finished	100.00%			
Medium ring	Ring_Diffuse.tga / Ring_Diffuse.tex	finished	100.00%			
Large ring	Ring_Diffuse.tga / Ring_Diffuse.tex	finished	100.00%			
Crate		finished	100.00%			
Item container		finished	100.00%			
Billboard		finished	100.00%			
Hand-held speed up		DNE	0.00%			
Hand-held expander		DNE	0.00%			
Level start		DNE	0.00%			
Level end		finished	100.00%			
Skybox top		finished	100.00%			
Skybox left		finished	100.00%			
Skybox right		finished	100.00%			
Skybox bottom		finished	100.00%			
Skybox front		finished	100.00%			
Skybox back		finished	100.00%			
Floor		finished	100.00%			

Concept Art	File name	Status	Percent Done	Date Completed	Priority	Completion Percentage
Player concept art		finished	100.00%			100.00%
Environment Concept Art		finished	100.00%			
Downer Concept Art		finished	100.00%			
Speed item		finished	100.00%			
Slow down item		finished	100.00%			
Mine Concept Art		finished	100.00%			

Images	File name	Status	Percent Done	Date Completed	Priority	Completion Percentage
UI button		finished	100.00%		Low (C4 default UI elements are fine)	76.92%
Level 1 picture		finished	100.00%			
Level 2 picture		finished	100.00%			
Level 3 picture		finished	100.00%			
Level 4 picture		finished	100.00%			
Level 5 picture		DNE	0.00%			
Level 6 picture		DNE	0.00%			
UI background		DNE	100.00%			
UI slider bar		finished	100.00%		Low	
UI slider pointer		finished	100.00%		Low	
Loading screen		DNE	0.00%			
UI font		finished	100.00%		Low	
Light particle		finished	100.00%			

Animations	File name	Status	Percent Done	Date Completed	Priority	Completion Percentage
Item Crush Animation		finished	100.00%			100.00%
Hand idle		finished	100.00%			
Ring expander idle		finished	100.00%			
Speed boost idle		finished	100.00%			
Downer idle		finished	100.00%			
Ring idle		finished	100.00%			

Sound Effects	File name	Status	Percent Done	Date Completed	Priority	Completion Percentage
Path generation	path.wav	finished	100%	8/4/2014		100%
Wind sound from increased speed	wind-howl-01.wav	finished	100%	8/4/2014		
Banking	bank.wav	finished	100%	8/4/2014		
Crash into environment	crash.wav	finished	100%	8/4/2014		
Mine explosion	Bomb-SoundBible.com-891110113	finished	100%	8/5/2014		
Downer electric buzz	Electricity-SoundBible.com-96551850	finished	100%	8/4/2014		
Afflicted by downer	downer.wav	finished	100%	8/4/2014		
Player character defeated	derez.wav	finished	100%	8/4/2014		
Powerup obtained	pickup.wav	finished	100%	8/4/2014		
Ring expansion activation	expansion.wav	finished	100%	8/4/2014		
Ring expansion end	contraction.wav	finished	100%	8/4/2014		
Speed boost powerup activation	speedboost.wav	finished	100%	8/4/2014		
Menu selection	select.wav	finished	100%	8/5/2014		
Menu cancel/back	cancel.wav	finished	100%	8/5/2014		
Go through small ring	rings.wav	finished	100%	8/27/2014		
Go through medium ring	ringm.wav	finished	100%	8/27/2014		
Go through big ring	ringl.wav	finished	100%	8/27/2014		

Music	File name	Status	Percent Done	Date Completed	Priority	Completion Percentage
Menu BGM		Complete	100.00%	8/26/2014		100.00%
Stage BGM 1		Complete	100.00%	8/20/2014		

Options Menu	Status	Percent Done	Date Completed	Priority	Completion Percentage
Show High Scores	finished	100.00%	8/22/2014		87.20%
Music Volume	finished	100.00%	8/27/2014		
Sound Volume	finished	100.00%			
Leap Sensitivity	DNE	100.00%			
Rift Sensitivity	DNE	80.00%			
Clear Scores	DNE	100.00%	8/22/2014		
Confirm Button	DNE	100.00%	8/22/2014		
Back Button	DNE	100.00%	8/22/2014		
Level Select Menu					
Level Select	finished	100.00%	8/22/2014		
Scroll between levels	finished	100.00%	8/22/2014		
Play Level Button	finished	100.00%	8/22/2014		
Show High Scores Butt	finished	100.00%	8/22/2014		
Back Button	finished	100.00%	8/22/2014		
Main Menu					
Level select menu	finished	100.00%	8/22/2014		
Options Menu	finished	100.00%	8/22/2014		
Quit to Desktop	finished	100.00%	8/22/2014		
Pause Menu					
Unpause Selection	DNE	0.00%			
Options Menu	DNE	0.00%			
Quit Level	DNE	0.00%			
Death Menu					
Restart Level	finished	100.00%	8/22/2014		
Quit Level	finished	100.00%	8/22/2014		
End of Level					
Show Score	finished	100.00%	8/22/2014		
Replay button	finished	100.00%	8/22/2014		
Menu Button	finished	100.00%	8/22/2014		
Next Level Button	finished	100.00%	8/22/2014		

Level Code	Status	Percent Done	Date Completed	Priority	Completion Percentage	
Ring Expander ite	finished	100.00%	9/28/2014	High	83.33%	
Speed boost item	finished	100.00%	9/28/2014	High		
Small Rings	finished	100.00%		High		
Medium Rings	finished	100.00%		High		
Large Rings	finished	100.00%		High		
Downer Behavior	finished	100.00%		High		
Level 1 Complete	finished	100.00%		High		
Level 2 Complete	finished	100.00%		High		
Level 3 Complete	finished	100.00%		High		
Level 4 Complete	finished	100.00%		High		
Level 5 Complete	DNE	0.00%		High		
Level 6 Complete	DNE	0.00%		High		

Multiplayer	Status	Percent Done	Date Completed	Priority	Completion Percentage
Server Setup	WIP	50.00%			45.00%
Client Setup	WIP	40.00%			

Player Assets	Status	Percent Done	Date Completed	Priority	Completion Percentage	Description
Handle Animations	WIP	50.00%			88.57%	
Collision Detection	finished	100.00%				
Light Path	finished	100.00%				
Player Spline Interpolation	finished	100.00%	8/8/2014			
Hand Animations	WIP	70.00%				
Hand Tuning	finished	100.00%				
Differentiate between path pieces the player is supposed to hit and isn't supposed to hit	Finished	100.00%	8/21/2014			This could be done by making a timer in each light path node after which it isn't safe or by making it so the player can (and will) collide with each node once, after which it isn't safe

Sound Assets	Status	Percent Done	Date Completed	Priority	Completion Percentage
Path generation	Implemented	100.00%	8/27/2014		89.47%
Wind sound from increased spee	DNE	0			
Banking	Implemented	100%	8/27/2014		
Crash into environment	Implemented	100%	8/27/2014		
Mine explosion	DNE	0			
Downer electric buzz	Implemented	100%	8/28/2014		
Afflicted by downer	Implemented	100%	8/27/2014		
Player character defeated	Implemented	100%	8/28/2014		
Powerup obtained	Implemented	100%	8/28/2014		
Ring expansion activation	Implemented	100%	9/2/2014		
Ring expansion end	Implemented	100%	9/27/2014		
Speed boost powerup activation	Implemented	100%	9/2/2014		
Menu selection	Implemented	100%	8/22/2014		
Menu cancel/back	Implemented	100%	8/22/2014		
Go through small ring	Implemented	100%	8/28/2014		
Go through medium ring	Implemented	100%	8/28/2014		
Go through big ring	Implemented	100%	8/28/2014		
Stage BGM 1	Implemented	100%	8/20/2014		
Menu BGM	Implemented	100%	8/27/2014		

Section	Percent Done	Total list Completion
Model Assets	100.00%	88.28%
Textures	88.89%	
Concept Art	100.00%	
Images	76.92%	
Animations	100.00%	
Sound Assets	100%	
Music	100.00%	
Menu Code	87.20%	
Level Code	83.33%	
Multiplayer Cod	45.00%	
Player Code	88.57%	
Sound Code	89.47%	