World Wizards: Developing a VR World Building Application

A Major Qualifying Project Report
Submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Bachelor of Science by:

Tyler Beaupre
Brian Keeley-DeBonis
Hope Wallace

Project Advisors:
Professor Jeffrey Kesselman
Professor Gillian Smith

Abstract

World Wizards is an open source and extendable world building environment that enables non-technical users to create 3D worlds in virtual reality and can be used for research, education, and product development purposes. It was developed for the HTC Vive using the Unity game engine. World Wizards embraces user-generated content, allowing users to build their own environments within VR and providing utilities to aid users in creating, distributing, and importing their own custom assets.

Acknowledgements

Our team would first and foremost like to thank our project advisors, Professors Jeffrey Kesselman and Gillian Smith for their support and guidance throughout this project. We would also like to thank WPI for giving us the resources and lab space for VR development. We also extend our thanks to Steve Finney, the artist who was contracted to create art assets for a professional level tileset. Lastly, we would like to thank the twenty WPI students who participated in our user testing and provided valuable feedback and data.

Table of Contents

List of Figures

List of Tables

1. Introduction

Neverwinter Nights (NWN) is a Role Playing Game (RPG) developed by BioWare in 2002 with gameplay mechanics based on Dungeons & Dragons. One of the things which made NWN so popular was the Aurora toolset, which allowed players to create their own content - including fresh storylines, tech demos, mini-games, new playable races and classes, and new enemies and gear. The toolset had a tile system for creating maps, placeable objects, and a built-in scripting language based on C for running cutscenes, quests, and conversations. By the end of 2002, there were already over 1,000 player-created modules available. The custom content community is still active today, and the Aurora toolset is still used by a number of colleges for educational purposes.

We believe that the educational and creative benefits of RPG toolkits such as the Aurora toolset for Neverwinter Nights are significant enough to merit further development in the area. The Aurora toolset fails to fulfill the expectations of software in the present day and the public deserves a worthy successor. Given the lack of modern world building tools, we set out to build a world building engine called World Wizards (WW). We decided to target virtual reality (VR) hardware because of its potential for an immersive world building experience and its support for intuitive motion controls and sensors that would be accessible to novice level users. VR is a modern technology which fits the needs of our project, allowing users to easily create worlds and providing them with a satisfying way of exploring their creations. We felt that VR would be an effective tool for modernizing this outdated area and a new way to bring its benefits to the public.

World Wizards is a multi-year project aiming to create an extendable game engine that enables users to create 3D worlds using the HTC Vive VR hardware. WW embraces user-generated content (UGC) and allows users to not only create their own levels and environments within VR, but supports users creating, distributing, and importing their own custom assets. The broad goal and vision of WW is to provide a flexible and adaptable world building environment that can be deployed for research, education, and product development. As the first MQP team on the World Wizards project, our goal was to create the core architecture. We developed the foundation of WW over A, B, and C term at Worcester Polytechnic Institute (WPI). Specifically, we developed an asset pipeline that supported UGC, and the core systems and tools for level building.

A common problem that users experience while using VR applications is nausea and dizziness. In an attempt to combat these issues in WW we conducted a usability evaluation with twenty WPI students. In addition to studying nausea in users, we were also able to determine how intuitive and accessible our VR controls were. We evaluated user experience through a series of tasks that included moving around, editing objects, and building structures in VR. We measured

user performance as they completed these tasks to determine the intuitiveness and accessibility of our application.

We asked users in a post survey to evaluate their experience in terms of nausea and dizziness, and if they felt that the current tools in W offered enough functionality to adequately create worlds. From this evaluation, we learned that for most users our application does not cause nausea or dizziness in most users and that after learning the controls, most users found them fairly intuitive.

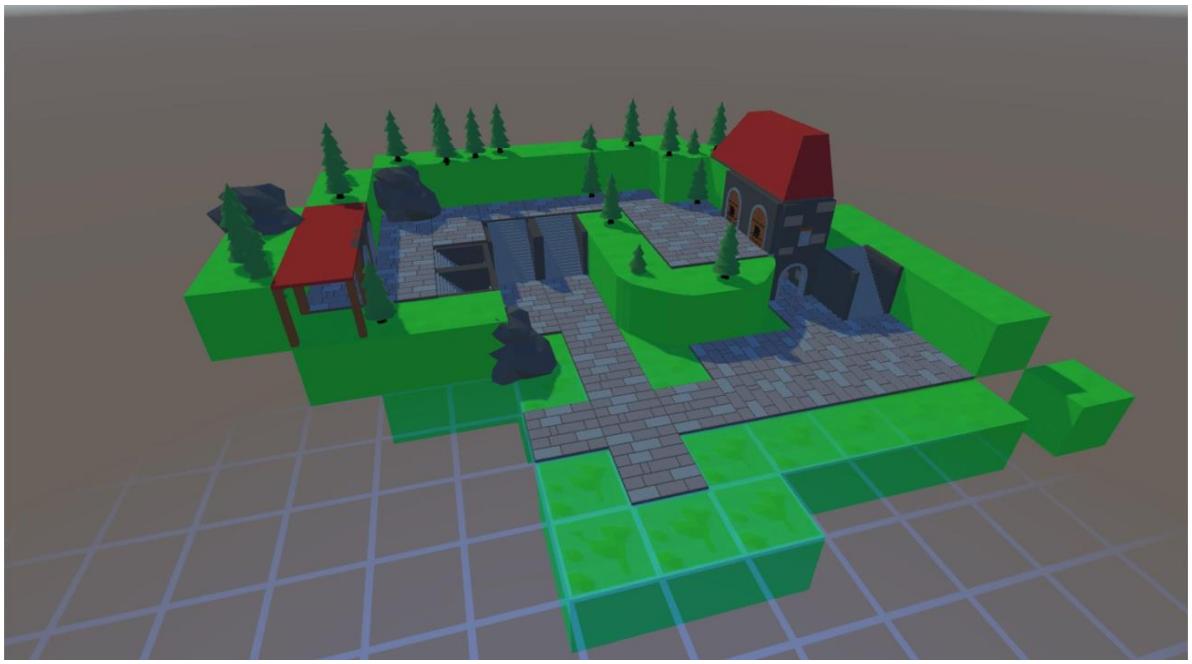


Figure 1: Screenshot of a small world created in WW using various tiles and props

WW lives in a public code repository on Github[1]. The repository contains both the source code and the resources needed to use WW in its host engine, Unity. WW is an open source project under the MIT license. Because of this, its development will not be limited by any current team, and it may be extended and used for a variety of VR or world building applications.

In this paper we will discuss the background research conducted in preparation for the project which impacted our overall design considerations, an in depth look at our experience design, and the technical design of the application. Then we will provide a description of our user testing, which found that there were some issues with controls, menus, and movement. Finally, we perform a post-mortem highlighting our successes and failures, and present a discussion of additional features such as interactable objects, NPCs, or a scripting system which may be added in the future.

[1] https://github.com/tbeaupre/worldWizardsCore

2. Background

WW is an open source UGC application for the HTC Vive that has the potential to fulfil many use cases ranging from educational to potentially commercial. We looked at three specific areas when conducting background research. First, we looked at the history of VR systems, controls, and user experience to learn how the HTC Vive came to be, how it works, and how to design the user experience for WW. Second, we looked at VR games and applications that shared similar features and goals as WW in order to get a sense of what kinds of applications already exist and how they were designed so we could compare to WW. Last, we looked at the history and effects of user generated content to learn about the games that allow UGC and what kinds of UGC those games allow.

2.1 History of Virtual Reality

VR is a type of experience meant to mimic real-life experiences, often achieved through the use of computers [38]. This modern definition of VR stems from a rich history of technical innovation that made VR what it is today. Knowing the history of VR and how VR works gave the WW team a lot of context when working with the HTC Vive.

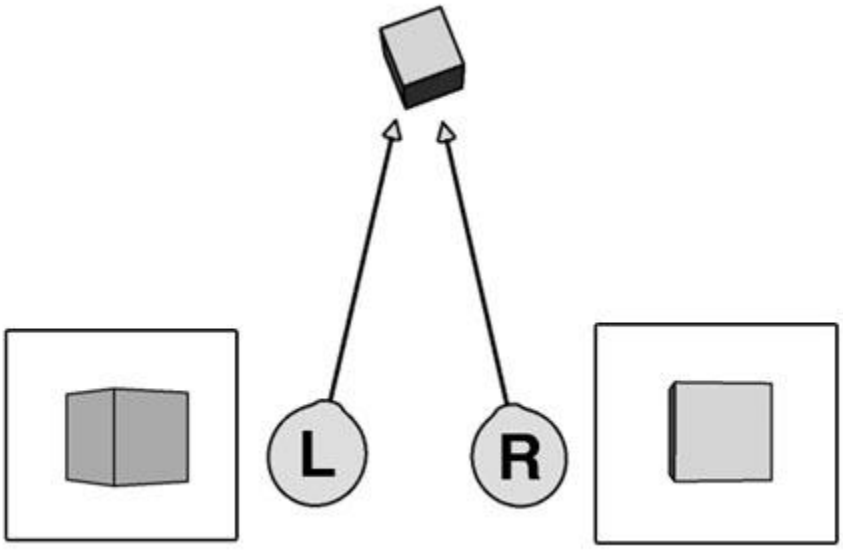


Figure 2: How the placement of the human eyes help the brain interpret depth[2]

Modern VR systems take advantage of the human ability to see stereoscopically. Stereoscopic vision is defined as the "brain's ability to put together two 2D images in such a way as to extrapolate depth" [26]. This ability is possible because our eyes are on the front of our faces and

[2] Northern Michigan University School of Art & Design. 2013. Retinal Disparity. (2013). Retrieved April 24, 2018 from http://art.nmu.edu/groups/cognates/wiki/1617e/Retinal_Disparity.html.

are separated. Each eye sees a slightly different two-dimensional image, which then gets interpreted by our brain as a three-dimensional image [26].

Using the fact that humans see stereoscopically, Sir Charles Wheatstone invented a device called the stereoscope in 1833. It used two mirrors to allow users to see a two-dimensional photograph as if it were a three-dimensional image. The popularity of the stereoscope grew tremendously when Queen Victoria became interested in it after using stereoscopes at the World's Fair in 1851 [32].

In 1929 the first known interactive virtual reality experience was invented - the "Link trainer". Named after its inventor, Edward Link, the "Link trainer" was an entirely electromechanical flight simulator. The device didn't have a screen, but it allowed users to practice controlling an airplane using a rudder and steering column. Though the "Link trainer" wasn't a product marketed to the public, it was successful in training over 500,000 pilots during World War II [36].

Using stereoscope technology, the View-Master was invented in 1938 by Harold Graves and William Gruber. The View-Master is a handheld toy stereoscope that can show full color pictures, as opposed to the original stereoscopes black and white images. It was sold to the public starting in 1940 and gained popularity once Disney themed images became available for it [28].

Figure 3: A View-Master, image from National Toy Hall of Fame [28]

In 1959, Morton Heilig attempted to create the first stereoscopic movie, which added functionality on top of the standard stereoscope. He invented the Sensorama, which is regarded as the first notable "simulated environment". Users could experience riding on a motorcycle through a city - they could "see the road, hear the engine, feel the vibration, and smell the motor's exhaust" [36].

Researching the early history of VR showed the WW team how modern VR works, where it came from, and how popular an idea VR is among the general public. The HTC Vive uses the fact that humans have stereoscopic vision in the design of its HMD - it has two screens that show slightly different images in order to show the player an illusion of being in a three-dimensional space. The View-Master showed us that VR has been a popular form of entertainment for a long time, which gives us hope that even if the current VR boom is temporary, that it will eventually move back into popularity.

Jaron Lanier, invented the term "virtual reality" while conducting research in 1987. He founded a virtual reality company called Virtual Programming Languages (VPL) Research, which invented and sold virtual reality products. They invented many VR devices that paved the way for future VR advancement. For example, the EyePhone was a head-mounted display (HMD) that displayed three-dimensional virtual environments using stereoscopic view. Another device invented by VPL Research was the Data-Glove, which was a glove that provided user input [37].

Prior to this point, most VR devices that involved more than simply looking at an image were either too expensive for the general public or used primarily for research. The Data-Glove was one of the first VR-esque devices modified to be sold as an entertainment product to the public, lighting the way for future VR devices to do the same. The Mattel Company evolved the Data-Glove into the Power Glove, an input device for games on the Nintendo Entertainment System [37]. The main difference between the Data-Glove and the Power Glove was their respective levels of accuracy. In order to make the Power Glove more affordable, it could only detect roll, as opposed to the Data-Glove which could detect yaw, pitch, and roll. Rolling Stone called the Power Glove a "legendary failure", as it only sold about 100,000 units during its lifetime. Despite its lack of popularity, the Power Glove is strikingly similar to modern entertainment VR devices [5].

Figure 4: The Power Glove[3]

[3] Evan Amos. 2018. File:NES-Power-Glove.jpg - Wikimedia Commons. Commons.wikimedia.org, (2018). Retrieved April 24, 2018 from https://commons.wikimedia.org/w/index.php?curid=16915852.

The VR industry boomed in the 1990's. Some of the devices introduced during this time period were the Nintendo Virtual Boy, Virtuality, and CAVE. While VR was widely popular at the time, the technology wasn't good enough or cheap enough to last. The VR industry crashed until the Oculus Rift was invented, and the modern era of VR began [25].

Learning about the swings in VR popularity made the WW team believe that while the VR industry has crashed (and may crash again), that VR technology today is far more advanced than it was even just 20 years ago and therefore will most likely succeed.

2.2 Modern Day Virtual Reality

There are many modern VR devices available for purchase today for various systems. The most popular VR devices available today include the Oculus Rift, Samsung Gear VR, PlayStation VR, Google Cardboard, and the HTC Vive.

The Oculus Rift was created by Palmer Luckey in 2012, with the intention of creating a head-mounted VR device that was more user-friendly than previous attempts. After coming up with a prototype, Luckey created a Kickstarter that, by the end of the campaign, raised US $2,437,429. The success of the Oculus Rift caused VR interest to skyrocket. Adding to the VR hype, Facebook bought the company that Luckey created for the Oculus Rift, called Oculus VR, in 2014 for $2 billion US dollars. The Oculus Rift started the VR movement that we are seeing today, and caused many other companies to develop their own VR devices [22]. The sudden interest of the general public in VR caused by the Oculus Rift made VR a viable choice for the WW team to develop for.

Figure 5: A modern Oculus Rift[4]

[4] Amazon. 2018. Retrieved April 24, 2018 from https://www.amazon.com/Oculus-Rift-Virtual-Reality-Headset-pc/dp/B00VF0IXEY.

Noticing the success of the Oculus Rift, Samsung hired the Oculus team to help them develop the Gear VR, a VR device set that allows users to experience content on their smartphones in VR [21]. Many other companies also followed suit: Sony released PlayStation VR, Google released Google Cardboard, and HTC released the Vive.

2.3 HTC Vive

The technology for the HTC Vive was developed by HTC in collaboration with Valve. In December of 2014, the first Vive developer kit came out and in 2015 the Vive was shown to the public at the Game Developers Conference. About a year later, it came out for sale to the general public [1]. The popularity of the HTC Vive was one of the reasons the WW team chose to develop for it - the fact that as of January 2nd, 2018 the HTC Vive accounted for 47% of the VR market makes our software accessible [6].

The HTC Vive has three main components: the head-mounted display (HMD), controllers, and base stations. The HMD contains two AMOLED displays with 1080 x 1200 pixel resolution each. It is connected to a computer via an HDMI port and a USB 2.0 port using the cables that run out of the top of the HMD. The refresh rate is 90 Hz and the field of view is 110 degrees. The controllers have several methods of input, including a trackpad, grip buttons, a trigger, a menu button, and a system button [10]. Both the HMD and the controllers' locations can be tracked in real world space using the base stations. Each base station is placed in opposite upper corners of the space where the Vive will be used. The specifications of the HTC Vive and the fact that it supports real world space movement contributed to the reasons why we chose to develop for it.

Figure 6: HTC Vive consumer edition [1]

2.4 Virtual Reality User Experience

User experience must be designed specifically for VR hardware and software. VR works in such a way that classic user input designs, such as heads-up displays (HUDs) and holding one controller with both hands, don't work well and therefore has inspired innovative controllers, control schemes, and user interface designs.

The main differences between VR controllers and traditional controllers are the HMD and motion controls. Standard VR HMDs are placed on the player's head and cover their eyes, immersing them in the game. This immersion creates the feeling of being in the game world, but completely blocks the player's view of the real world including any other controllers the player may be using and the player's entire body [20]. Traditional video game displays don't block the player's view of the real world at all, making the HMD an interesting challenge for developers. While there are non-VR controllers that use motion controls, such as Nintendo Wii's Wiimote and Microsoft Xbox's Kinect, the added factor of the player's view being obstructed by the HMD causes some interesting challenges. Motion controls are an important aspect of VR because most VR devices can track the location of both the HMD and handheld controller(s), allowing for the player to move around in real-world space to interact with the application they're using.

Figure 7: Brian, one of the WW team members, wearing an HTC Vive HMD

Steam VR, a Steam plugin that provides a useful software library for use with the HTC Vive, provides a good example of how VR software has adapted to VR controllers. Its main purpose is to allow people to play VR games they have in their Steam library with an HTC Vive, but also includes some side features useful for combating VR challenges. One such feature is the ability to see the controllers through the HMD display. While the player is in-application, the controllers

appear in the game world and move in the game world as they are moved in the real world. This characteristic of Steam VR combats the issue of not being able to see the controller while wearing an HMD and gives the player more control over their actions.

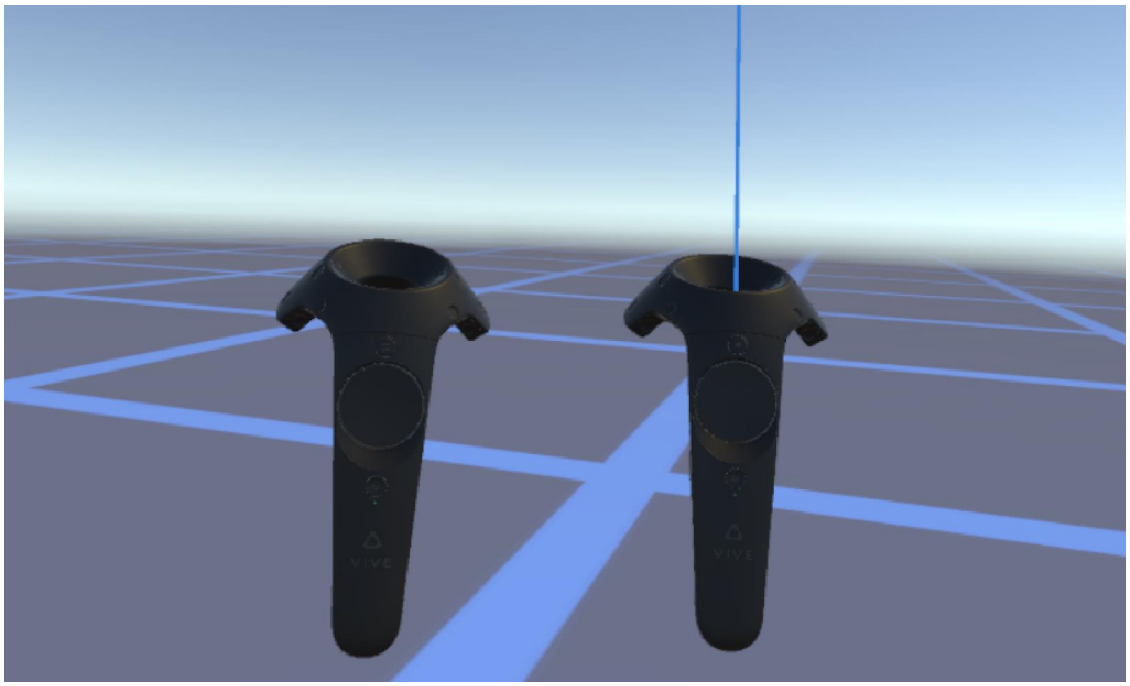


Figure 8: The HTC Vive controllers in WW as seen through the HMD

Another innovation of VR software is an optional grid that shows you the edge of the real world space, provided by HTC Vive software. Room-scale VR applications require the use of real world space, so the player can walk around while being tracked by the HMD. This trait requires the dedication of open, unobstructed real world space that may be surrounded by walls or other objects. When Steam VR is set up with the HTC Vive, the person setting up must tell it where the edges of the room are. Using this information, Vive software then shows a fence-like structure in-application that informs the player where the boundaries of the real world space are in order to prevent injury. WW takes advantage of this and the previously mentioned feature of VR software in order to provide the most comfortable user experience possible.

VR also poses some interesting challenges in regards to user interface (UI) design. Many of the types of menus we see in non-VR applications are not directly applicable to VR applications. For example, heads-up displays (HUDs) don't usually work in VR. Because of their proximity to the "front" of the screen, and HMD screens being very close to user's eyes, HUDs in VR tend to be blurry [34]. Therefore, almost every UI in VR applications are within the application environment. The WW team used this information when designing our menus. For example, instead of using a HUD to change tools and how what tool was currently being used, we created a menu that exists within the game world space and follows the player around based on their controller location. More information about this menu can be found in Section 5.3.1.3.

Within the realm of menus that exist within the game world, there are some menus that work better than others. For example, a comparative study conducted at University Carlos III de Madrid in Spain showed that within VR applications the time it takes for users to complete tasks is statistically less using radial menus than linear menus [2]. The WW team believes that radial menus would be a useful addition to WW in the future for a menu that aids players in choosing the objects they want to place.

2.5 Similar Titles

With approximately 190 VR titles currently available on the Steam Store (a popular marketplace for computer games), VR games and applications are increasing in availability [27]. It is important to discuss similarities between games in this market with the World Wizards project, as well as highlight the novel aspects of World Wizards.

Of the 190 VR titles, there are four titles that merit discussion due to their similarities and varying degrees of commercial success. Our criteria for selecting these four titles was to look briefly at all of the available VR titles on the Steam Store and then focus on the titles that stood out as being similar in terms of planned features or goals to WW. We also included Minecraft, which is not on the Steam Store, due to our prior awareness and experience with it. We considered commercial success as a category when conducting this background research as we intuitively felt that there is a correlation between well implemented features and an active user base of players, and commercial success.

Minecraft is a sandbox video game with an active modding community that creates user generated content. Minecraft is game that allows players to explore and create almost anything they desire within a procedurally generated world of cubes. Minecraft is also optionally multiplayer, and allows players to collaborate and build worlds together. The game also features a crafting system, allowing players to craft different materials. While Minecraft has been out since November 2011, a VR version of Minecraft was officially released by Microsoft in 2016 [19].

Minecraft has also been used as an educational tool in the classroom to teach STEAM topics. The educational version of the game currently features electrical circuit blocks and chemistry building blocks that allow players to create basic programs and learn about the periodic table and atomic structure. Minecraft also has an accompanying computer science introductory course. It is important to note that Minecraft is not open source [18]. An application that is not source is limited in that people cannot contribute directly to the development of the project, nor can they make their own spinoff projects using the original open source project as a foundation. In terms of commercial success, this clearly has not limited or impacted Minecraft in any way.

With more than 144,000,000 copies of Minecraft sold and a reported 72 million active players in December of 2017 (this includes mobile sales and players), Minecraft is one of the most

successful games to date [23]. With its overwhelming commercial success, Minecraft demonstrates the huge demand and lifespan for games that focus on creative building, multiplayer, and sandbox gameplay. In contrast, WW is a non-commercial project that currently has no active users beyond its development team. Minecraft is also limited to cube style graphics. WW has the capability for much better visuals as tiles are not limited to cubes, but can be any 3D art asset creation.

Figure 9: Minecraft screenshot of an underwater world that is entirely editable by the player [19]

Moxbox is an early access commercial VR game that focuses on user creation, modding and multiplayer developed by Alientrap [17]. Its goal is to enable players to create and share their own experiences in VR. It has been in early access since August of 2016.

In many ways, Modbox is further along than World Wizards in terms of raw features and functionality. It allows players to construct custom assets within the game from selection of primitive objects. It features a richer material system with physical properties like durability, bounciness, and friction. It also has more polished and powerful VR tools for creation and editing. Modbox also supports a basic wiring system that allows the player to create interactable contraptions by linking various exposed properties of entities together.

Modbox also has an SDK for Unity to allow users to create mods for Modbox. The entirety of Modbox runs in the SDK. It allows custom C# scripting and the mods can be output as Unity Asset Bundles and shared. The custom scripts can be compiled into DLLs which are uploaded along with the mod. Mods can include not only user created artwork and environments but also user created gameplay mechanisms. It is important to note that Modbox is not open source which means its development is limited to the speed and vision of the company's team.

Figure 10: Modbox promo showing a user created environment and gameplay experience [17]

Tilt Brush is a commercial 3D painting application for VR developed by Google and first released in August of 2016 [29]. It features an extremely polished user interface and set of brush tools. Its singular focus is to allow users to create and output works of art using the application. The 3D brush artwork that a user creates can be exported and shared in VR and as animated GIFs for the web. The relation of Tilt Brush to WW is that Tilt Brush is an example of a successful VR application that allows novice level users to create original works of art with ease. The controls are simple and intuitive to use and allow for users of all ages to use the application without having to worry about a steep learning curve.

Figure 11: Tilt Brush screenshot showing a 3D painting that a user created [29]

Tiny Town VR is a commercial VR game that allows players to create towns and cities in VR. It is developed by Lumbernauts and was first released in August of 2017 [30]. Players can decorate their towns with different townspeople and characters and give them different speech bubbles.

Players can then take screenshots of their creations and scenes and share them with others. As of March 2018, Tiny Town VR has 43 reviews on the Steam Store. Tiny Town VR is not comparable to the above titles in terms of commercial success [30]. However, because its purpose as an application is to be a creative platform in the VR space, it is noteworthy.

Figure 12: Tiny Town screenshot showing a scene that user created [30]

Table 1: Comparison of Features between World Wizards and Selected VR Titles

	Minecraft	**Modbox**	**Tilt Brush**	**Tiny Town**	**World Wizards**
Supports UGC	YES	YES	YES	YES	YES
Open Source	NO	NO	NO	NO	YES
VR Support	YES	YES	YES	YES	YES
Desktop Support	YES	YES	YES	NO	YES

While the above titles have, to varying degrees, influenced our development of World Wizards, there are several differentiators that separate our project. Considering the long term vision and goals of World Wizards, the project is currently still in an unpolished and early development stage. These games, from which we have drawn inspiration, are much further along in their development and are far more polished and feature complete. The aforementioned World Wizards problem statement and objective of providing an intuitive and adaptable world building platform for VR that can be deployed in research, product development, and education is also unique. Finally, World Wizards has noncommercial roots and is open source. The repository for

World Wizards is publicly available on Github and will allow contributions and forking from members of the community. The license World Wizards is distributed under is MIT License. None of the titles above allow players or community members to directly contribute code to the development, nor can they incorporate or extend the codebase.

2.6 User Generated Content

User-Generated Content (UGC) in the context of video games is any content (including storylines, art assets, or gameplay elements) created by the end-user that may be shared and played on the system with which it was created. UGC is seen in numerous fields, but our research covering it has pertained primarily to its use in video games, as this is most relevant to the World Wizards project. One of the earliest games to heavily focus on UGC was the Adventure Construction Set, a program released by Electronic Arts in 1984 for the Commodore 64. The Adventure Construction Set allowed players to create tile based adventure games. While the program would come with a sample game included, the primary role of the program was for users to create their own games. The Adventure Construction Set Club (mentioned on the game's manual) gave users access to a library of games created by other users [7]. This kind of a centralized community was rather unique in the world of shareware, but has since become the standard for these types of UGC centered games.

Figure 13: Screenshot of room created in Adventure Construction Set [7]

Another subsection of UGC in video games is modding. Mods (short for "modification") are alterations to a game which can be graphical or gameplay related. Mods can take numerous different forms, from purely aesthetic graphical overhauls, to completely new games made using the framework of the original. While mods exist in an unsettled legal grey zone, some companies such as id Software, Valve Corporation, Blizzard Studios, and Bethesda Softworks actively support their respective modding communities.

One of the first large modding communities was for id Software's *Doom* (1993). *Doom*'s WAD (short for "Where's All the Data?" [31]) system for storing data allowed for easy modification of maps, sounds, characters, and weapons. *Doom* is considered by many to be the grandfather of all first person shooters, and its mod-making community was one of the keys to its massive popularity.

In a few instances, mods have been so popular that they were ported into their own independent games [11]. *Counter-Strike* was originally a multiplayer mod for Valve's *Half-Life*. The intellectual rights to *Counter-Strike* were acquired by Valve, who released an independent remake of the mod called *Counter-Strike: Source* in 2004. The sequel to *Counter-Strike: Source*, *Counter-Strike: Global Offensive* (2012) is one of the most popular eSports today [9].

Perhaps the most famous example of a mod becoming an independent game is Defense of the Ancients, more easily recognized by its acronym, DotA [15]. Currently, the most popular incarnation of DotA is Valve's Dota 2 [16]. What started as a custom game mod for Blizzard's *Warcraft III: Reign of Chaos* (2003), has become an international phenomenon, spawning the entire Multiplayer Online Battle Arena (MOBA) genre and becoming the largest eSport in the world [39].

Figure 14: Screenshot from Dota 2 [15]

DotA was originally inspired by a mod for a separate game, Aeon of Strife, a custom game mod for Blizzard's *Starcraft* (1998). DotA was created in 2002 by Kyle Sommer, under the alias Eul. Throughout its life, development of DotA has passed between many different hands, including Guinsoo (who would later become a game designer at Riot Games, the creators of *League of Legends*) and IceFrog (who was hired in 2009 by Valve to create the modernized sequel, Dota 2, as a stand-alone game) [39][12].

Figure 15: Screenshot from DotA Allstars [12]

If these toolsets and modding communities had not existed, many of the most popular games of the modern era may never have been created. We believe it is important for these tools to be available to users giving them the ability to create the next generation of video games. World Wizards is free, open source, and specifically designed with non-technical users in mind, making it as accessible as possible.

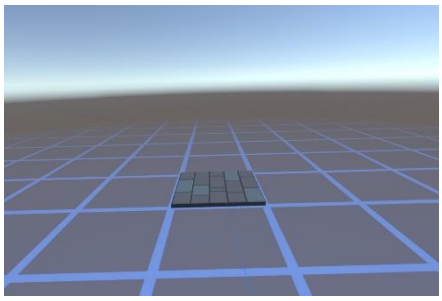
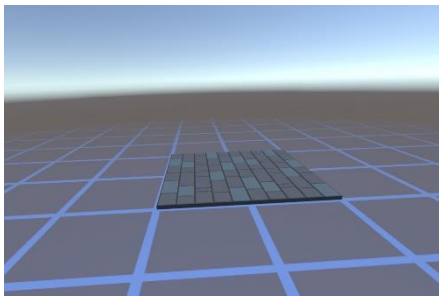
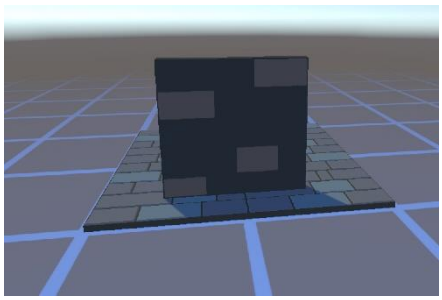
3. Experience Design

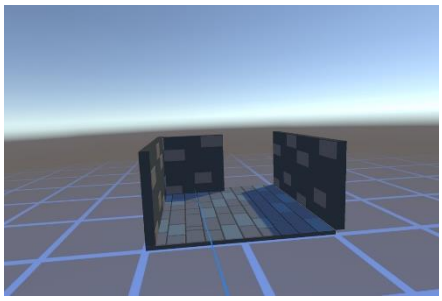
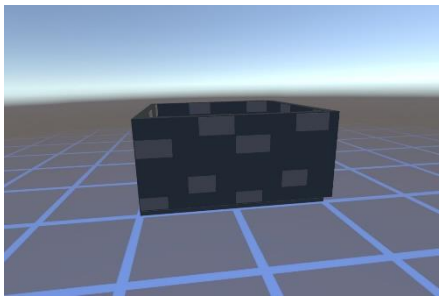
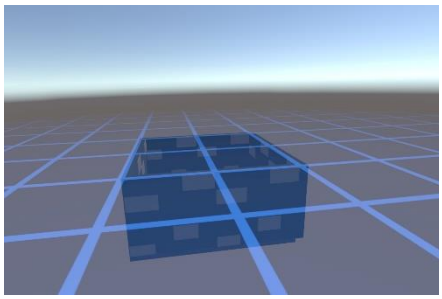
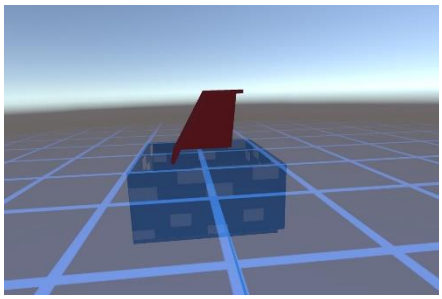
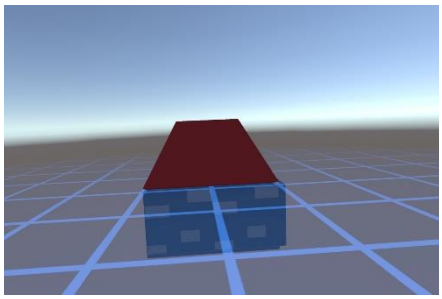
This chapter will first explain how WW is played in its final state, then examine the process that went into building WW, the design decisions we made along the way, and how we made those decisions.

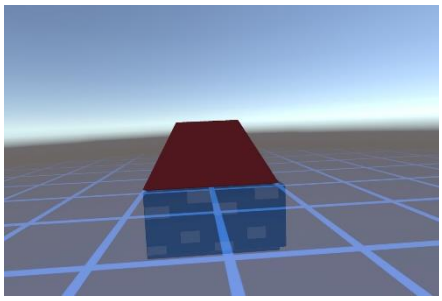
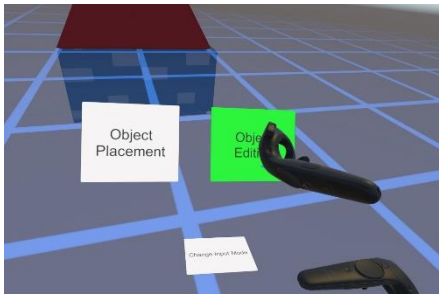
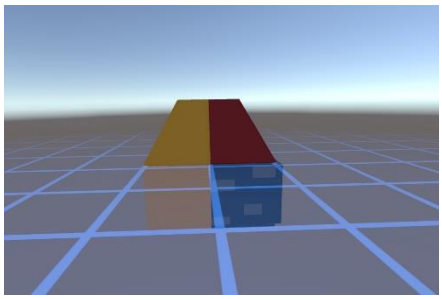
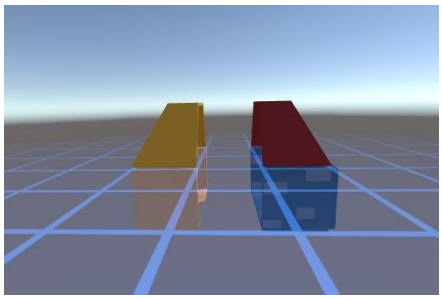
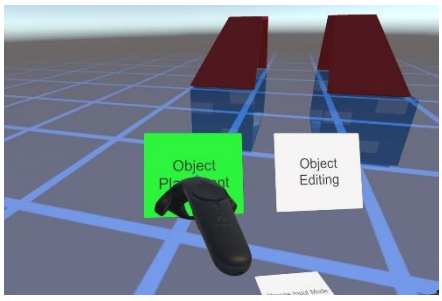
3.1 Walkthrough of Editing Scenario

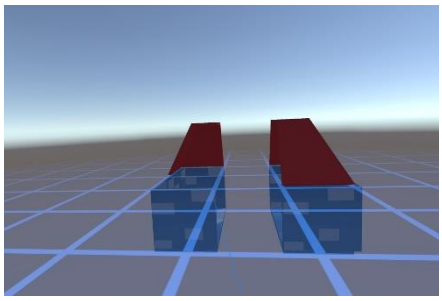
The following actions are reproducible and result in a user creating a small house in WW and then editing the house.

Table 2: A walkthrough of using WW to create and edit a house

Action	Description of Action	Picture of Action
Cycle objects	Using the `CreateObjectTool`, user cycles through objects by touching the top portion of the touch button on the right Vive controller and locates a floor tile.	
Place floor tiles	Using the `CreateObjectTool`, user places 4 floor tiles by pressing the trigger of the right Vive controller 4 times, placing the floor tiles into a 2x2 floor.	
Cycle objects	Using the `CreateObjectTool`, user cycles through objects by touching the top portion of the touch button on the right Vive controller and locates a wall tile.	

Rotate wall tile	Using the `CreateObjectTool`, user rotates the wall tile by increments of 90 degrees by touching the left and right sides touch button on the right Vive controller.	
Place wall tiles	Using the `CreateObjectTool`, user places 8 wall tiles by pressing the trigger of the right Vive controller 8 times, placing the wall tiles along the perimeter of the 2x2 floor. The user first rotates the wall tiles into the proper orientation in the prior action.	
Move construction grid	Using the `CreateObjectTool`, user moves the construction grid up 1 level by pressing the top part of the touch button on the right Vive controller.	
Cycle objects	Using the `CreateObjectTool`, user cycles through objects by touching the top portion of the touch button on the right Vive controller and locates a corner roof tile.	
Rotate roof tile	Using the `CreateObjectTool`, user rotates the corner roof tile by increments of 90 degrees by touching the left and right sides touch button on the right Vive controller.	

Place corner roof tiles	Using the `CreateObjectTool`, user places 4 corner roof tiles by pressing the trigger of the right Vive controller 4 times, placing the corner roof tiles, enclosing the structure. The user first rotates the corner roof tiles into the proper orientation in the prior action.	
Switch to `EditObjectTool`	Using the menu attached to the left Vive controller, the user moves the right Vive controller into the button labeled “Object Editing” and switches the tool in their right hand from the `CreateObjectTool` to the `EditObjectTool`.	
Select half of the house structure	Using the `EditObjectTool`, user holds down the grip on the right Vive controller and draws a marquee box, selecting the left half of the house structure.	
Move selection	Using the `EditObjectTool`, user holds down the trigger on the right Vive controller and points the controller at a space on the construction grid 1 unit to the left. The tiles selected move to the new position.	
Switch to `CreateObjectTool`	Using the menu attached to the left Vive controller, the user moves the right Vive controller into the button labeled “Object Placement” and switches the tool in their right hand back to the `CreateObjectTool`.	

Delete part of roof	Using the `CreateObjectTool`, user points the right Vive controller at 1 of the corner roof tiles on left structure and presses the grip on the right controller. The corner roof tile is deleted.	

3.2 Final Design

The final iteration of WW involved implementing the input system and writing the VR control scheme. The input system was designed around the idea of being able to swap control schemes at runtime, because we realized that the number of buttons on the controller is too few to support every function at once. Figure 16 shows the HTC Vive controllers and explains the buttons on them.

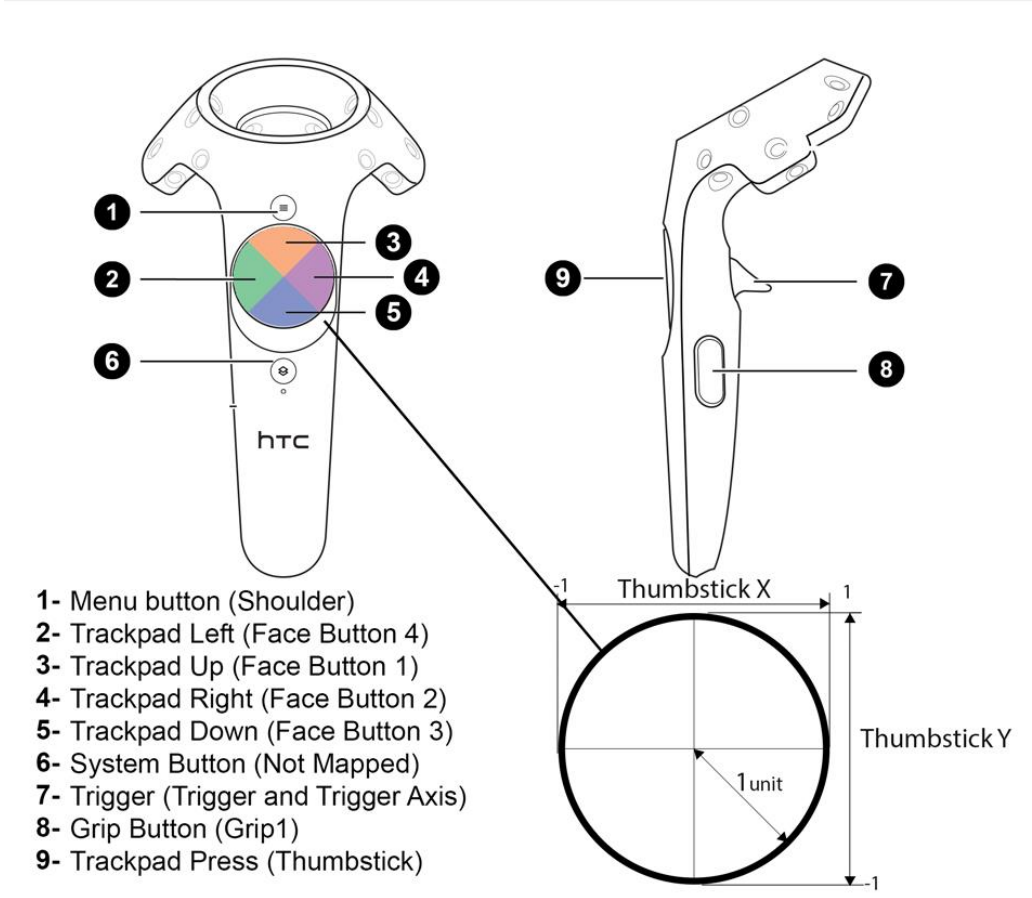


Figure 16: The HTC Vive controller button setup [23]

The current VR control scheme is split up into separate `Tools`, each of which has its own specific function. The `StandardTool`[5] handles player movement, the `ObjectPlacementTool` allows players to choose and place objects in the world, the `ObjectEditTool` lets users edit objects previously placed in the world, and the `MenuTraversalTool` allows users to interact with menus.

The `StandardTool` is permanently on the left controller, because we think players will generally be moving a lot. The `ObjectPlacementTool` or the `ObjectEditTool` can be on the right controller, and the player is able to swap them out at will with the arm menu. The `MenuTraversalTool` replaces the `Tool` currently on the right controller when the player opens a menu. The controls for each of those `Tools`, including the number of the button they correspond with in Figure 16 are as follows:

Table 3: `StandardTool` control scheme

Button	Function
Trackpad (2-5)	Touch (2-5) - Horizontal movement: forward, backward, left, right
	Press (3, 5) - Vertical movement: up, down
Trigger (7)	Hold down - Show laser pointer
	Release - Teleport to the location you were pointing at

Table 4: `ObjectPlacementTool` control scheme

Button	Function
Menu Button (1)	Open filtering and loading menu, swap `Tool` to `MenuTraversalTool`
Grip (8)	Delete object the controller is pointing at
Trackpad (2-5)	Touch (3, 5) - Cycle through objects being placed
	Press (3, 5) - Move construction grid up and down
	Press (2, 4) - Rotate preview object
Trigger (7)	Hold - Show preview object where controller is pointing
	Release - Place object where controller is pointing

[5] This `Tool` is called the `StandardTool` (instead of `MovementTool`) because it was planned to have more functionality, but due to scoping issues ended up only controlling movement.

Table 5: `ObjectEditTool` control scheme

Button	Function
Menu Button (1)	Open filtering and loading menu, swap `Tool` to `MenuTraversalTool`
Grip (8)	Click - Select single object the controller is pointing at
	Hold - Move controller diagonally to marquee select multiple objects
	Release - Confirm selection
Trackpad (2-5)	Press (3, 5) - Move selection up/down one cell
	Press (2, 4) - Rotate selection
Trigger (7)	Hold - Move current selection
	Release - Place current selection

Table 6: `MenuTraversalTool` control scheme

Button	Function
Menu Button (1)	Close menu, swap to previous `Tool`
Trigger (7)	Click - Press the button the controller is pointing at

The VR control scheme was designed to allow users to quickly and easily build worlds. Each `Tool` was designed to both fit the controller's buttons and give the users as much flexibility as possible.

The `StandardTool` is always on the left controller because movement is an often-used component of building a world. We mapped horizontal movement to touching the trackpad because we thought it would be used more often than vertical movement. Also, we found that touching the trackpad accidentally is very easy to do, so we didn't map vertical movement to trackpad touch to prevent users from feeling nauseous. Vertical movement is mapped to pressing the trackpad either up or down. This function proved to be a mild issue, however, due to the trackpads registering touch before press, causing users to move forward and up at the same time. This issue is further discussed in section 5.3.1.1. The teleport function was mapped to the trigger because we thought it was the most comfortable and simple button to press on the controller, and goes along well with pointing the controller, which is a necessary component of teleporting.

The `ObjectPlacementTool` is one of two `Tools` the user can put on the right controller. The application starts with this Tool already on the right controller, as opposed to the `ObjectEditTool`, because placing objects is the precursor to editing objects. Cycling through objects is done by touching the trackpad because we thought that users would be cycling often, so we wanted it to be as easy as possible to do. Placing new objects is done with the trigger, because we felt that using the trigger is the most comfortable when you have to point with the controller. Trackpad presses move the construction grid and rotate preview objects because the grid is being moved up and down and rotation is done to the left and right, so having directional buttons associated to these functions makes sense. The delete function was mapped to the grip button, mainly because deleting is an important function and there weren't any other buttons left.

The other `Tool` that can be put on the right controller by the player is the `ObjectEditTool`. Trackpad presses were chosen to handle rotating selections and moving selections vertically because these are similar functions to trackpad presses for the `ObjectPlacementTool`. Moving and placing selections is done with the trigger because it involves pointing and also mirrors the trigger function for the `ObjectPlacementTool`. Selection is done with the grip button, mainly because there weren't any other buttons to use and the WW team knew it was important to have selection as a function in this `Tool`.

The final `Tool` present in WW is the `MenuTraversalTool`. The mapping for this `Tool` is simple - the trigger is used to click whatever button the user is pointing the controller at. The main way to improve this `Tool` would be to get rid of it entirely by checking a raycast from the controller to see if it hits a menu first, and then switching the controls accordingly.

Overall, when designing the VR control scheme, the WW team had the most difficulty with the limited number of buttons on the HTC Vive controllers. We found that the grip button was awkward to use so we were reluctant to use it. Unfortunately we were forced to map it in order to include all of the functionality we deemed necessary for world building. Future iterations of WW should focus on testing new control schemes and figuring out novel ways handle user input.

The input system also supports desktop controls. We implemented these as a way to test functionality without using VR, but realized towards the end of the project that desktop controls would allow many more people to be able to use our application. Currently, the desktop controls are extremely unintuitive, as we did not plan for anyone besides the WW team to use them. Future iterations of WW should remap the desktop controls to make the application more accessible.

3.3 Original Design

Before the WW team started building WW, we created a design document explaining how we envisioned the application as a whole by the end of the project. This section will discuss our ideas at the time about the overall goals of the project, the control scheme, user interfaces,

camera modes, and code architecture. Then, for each of these we will describe what holds true in the final project, what was cut, and why those things were cut. The entire design document can be found in Appendix A.

3.3.1 Goals

Our overall goals were the following:

1. Intuitive tile and prop placement, leveraging the power of VR.
2. Multi-level support with Portals to traverse them.
3. Fully fledged and sophisticated, yet simple, level editor.
4. Import and setup custom tiles and props using Unity Editor extension and then export as asset bundles to be used in the World Wizards Application.
5. Place interactable objects in the world such as doors, levers, traps.
6. Gesture-based object transformations and controls.
7. Group objects together for multi-object editing capabilities.
8. Portable file system to save levels and user setting. Use a common format like JSON.

The ambiguity of some of these goals makes it difficult to evaluate whether we met them or not, but we believe we met most of these goals by the end of the project. Based on our user testing, described further in Section 5, WW has relatively intuitive tile and prop placement mechanisms. WW also supports importing and setup of custom tiles and props, multi-object editing, and portable save files using JSON.

However, WW does not include some of the features described in our goals. We did not implement portals to travel between levels, or interactable objects. Also, the WW controls are largely button presses as opposed to the gesture-based controls we had planned on. Most of these goals were not met due to time constraints. We believe that all of these features should be included in future iterations of WW.

3.3.2 Control Scheme

The WW team envisioned the control scheme to consist of mostly gesture-based controls, such as head movement and controller movement. We wanted the menus in the application to be scrolled through by swiping the controller and selections to be made by users looking long enough at the selection they want to make. The controls for selecting and rotating objects were going to be specific gestures with the controllers. The team also pictured a virtual workbench that followed users around, allowing them to physically pick up the tools they want to use (such as a selection tool, or rotation tool).

Due mainly to time constraints, the current control scheme consists almost entirely of button presses on the controllers, as shown in the tables in the previous section. Only the arm menu, which allows users to change the control scheme on the right controller by physically touching the buttons with the virtual controller, has controls that do not involve pressing buttons on the controllers.

The workbench idea never came to fruition either, though the team used the idea of having "tools" in the creation of our user input system. Each controller has a `Tool` attached to it, an object type which defines the control scheme for the controller it's attached to. `Tools` can be swapped out at runtime using a menu, allowing the user to essentially "pick up" different tools as they build despite not having an actual workbench. Figure 17 shows the menu used for swapping `Tools` in WW.

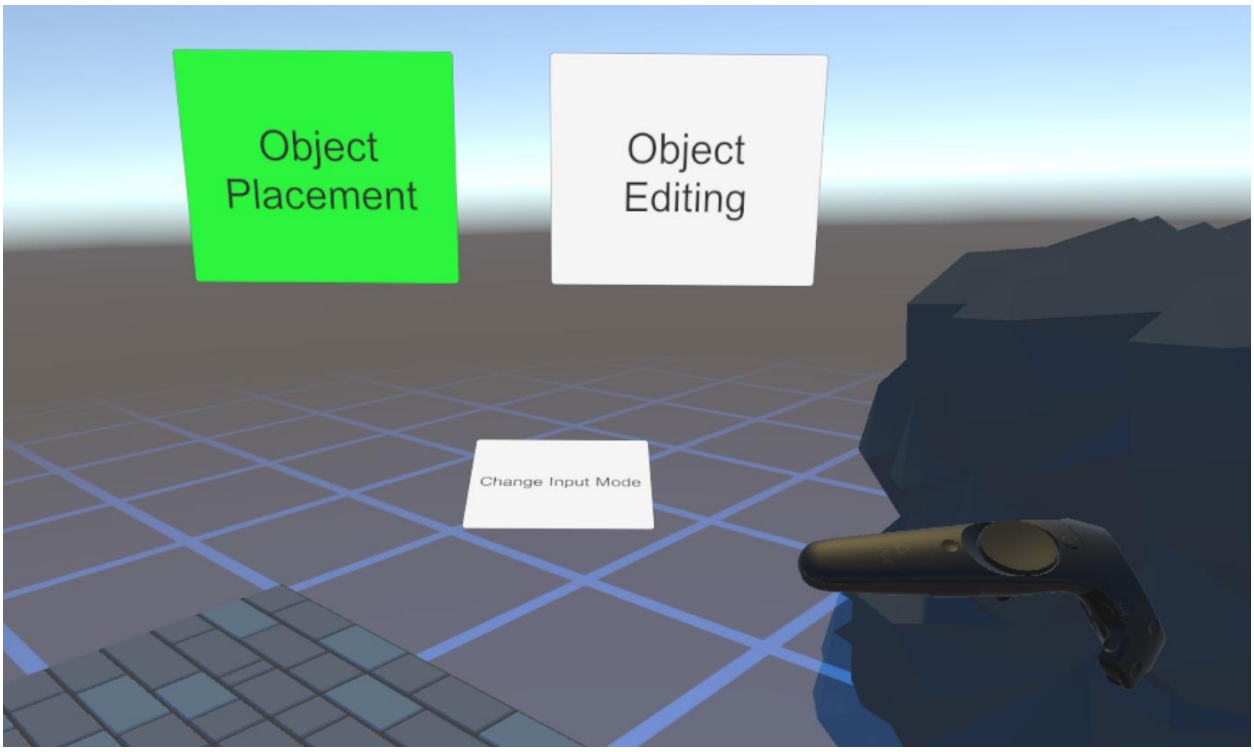


Figure 17: The arm menu, used to change the current `Tool` on the right controller

3.3.3 User Interfaces

Originally, the WW team thought that we would create many menus placed throughout the application. We wanted a main menu, where users could select which level they wanted to load and change profiles that was controlled using swiping motions and gaze to select. We also wanted to incorporate radial menus in some way, where users could gesture with their controller to make their selection.

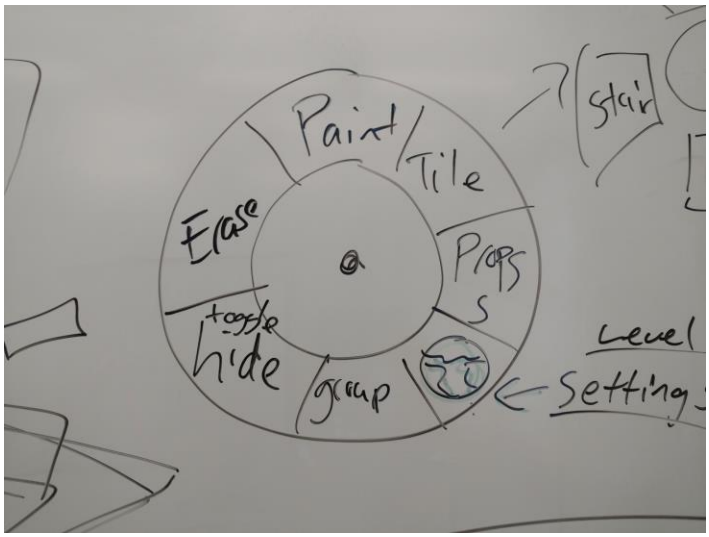


Figure 18: An early mockup of a radial menu idea

These gesture-based menus did not make it into the final product. Only two menus are in the final project: one that allows players to filter objects, load from a save file, and delete everything in the scene, and another that allows the player to change the `Tool` that is on the right controller. The first of those menus appears with the press of a button and players make selections by pointing and clicking the trigger on the right controller. The other menu, the arm menu, is always visible, and is controlled by using the right controller to physically press the menu options. The arm menu is described in further detail in section 5.3.1.3.

These two menus provided most of the functionality we were striving for without being too time consuming to implement. The filtering and loading menu allowed us to test some functionality, and the arm menu allows users to quickly switch between `Tools`. We believe that future iterations of this project should include the original ideas the WW team had for menus, especially the radial menu idea as a way to choose objects the user wants to place.

3.3.4 Camera Modes

Plans for two different camera modes in WW were part of our design document. The first mode was a workbench perspective, which showed the player a scaled down version of their world on a table in front of them that the player would be able to walk around in order to edit the world. This mode would allow players to quickly change their world and see all of it at the same time. The second mode was a first-person player perspective, which allowed players to see what their world looks like with a better sense of scale and proportion than the previous camera mode.

There is only one camera mode in the final version of WW which allows players to move freely enough to see the world in first-person or birds-eye view and edit the world. We believe that the final version of the camera is better than what we had originally planned because it allows players to build from either perspective and not have to manually swap between modes. Some

experimentation might be helpful for future iterations of WW to determine if more than one camera mode would be useful to players, as we did not actually test having multiple camera modes.

3.3.5 Code Architecture

In the design document, the WW team laid out what we thought would be the basic code architecture for WW. It consisted of having three namespaces in order to separate data types, controllers, and user settings.

Largely the code architecture remained the same, with some further separation of code into sub-namespaces. We did not implement user settings due to time constraints.

3.4 Overall

WW was designed to be an open source application to be used for teaching and for building an online community where anyone can add functionality or art assets to the project. We designed the entire application to be easy to set up and use in order to promote this. The control scheme was meant to be simple and intuitive, yet provide all of the necessary tools to build worlds. We expect that future work on the project will try to keep functionality as simple to use as possible in order for WW to be accessible to as many people as possible.

WW is also a great teaching tool for VR because it is open source - anyone can look at all of the code that this project is comprised of and modify it in any way they want. WW is also made to promote UGC through its art asset pipeline, as explained in section 4.5. Both of these factors will hopefully help build an online community out of the people working on WW code and art.

Currently, WW only supports static object placement as a means of building virtual worlds, but eventually the project is meant to be a tool used to create games. The next iterations of the project, as described in Section 6.5, will most likely support scripting, artificial intelligence, interactable object, and more. It is planned to eventually be made into a fully functional game engine where people can create their own interactive experiences.

4. Technical Design

4.1 Hardware

When creating a virtual reality application, an important consideration to make is what hardware the application will target. We developed our project for the HTC Vive, a virtual reality headset developed by HTC and the Valve Corporation [35]. The headset interacts with two 'Lighthouse' base stations to create a 15x15 foot radius area in which the player's movements are tracked. Our application utilizes two controllers, each with a number of buttons, a trigger, a touchpad, and motion tracking. The Lighthouses send out infrared light pulses which are detected by sensors in the headset and controllers, allowing the player's movements to be tracked with sub-millimeter precision [4]. Though our application does not currently include any sound, the Vive has support for headphones which may be used to bolster the player's experience with 360 degree sound.

Figure 19: HTC Vive Virtual Reality Headset

Currently, the two primary options for VR headsets are the Oculus Rift and the HTC Vive. The Oculus offers comparable power [3], and similar Unity support, but fails to offer the safety features of the Vive (such as the Chaperone system or the front facing camera), or the same "room scale" tracking. This "room scale" tracking is especially helpful for when a player wants to get a better look at small details that might otherwise be difficult to view. Rather than attempting to make fine maneuvers with the controllers, they can simply walk up for a closer look. We wanted to give the player complete motion around their world and to keep them safe throughout, so we chose to design for the HTC Vive.

4.2 Software

For our project, we needed an Integrated Development Environment (IDE), a game engine, and some form of source control. In this section, we will discuss the tools which we used to develop our application and how we decided on them.

We chose to build our application in Unity, a cross-platform game engine created by Unity Technologies. Unity offers support for 2D and 3D games on desktop, mobile, and consoles [13]. In addition, the Steam VR plugin provides support for VR applications. We will discuss this plugin in a later section, but its integration with Unity was a consideration we made when choosing a game engine. Since our project is purely academic and open-source, we operate under the personal Unity license and are able to use it for free [33]. This will be helpful for the long-term life of the project by removing the complications of acquiring funding or paying out of pocket.

Figure 20: Screenshot of Unity game engine

While this is purely preferential, Unity's scripts are written in C#, a language which all of us have experience with. This allows us to use IDEs we are familiar with and in a language we are familiar with, helping to increase our productivity and the cleanliness of our code. Unity is comparable to any other engine on the market, but it's free use, Steam VR plugin, and scripting in C# have made it an ideal option for this project.

For an IDE we chose to use Rider, a relatively new, cross-platform .NET IDE created by JetBrains. Rider is based upon IntelliJ, a popular Java IDE, but also includes all of the features of ReSharper (a Visual Studio extension for .NET developers created by JetBrains) built in. One of the reasons behind this decision was that one of our members was a Mac user. This meant that whichever IDE we decided upon needed to be cross-platform. Additionally, every member was

familiar with JetBrains' other IDEs, so it was comfortable for all of us and we didn't need to waste any time learning a new IDE. The final factor in our decision was that Rider was developed with Unity in mind. It gathers information from Unity projects and uses it to display a Unity symbol next to Unity-specific code.

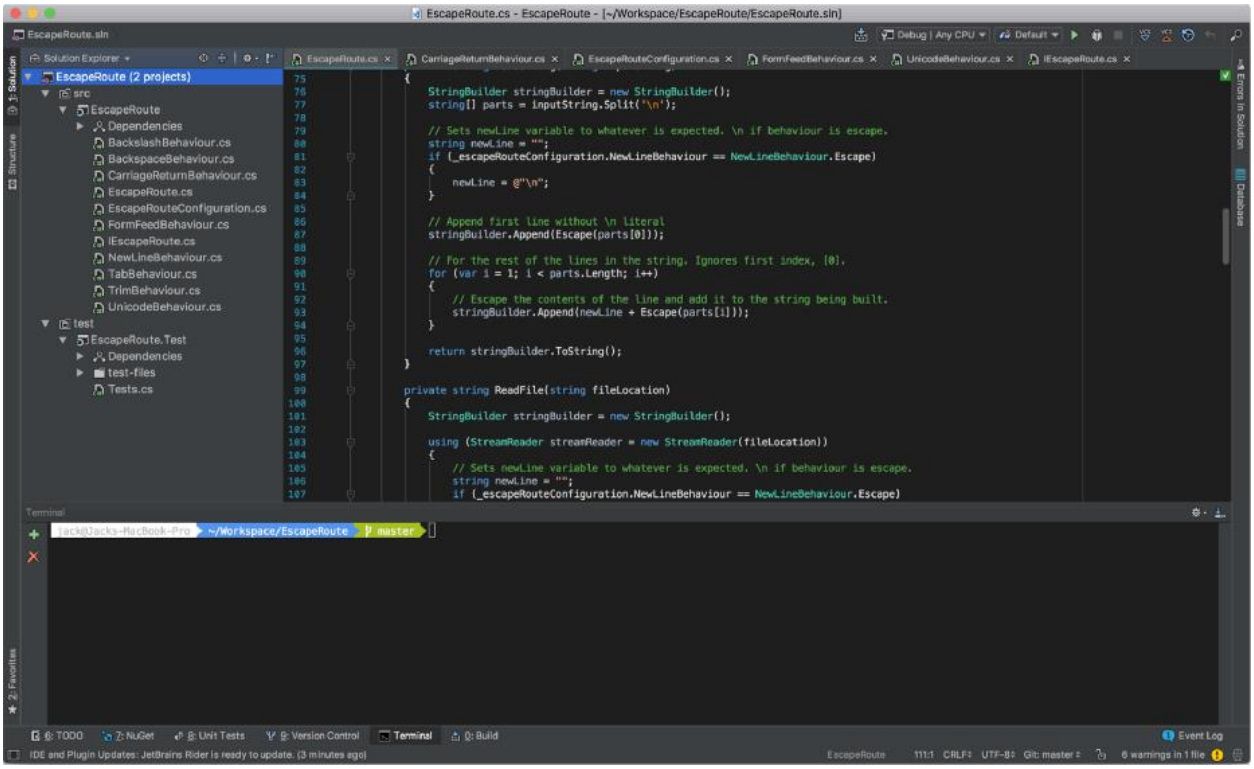


Figure 21: Screenshot of JetBrains' Rider IDE

The industry standards for free source control are Git and Subversion (SVN). SVN is a simple and mature source control option which handles binary files elegantly and allows for file locking in situations where a developer wants sole control over the edits to a file. Git on the other hand, is distributed, much faster, includes access control (which is very useful in open-source projects), and has great support for branches and conflict resolution. Because our project is open-source and our development style includes many branches, we chose Git for source control. Our source code exists in a public repository on GitHub, a free, web-based hosting service for git repositories.

4.3 System Architecture

The WW system architecture will be described using the project's namespace structure, then the design patterns the code employs, and finally a high level explanation of the most important parts of the code structure.

4.3.1 Namespace Structure

The WW project uses namespaces in order to provide structure to the code base. There are two top level namespaces, `core` and `ThirdParty`, and several sub-namespaces within each of those. `core` contains all of the code that the WW team wrote broken up into several namespaces, while `ThirdParty` contains code that is used within WW but was not written by the WW team. The top-level namespace hierarchy is as follows:

- `WorldWizards`
    - `core`
        - `controller`
        - `entity`
        - `experimental`
        - `file`
        - `input`
        - `manager`
        - `menus`
    - `ThirdParty`
        - `JsonDotNet`
        - `SteamVR`
        - `WWUtils`

The sub-namespaces within `core` each have a specific function, as shown and described in the following list:

- `controller`
    - `.builder` - Controllers for scaling and moving the construction grid, switching game object materials, and scaling the scene
    - `.resources` - Resource loading, Asset Bundle loading, and functions for retrieving specific sets of resources
- `entity`
    - `.common` - Type definitions for `WWObjects` and interaction types
    - `.coordinate` - Definition of `WWCoordinate` and WW `Vector3`
        - `.utils` - Helper functions for conversion between `WWCoordinate` and Unity coordinate
    - `.gameObject` - Type definitions for `WWObjects`
        - `.resource` - `WWResource` type definition
            - `.metadata` - Definitions for metadata of `WWObjects` (including subtypes), `WWResources`, `WWDoors`, and `WWWalls`
        - `.utils` - Factory that creates and instantiates `WWObjects`
    - `.level` - Data structure that maintains all of the `WWObjects` in the scene
        - `.utils` - Algorithms for building worlds faster in WW (not currently utilized)

- `experimental` - Files used for testing ideas and developing new features (not utilized)
- `file`
  - `.entity` - JsonBlob definitions and construction (used for saving/loading)
  - `.utils` - Reading/writing JsonBlobs to files
- `input`
  - `.Desktop` - Desktop-specific control scheme and input listener
  - `.Tools` - Control scheme definitions for VR controllers, allows controls schemes to be easily switched at runtime
    - `.utils` - Shared functions of `Tools`
  - `.VRControls` - VR specific input listener, controller collision functions
- `manager` - `ManagerRegistry` that makes sure every `Manager` is a `Singleton` and allows retrieval of each `Manager`, `Manager` definitions, `Singleton` definition
- `menus` - `WWMenu` and `WWButton` definitions, classes for each menu in WW, helper function for adding menu items at runtime

4.3.2 Code Structure

This section will explain the most important code structures of WW. The focus will be on the `Managers` in WW, as they are at the core of how the application works.

When the WW application is started, `WWMain` creates the `ManagerRegistry`. The `ManagerRegistry` then creates a single instance of every class that implements the `Manager` interface. Both the `ManagerRegistry` and all of the `Managers` are `Singletons`, which ensures that only one instance of each is present in WW at any given time. This is important because if any of these classes had more than one instance, much of the work done in WW could be done more than once, and there would be confusion as to which instance to use in other parts of the code. For more information about the Singleton design pattern, or any of the design patterns mentioned in this section, reference the book *Design Patterns* [8].

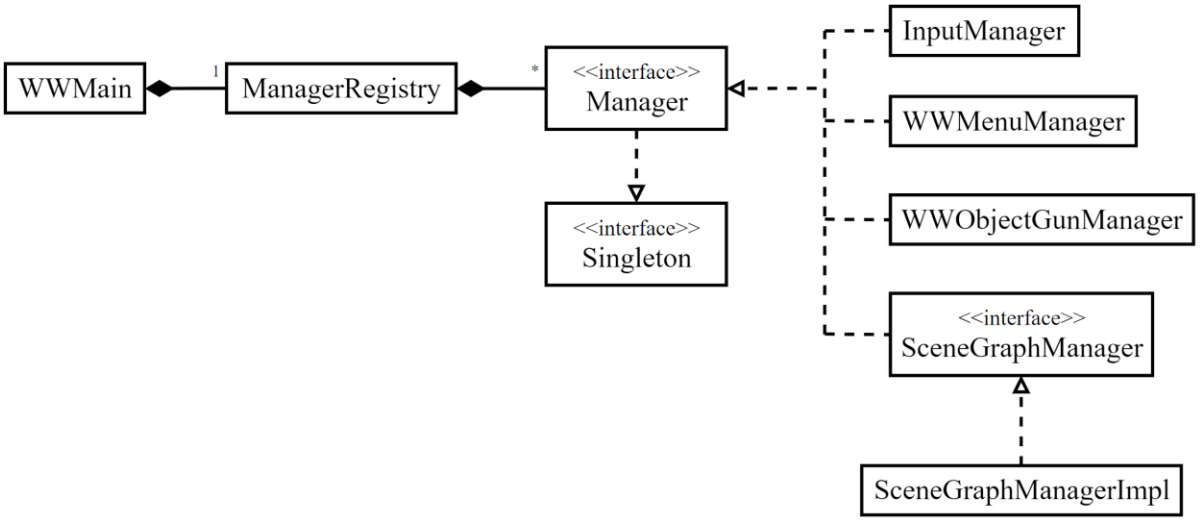


Figure 22: WW Manager UML

Each of the `Managers` has a specific task it carries out. User input is handed by the `InputManager`, which has two `InputListeners`. The `InputListeners` are always the same type, either `VRListener` or `DesktopListener` based on whether WW detects a VR machine or not. Each controller has one `InputListener` attached to it. Each `InputListener` has a `Tool`, which can be of type `StandardTool`, `CreateObjectTool`, `EditObjectTool`, or `MenuTraversalTool`. Each of these `Tools` defines the control scheme and functionality of the controller it is associated with.

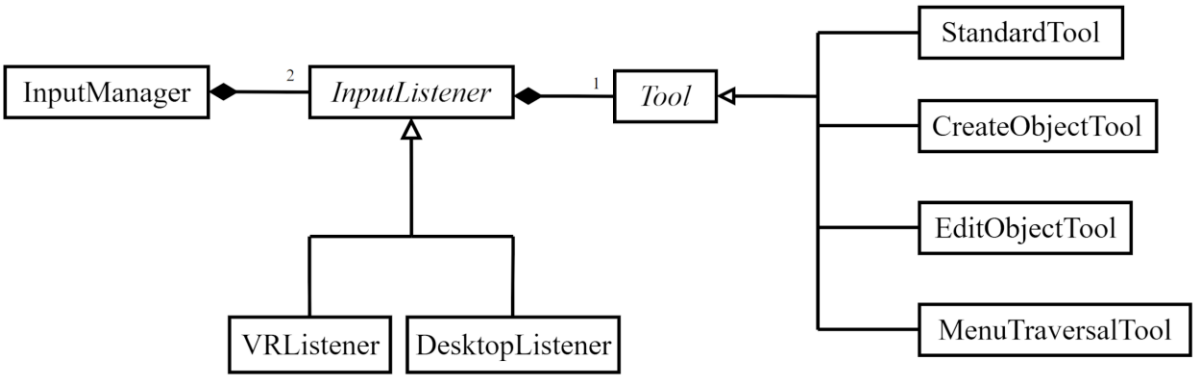


Figure 23: WW Input UML

WW menus are handled by the `WWMenuManager`. The `WWMenuManager` keeps references to all of the `WWMenus` that exist in the application and handles each menu's instantiation. There are currently three `WWMenus` in the application: `ArmMenu`, `PopupArmMenu`, and `AssetBundleMenu`.

Only the AssetBundleMenu is present when using desktop controls. The functionality of the ArmMenu and PopupArmMenu are mapped to keyboard keys in desktop mode.

When WW is started up, the MenuManager gets all of the menus present in the Resources folder, instantiates them and decides whether to show or hide each menu. Then, it stores a reference to every menu in a Dictionary, indexed by the name of the menu. This functionality allows every menu to be accessed easily by any other part of the code base. The MenuManager also has functions to handle showing and hiding menus, and checking if a specific menu exists. The ManagerRegistry is what starts the MenuManager and gives the rest of the code base access to its single instance.

If a developer wants to make a new menu, the menus canvas needs to have a script attached to it that extends WWMenu. The menu then needs to be placed in the Resources/Prefabs/Menus folder as a prefab. WWMenuManager will then know to instantiate it at runtime.

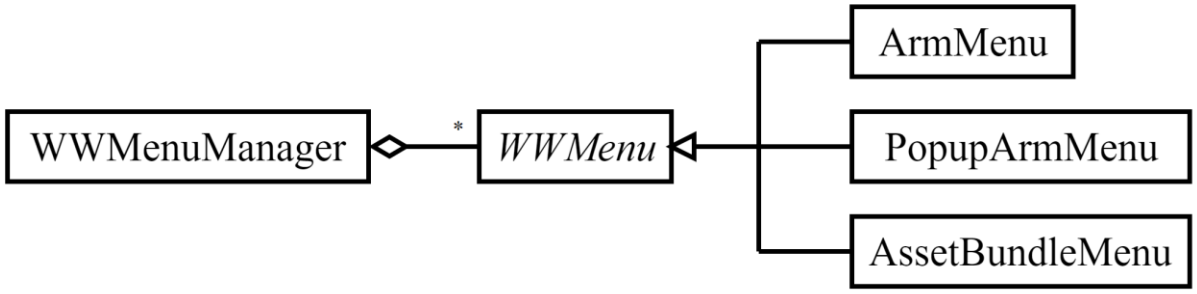


Figure 24: WW Menu UML

The WWObjectGunManager is responsible for keeping track of the WWObjects that the user can currently place in the game world. Users can filter what they are currently placing in order to make it easier to find what they want. Filtering can be done by WWObject type, such as Tile or Prop, or by Asset Bundle. WWObjectGunManager uses a List to keep track of all the WWObjects the user wants to be able to place at any given time. Users can change what group of WWObjects they are placing by changing settings in the AssetBundleMenu, which tells the WWObjectGunManager what group of WWObjects to have available to the user.

Every tile and prop in WW is of type WWObject. WWObjects are created using the WWObjectFactory. The WWObjectFactory implements a Factory design pattern in order to allow other parts of the code to create new instances of WWObjects. Each WWObject has a WWObjectData and a WWResourceMetadata associated with it, which provide the WW system with information for each object such as its ID and WWTransform. The WWTransform contains a WWCoordinate and rotation information. While WWObjects are rendered in Unity using a Unity transform, internally WWTransforms are used by WW for many operations. Each WWObject can be of type Tile, Prop, or Interactable. Currently, WW does not use the Interactable type,

though it is planned for future iterations of the project. It will be a type that defines how and when users can interact with objects of this type.

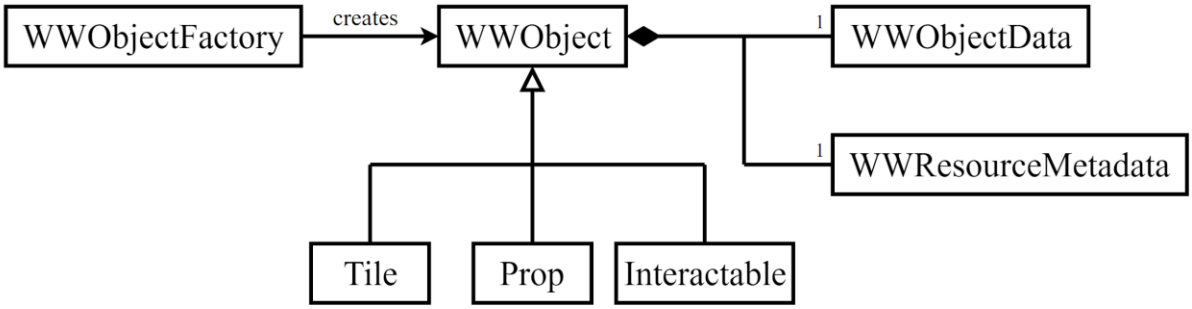


Figure 25: WW Object UML

Instantiated `WWObjects` are stored and handled by the `SceneGraphManager`. The `SceneGraphManager` maintains the state of the scene graph and exposes all of the operations that can be performed on the scene graph.

The `SceneGraphManger` supports the following operations: Adding and removing objects, querying all objects at a specific coordinate index, and determining whether or not a list of objects will collide with existing objects in the scene graph.

Currently the implementation for the scene graph, `SceneGraphManagerImpl`, uses a data structure named `SceneDictionary`. The `SceneDictionary` index is a data structure that efficiently maintains the `WWObjects` in a sparse graph representation. The motivation for choosing a sparse graph representation was to prevent memory from being wasted allocating space for tiles that the player had not yet placed. By choosing a sparse graph representation, we also did not have to impose arbitrary boundary restrictions on the size of the levels that players can create. Lastly, with a sparse graph implementation, it is not necessary to handle the operation of resizing the data structure if a player wishes to build outside of the boundary. The `SceneDictionary` indexes `WWObjects` by both their unique identifier and their coordinate index, allowing for constant time lookups for either method of index. Because objects are indexed by coordinate index, WW can quickly determine whether a tile will collide with existing tiles because it can narrow down its collision check to only the tiles at the coordinate index where the object is being placed.

4.4 Systems

4.4.1 Coordinate System

WW has its own internal coordinate system, which is used when making calculations and saving to files. Externally, `WWObjects` are realized in the Unity Game engine and are rendered at the

correct location after converting from a `WWCoordinate` to a Unity position. These coordinates are used to position objects like tiles and props in WW. A cell is a cube like unit of space that can hold tiles and props. A tile is a graphical asset that can be placed at a cell. Cells have 6 faces which tiles can occupy. The faces a tile occupies is determined by the artist who creates and sets up the tile. Props are placed cells but can also have an additional offset within the cell. The functionality of cells, tiles and props is described in more detail in the Tile System section.

A `WWCoordinate` contains 2 components and is an immutable type. The first value is the `Index` and it has 3 integer values: x, y, and z. The index represents what cell this coordinate is located at. The second value of a `WWCoordinate` is the `Offset` within that cell and consists of 3 floating point values: x y and z. The range of these values are exclusively between -1 and 1. If a value of greater than or equal to 1 is supplied to a coordinate, the corresponding component of the index is incremented by one and that offset component's value is set to 0. The reason for enforcing exclusivity is to ensure that when converting from a Unity space position to a `WWCoordinate`, the same coordinate is produced. This is necessary because while the index component is internal to the cell, overlapping could occur with the offset component. This occurs whenever a coordinate is created. Because coordinates are immutable, the index or offset components cannot be changed on an existing coordinate. Instead, a new coordinate must be created and these exclusivity constraints will be applied. A value of 0 indicates that there is no additional offset. Likewise, if a value of less than or equal to -1 is supplied to a coordinate offset, the corresponding component of the index is decremented by one and that offset component's value is set to 0. The offset is measured from the center of the cell in x, y, z space.

The advantage of World Wizards' coordinate system is that scaling up or down the size of a cell or all cells does not require altering the coordinate of any tiles. While scaling the world does not occur frequently, it is a potential feature that could allow for dynamically adjusting the world in VR based on a user's height. The coordinate is independent. Another advantage of a coordinate being separated into an index and an offset is that the implementation for ensuring that tiles snap to discrete cells is as simple as ignoring or zeroing the offset component of a coordinate.

To handle the conversion between a `WWCoordinate` and a Unity `Vector3` position, a utility helper class named `CoordinateHelper` has functions to convert between these two coordinate systems.

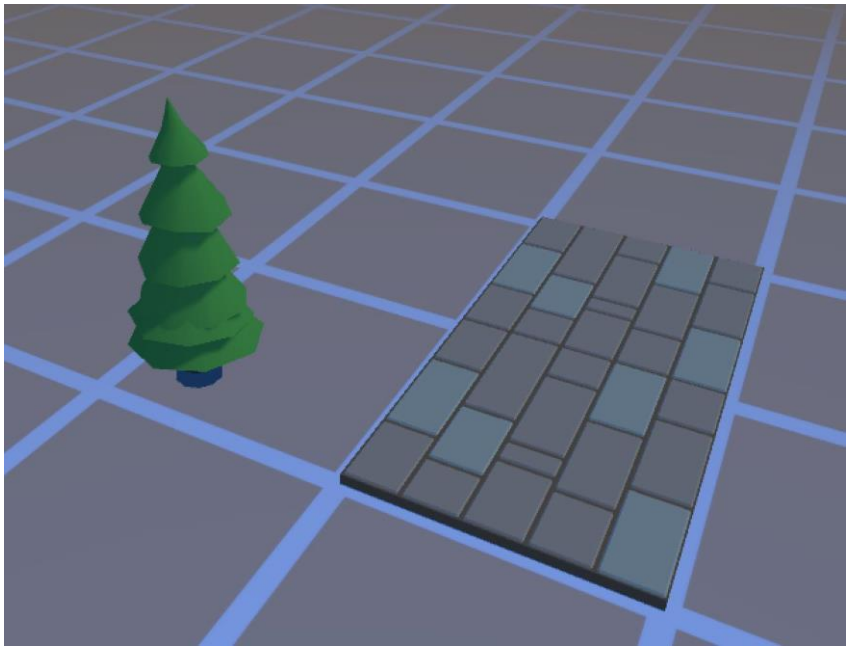


Figure 26: Two floor tiles and a tree prop on the grid plane

In the above figure, the tree prop exists with a coordinate index of 0, 0, 0. Its offset however is approximately -0.5, -.999, 0.5. This is its position relative to whatever coordinate index it exists at. The two tiles have offsets of 0, 0, 0. The floor tile directly to the right of the tree has a coordinate index of 1, 0, 0 and the tile diagonally to the right of the tree has a coordinate index of 1, 0, 1. If the entire world scale was to change, these internal `WWCoordinate` values would remain unchanged. Simply the scale factor that is multiplied when converting from a `WWCoordinate` to a Unity position would change. The Unity scale would also change and this would be reflected on screen as the actual models being rendered in Unity are scaled.

4.4.2 Saving and Loading.

WW saves the minimal amount of data for a level to a JSON formatted file. Rather than making the entire `SceneGraph` serializable and saving the entire data structure to a file, a JSON string is generated that contains the minimal amount of data necessary to fully recreate the scene. For example, a parsable JSON string for a level containing a single tile would look like the following example.

```
[
  {
    "children":[

    ],
    "wwTransform":{
      "coordinateJSONBlob":{
        "indexX":-1,
        "indexY":0,
        "indexZ":-1,
        "offsetX":0.0,
        "offsetY":0.0,
        "offsetZ":0.0
      },
      "rotation":900
    },
    "id":"33c4b0a4-4a41-4c41-840f-212328b355e1",
    "parent":"00000000-0000-0000-0000-000000000000",
    "resourceTag":"assets/arteria/tiles/medwall_c.prefab",
    "type":0
  }
]
```


Figure 27: An example JSON file for saving and loading scenes into WW

This JSON blob stores the list of children of the `WWObject`, the transform which contains the coordinate, the GUID given to this `WWObject`, the parent, the type of the `WWObject`, and the resource tag of the asset to load from the asset bundle, which in this case is a relative file path. Note, this `WWObject` does not have any children, which is why the children array is empty, and there is no parent so the parent is set to the default of a GUID.

4.4.3 Tile System

In WW, rather than limiting the system to only allow a single tile to occupy a cell, multiple tiles can occupy the same space so long as they do not collide with each other. In this context, a collision occurs when the defined face of a tile, which the artist or creator of the tile asset dictates, is overlapping the defined face of another tile. Each tile is graphically a custom art asset, and multiple tiles can be combined in the same space.

4.4.3.1 Large and Redundant Tile Combinations

If only a single tile can occupy a cell, many permutations of tiles have to be created and lots of mesh data is needlessly duplicated. Consider a tile where there is a floor and a wall. Now consider a corner tile in the same style for this floor and wall tile. Next consider a floor by itself, and a wall by itself. Finally consider wanting to mix and match different styles of floors and walls. The number of combinations becomes extremely large. If only one tile can occupy a cell, a unique tile would have to exist for all of these combinations. This system is in part similar to how buildings are constructed in The Sims games [24]. There are simply too many permutations for different arrangements of tiles in The Sims games, that a system allowing combining various tiles becomes necessary.

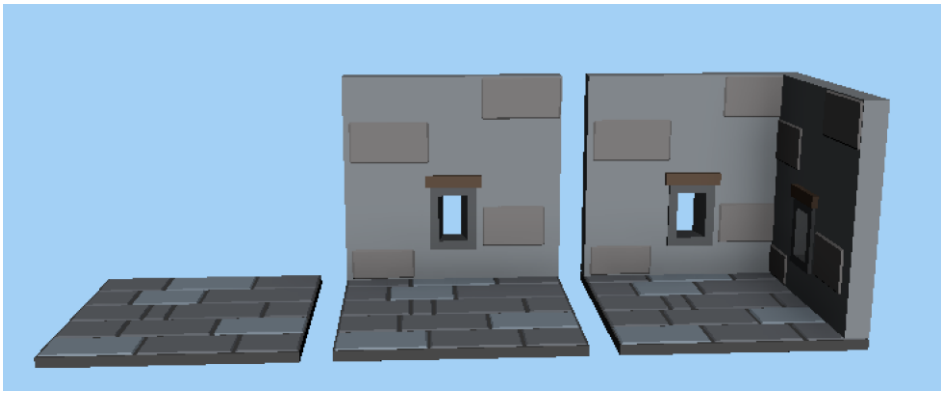


Figure 28: Three Unique Tiles

4.4.3.2 Tiles

Multiple tiles can exist in the same cell so long as they do not share the same face. This means that the above three unique tiles, and many more, can be made from a composition of just two tiles in our system. Currently, the faces a tile can occupy can be thought of as a 6 bit mask. Each bit represents an occupied face of cell. Cells have six faces, and can be thought of as a cube in 3D space. We refer to these six faces as Top, Bottom, North, South, East, and West. A tile can occupy space in all, none, or some of the faces. So for example, a floor tile would most likely occupy space in the Bottom face. This means only one floor tile could exist in a cell. If the art asset for a floor tile is not meant to conflict with existing tiles, perhaps the tile is graphically just a layer of leaves, then the artist setting up the tile can decide that the tile occupies no faces. Note that these faces are internal to the tile index that they occupy, so a tile that occupies the bottom face at a cell with a y index of 1 does not collide with a tile that occupies the top face at a cell with a y index of 0. Tiles can also be rotated along the y axis at discrete intervals of 90 degrees.

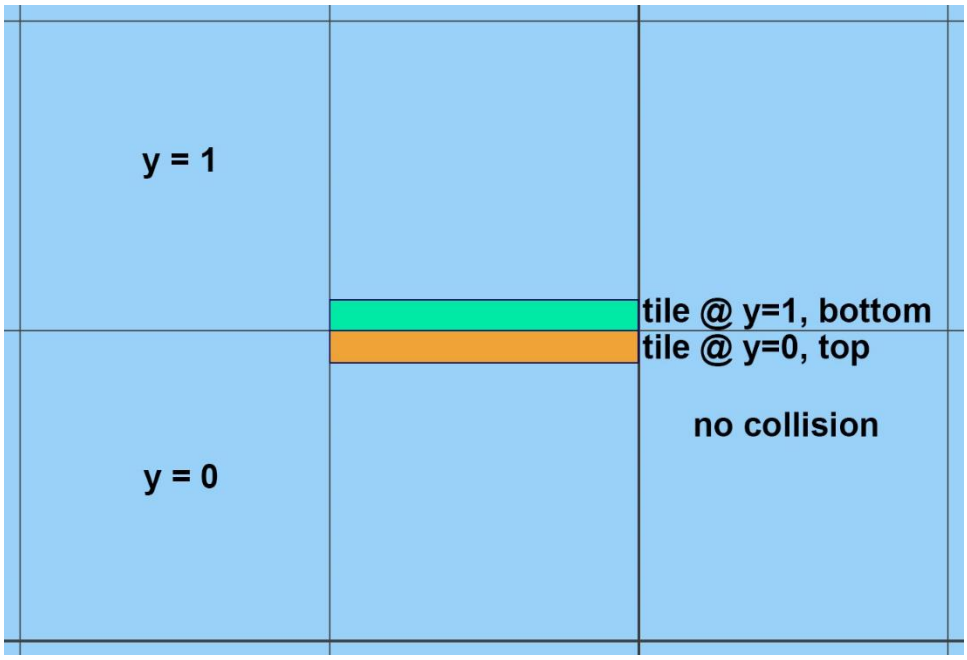


Figure 29: Diagram showing that faces are internal to the cell the tile is placed at

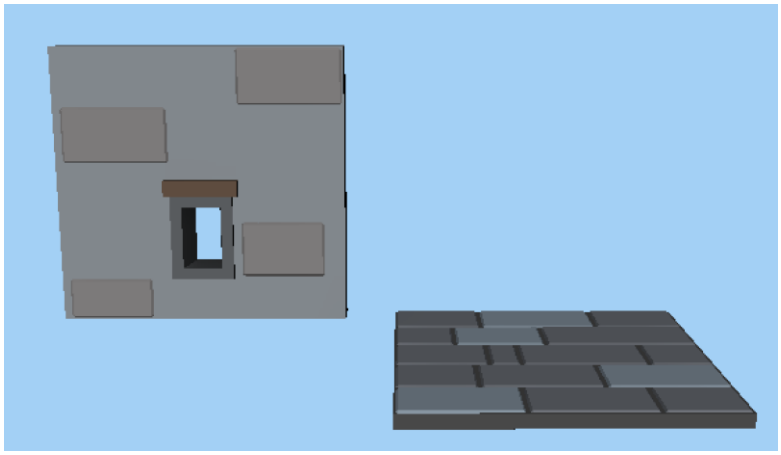


Figure 30: Two tiles, a floor and wall, can be combined in multiple ways

WW not only supports tiles, but also props. Props are ideal for objects like trees, rocks, and furniture. Props do not occupy any faces in a cell. This means a player may place as many props as they want in a cell. The other difference between props and tiles is that props can be placed anywhere.

4.4.3.3 Placement of Objects

World Wizards has a grid plane that the user can move up and down vertically. The grid plane defines the current layer. Players can only place tiles on the current layer. The height of this current layer determines the y-axis for newly placed objects. When placing an object, WW checks with objects that already exist in the scene graph and determines if the object being

placed can actually fit without colliding. The scene graph can also determine if there are any possible rotations that would make this object being placed able to fit without colliding. If a tile does not fit, it is not placed into the scene graph. If the user is currently using the edit object tool, the tiles being moved will snap back to the last valid position. If the user is attempting to create a new object with the create object tool, the object will not be created.

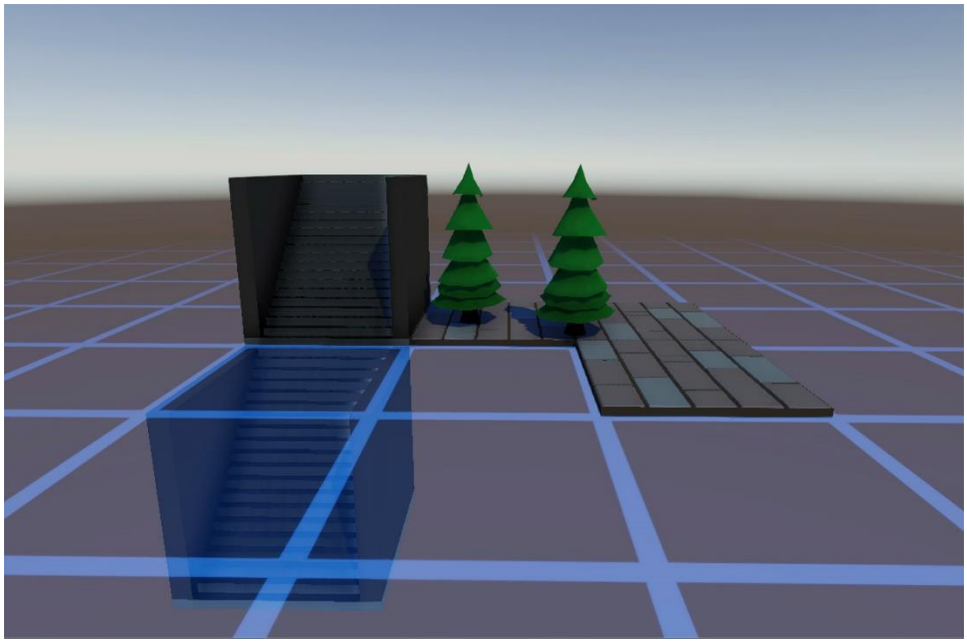


Figure 31: A view of WW demoing tiles being placed on the construction grid

Props, however, snap to the underlying surface below them. If there are no surfaces below when placing a prop, the current layer from the construction grid is used to determine the y-axis. Props can be of arbitrary size and be larger than tiles. Props are placed relative to cells and their position is represented using the offset component of a coordinate.

Interactables, if they existed in our current implementation, would likely be placed similar to a props. However, there is no reason a tile could not be made interactable as well and therefore would be placed with same logic used to place tiles.

4.5 Art Pipeline

The art pipeline is closely related to how the tile system is designed. The goal of the art pipeline is to allow artists to be able to easily configure and bring custom art tiles and props into WW.

4.5.1 Workflow

An artist will import their 3D tiles and props in Unity. WW provides the Tile Setup Guide Prefab which serves as a visual guide to assist in setting up the appropriate faces that the tile may occupy.

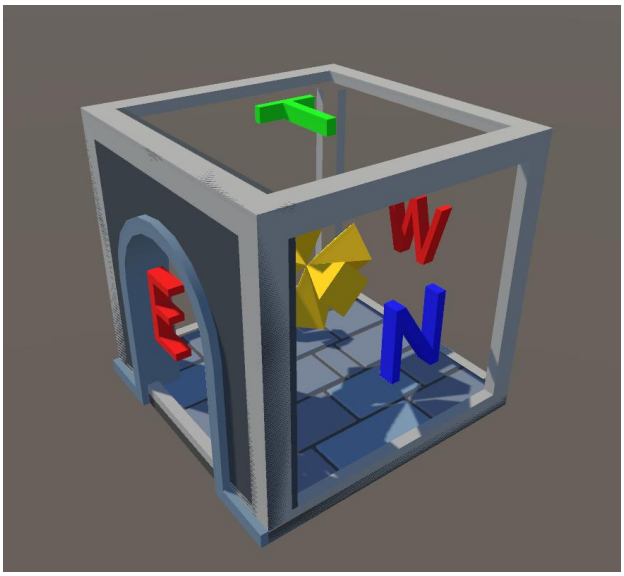


Figure 32: The Tile Setup Guide visually shows the faces a tile can occupy

This visual guide assists with determining what faces the tile should occupy. Tiles also must have a pivot that is in the exact center of the cell. This makes implementing the rotation of tiles much easier. The visual guide shows in yellow where the pivot point should be. Props may have arbitrary pivots and ideally will be at a pivot point that makes sense for the rotation of the prop. For example, a tree prop would most likely have its pivot point at the center of where its trunk would naturally touch the ground.

For the asset to be used in the game, the user must attach a `WWResourceMetaData` component to the root of the tile or the prop GameObject. All WW objects (`WWObjects`), which includes tiles and props, must have this component attached to their root before being exported to an asset bundle, otherwise the WW system will be unable to recognize the object. The `WWResourceMetaData` has a sibling script named `WWResourceMetaDataEditor` which serves as a Unity editor friendly UI that speeds up the configuration process for describing the type and properties of the `WWObject`. This editor script allows the user to specify the type of the `WWObject` as well other properties like which faces of the tile occupy space and should collide.

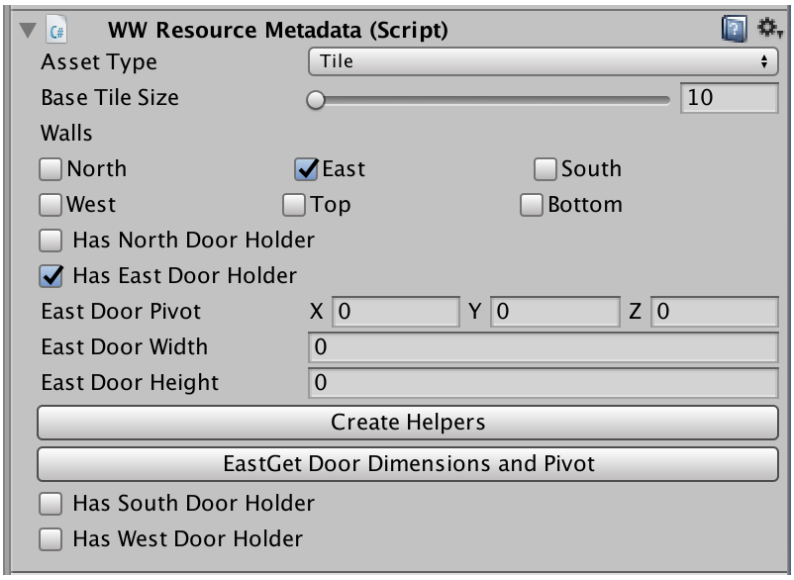


Figure 33: The WWResourceMetaDataEditor UI allows easier configuration

WW contracted art tiles from 3D artist, Steve Finney, whose professional work can be viewed at his website[6]. Finney provided a Unity Package containing a Unity Scene which held the various art tiles and props he created. We attached a `WWResourceMetadata` component to each individual tile and prop. Then, using the Tile Setup Guide, we were able to easily determine which faces each tile occupied. After setting up all of the tiles and props, we exported them to an asset bundle and were then able to use them in WW.

Figure 34: A screenshot of a small level built in WW using the tile set created by Finney

[6] https://arteria3d.com

4.5.2 Asset Bundles

An important feature of our application is the ability for users to add their own art assets to the game. This is accomplished by utilizing asset bundles, a feature of Unity which allows users to bundle up many art assets into a single file. Our application will load any asset bundle which it finds in the AssetBundles and if the prefabs inside of it are valid, they are added to the game as player-usable tiles or props. In order for a prefab to be valid it must have the `WWResourceMetadata` component attached to it.

The easiest way to create asset bundles is with the `AssetBundleManager`, a tool provided by Unity which extends the editor. After installing this tool, the inspector panel for any component will include a field for choosing an asset bundle to put the component in. When all desired assets have had the `WWResourceMetadata` component attached and this field correctly assigned, there is a drop down menu option to build asset bundles.

5. Usability Evaluation

The WW team conducted a usability evaluation of the WW application in order to determine how it could be improved and the quality of our design decisions. We were specifically interested in how user-friendly our controls were, if and when users were dizzy or nauseous, and any bugs in the software. The results of the usability evaluation will be used by future teams to help correct any mistakes we made and provide guidance on the most important aspects of the next iteration of the project. This section explains our evaluation procedure, the general feedback we got from participants in the usability evaluation, and the data we collected.

5.1 Procedure

Each session of the usability evaluation started with one member of the WW team explaining the study procedure to the participant, going over the application controls with them, and getting them set up with the HTC Vive headset and controllers. We then allowed them between five and ten minutes, to get acclimated to the controls and using a VR application if they had never used VR before. We planned on cutting participants off after ten minutes, but never had to. During this acclimation time, participants were in a starting area that the WW team created, as shown in Figures 35 and 36. We asked the participant to express any thoughts or concerns they had while playing, using a “think-aloud” protocol, and had a member of the WW team take notes about what they said. Once the participant expressed that they felt comfortable with the controls and being immersed in VR, we presented them with the following three tasks, shown in Figure 37:

1. Move to the big red table (one operation)
2. Move the green cube onto the red target point (three or four operations)
3. Recreate the structure on the foundation next to it (nine or ten operations)

A member of the WW team timed the users completing each task, starting the timer after each task was fully explained. Finally, participants were assisted with removing the VR equipment and asked to fill out a survey. The entire process took between 20 and 30 minutes for each participant.

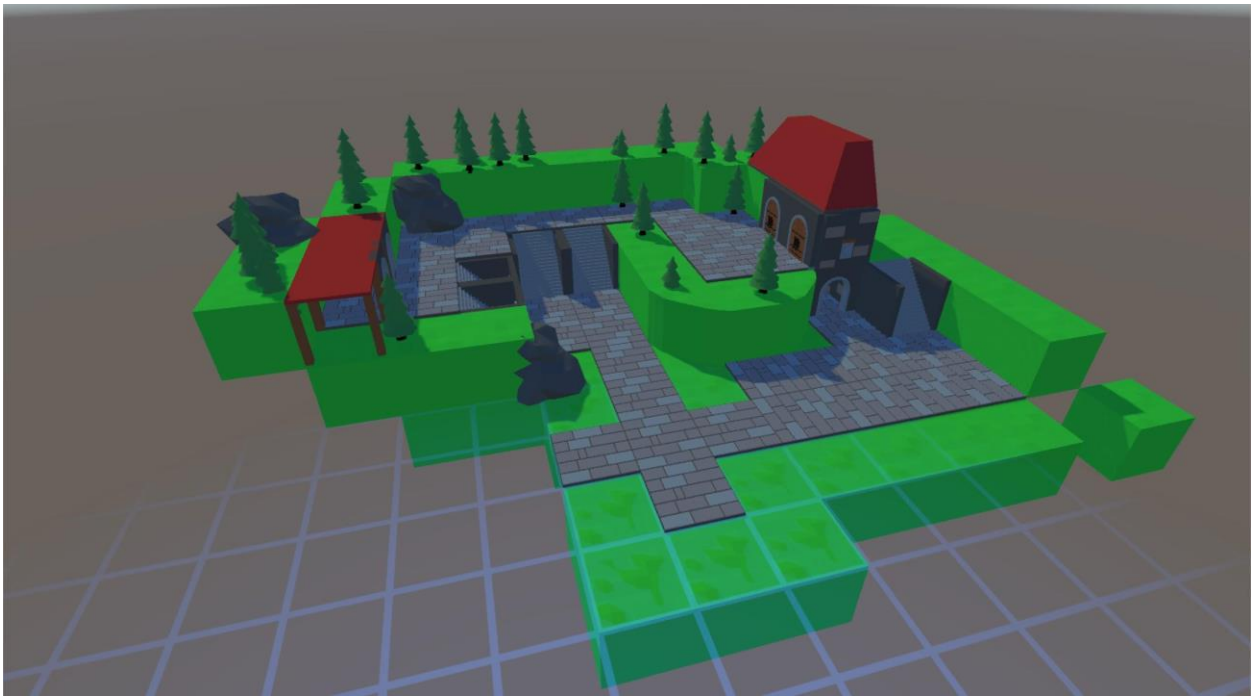


Figure 35: The entire starting area from a birds-eyes view

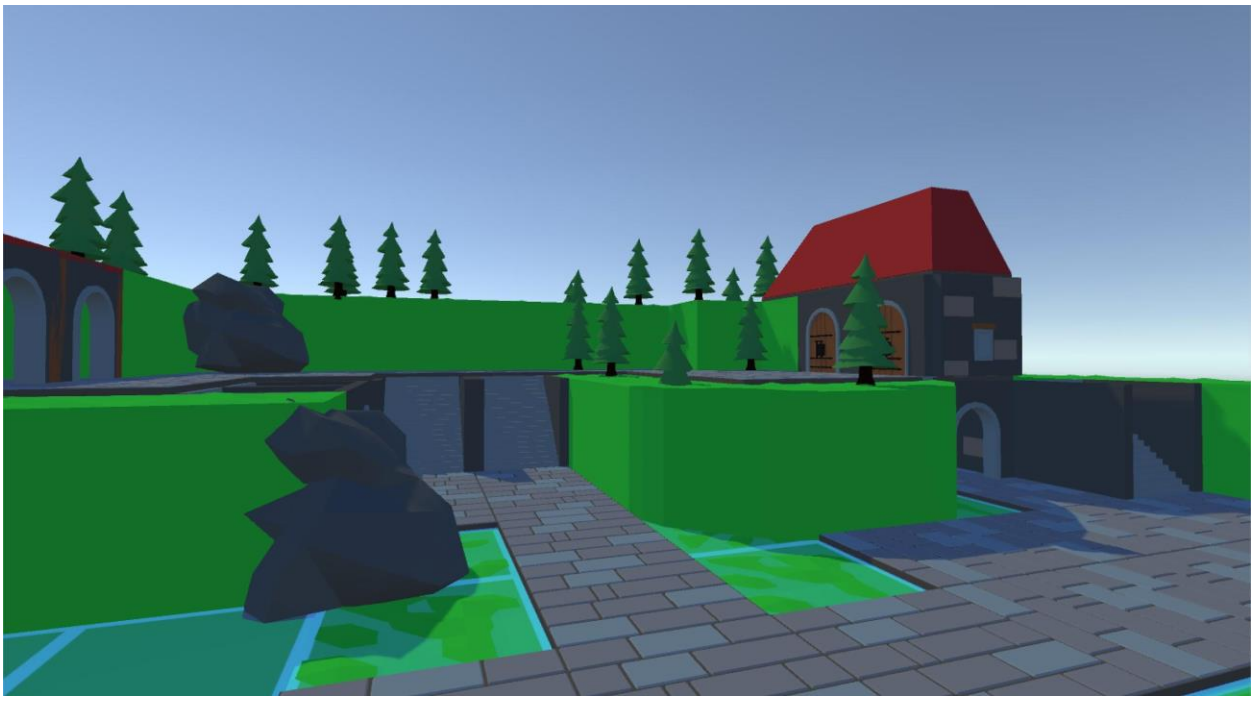


Figure 36: A section of the starting area

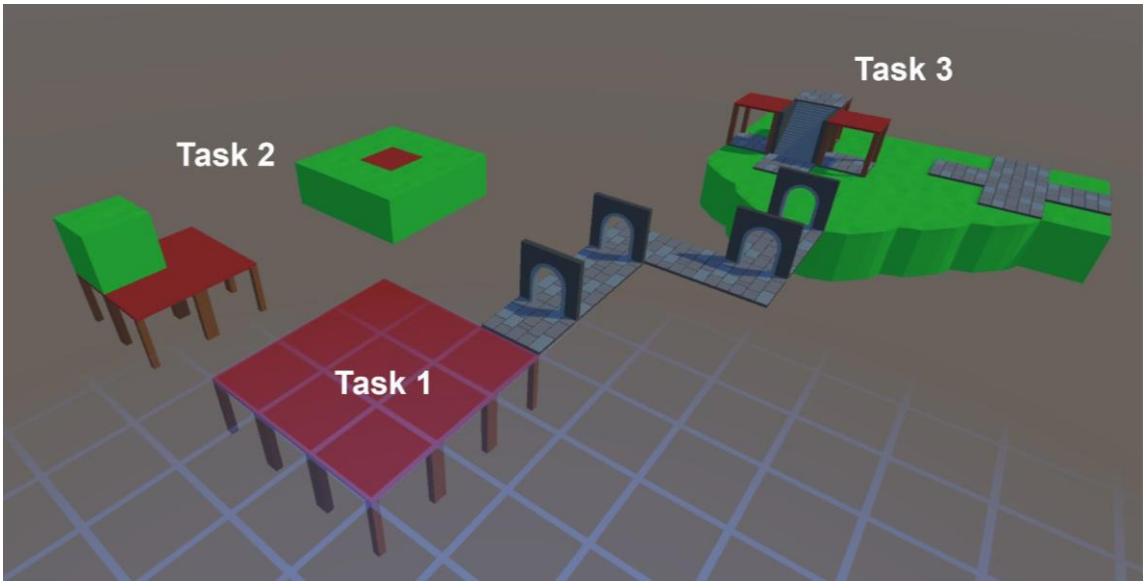


Figure 37: The three tasks that participants were asked to complete

5.2 Rationale

The usability evaluation was set up in such a way as to give the WW team the most useful feedback about the WW application as possible. The in-application procedure was designed to help the WW team find bugs in the application and understand if the controls were intuitive. The first five to ten minutes of each session, where participants were allowed to do whatever they wanted in order to get acclimated with the controls, helped the WW team to discover new bugs. The tasks we created for the participants to complete allowed us to evaluate whether the controls were easy to use and remember.

The survey following the tasks was designed to give us quantitative feedback that we could analyze. The survey questions asked about the controls, how nauseous participants felt, and whether the application included enough functionality. The survey also asked open ended questions about any feedback participants had and demographic questions. All of these questions allowed the WW team to run statistical analyses to determine correlations between the data and draw conclusions about the next steps for the WW application.

5.3 Feedback and Results

Our sources of feedback were the notes taken by WW team members during testing, the timing of the trials, and the survey responses from participants in our usability evaluation. The notes taken during user testing consist of observations the WW team made while participants played WW and any interesting comments and questions each participant said. We timed each participant completing each trial to get a sense of how well participants understood the controls.

The survey asked participants about their experience with WW in general and asked a few specific questions that we had regarding the WW application. All notes that were taken are shown in Appendix D, all of the trial times are in Appendix E, and the raw survey results are in Appendix F. In this section, we will explain the most common and interesting notes we took, describe the results of the timed trials, and present the results of the survey. The conclusions we drew based on these results are discussed in the following section.

5.3.1 Notes

Looking over the notes as a whole provided many insights about the usability of the WW application. Participants expressed both difficulty and satisfaction with movement, world building, the arm menu, and the application as a whole.

5.3.1.1 Movement

One of the most common kinds of notes the WW team took were related to confusion about the control scheme. Many participants accidentally hit the wrong button(s) when trying to move or thought that the movement controls were difficult to perform.

Participants tended to have the most issues with the controller's trackpad, which is one of the main ways to move in WW. The trackpad has two different kinds of input: touch and press. The trackpad can register a person's touch and the location of the touch on the trackpad, and can register a person clicking different sections of the trackpad. The difference between these two types of input caused confusion for many participants. They had a hard time remembering if a control was mapped to the touch function or the press function. Because the player has to touch the trackpad first in order to press it, there were also issues with both inputs being registered by WW at the same time.

The trackpad problem was most prevalent in regards to movement - five participants accidentally moved upward and forward at the same time, because horizontal movement is mapped to the touch function and vertical movement is mapped to the press function. There was also a recurring issue with moving downward, as it was difficult to hit the trackpad button correctly in order to do so.

Four participants did not enjoy the sensation of moving, especially vertically. One participant expressed that it made them feel slightly nauseous, another said that movement "feels weird", and one participant seemed dizzy and almost lost their footing while moving vertically.

Conversely, participants also had positive feedback regarding movement. Participants expressed that the controls for movement were easy to remember compared to the rest of the controls in WW. A few people expressed that they enjoyed teleporting using the trigger and that it was "much more convenient" than using the trackpad to move.

5.3.1.2 World Building

Another frequent point of feedback in the notes were about world building. In particular, participants tended to have issues with cycling through objects and differentiating preview objects from placed objects.

When a player wants to place an object in the world, they have to cycle through all of the objects currently available to them to find the one they want to place. This is done by tapping the top or the bottom of the right controller. A preview objects are shown on the grid wherever the right controller is pointing. Once the player finds the object they want, they have to hold down the trigger on the right controller, point to where they want to place the object, and then release the trigger to actually place it in the world.

Problems with the control scheme for cycling and placing objects was a major point of feedback from participants. Two participants were not pointing the controller at the grid while cycling, which caused confusion because the preview objects were not being shown in front of them. Four participants thought that preview objects had already been placed because they look exactly the same as placed objects. Because cycling objects was done by touching the trackpad, some participants would accidentally cycle objects while trying to perform the press function of the trackpad, similarly to the movement issue explained in the section above.

More points of feedback we received involved the construction grid. Two participants were confused that when placing objects in the world only tiles were constrained to the grid, and props were not. The WW team implemented this intentionally, since props are not constrained by cells we thought they shouldn't be constrained to the construction grid either. We didn't realize that implementing it this way would cause so much confusion. Moving the grid by accident was also a common mistake made by participants.

Z-fighting, a 3D rendering issue caused by two objects occupying the same space, was noticed by two participants while placing objects in the world. The WW application does not allow tiles to be placed where they would collide with the faces occupied by another tile, but preview objects and selections do not prevent collision. Therefore, when a player is previewing an object or moving a selection of objects, they can move it into the same space as a placed tile, causing z-fighting to occur. An example of z-fighting in WW is shown in Figure 38.

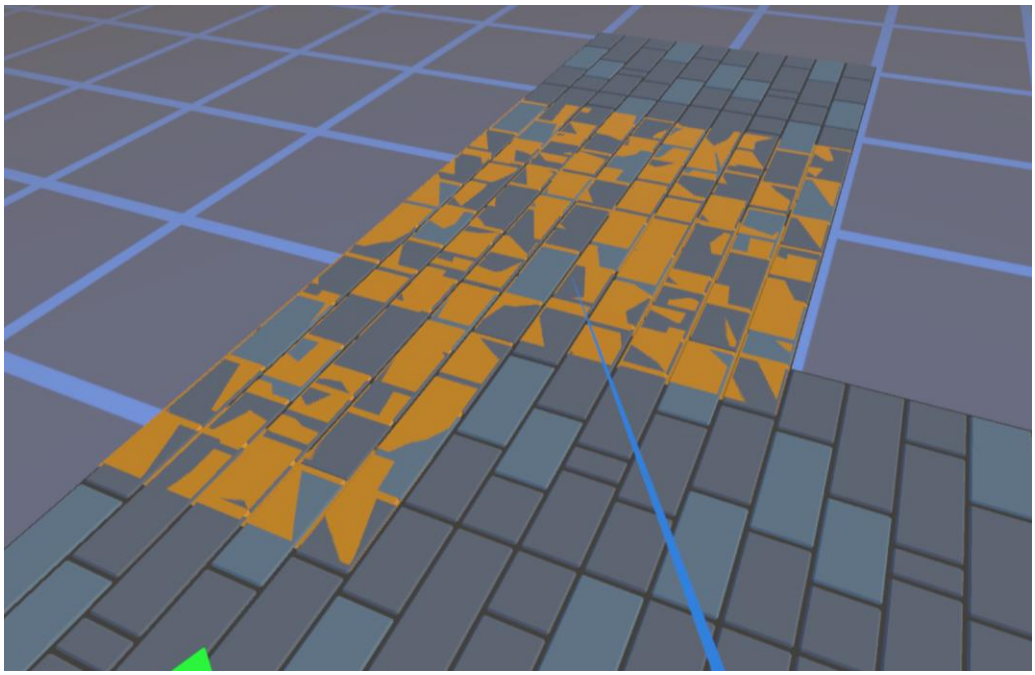


Figure 38: Z-fighting in WW

Participants also had positive feedback regarding world building. One participant said that the grid system was convenient because it made everything line up nicely. Three participants seemed to relish in the creativity of building whatever they wanted. Two participants stated that they liked the laser pointer showing where you were placing objects.

5.3.1.3 Arm Menu

The arm menu was a frequent point of comment during the usability evaluation. The arm menu is parented to the left controller in such a way that it "sits" on top of where the player's arm is relative to the controller. The function of the arm menu is to switch the control scheme on the right controller between Object Placement mode and Object Editing mode and give the player visual feedback about what mode their controller is in. There are two buttons on the top that correspond to the two different modes and a button on the bottom that toggles between showing and hiding the top buttons of the menu. Players must touch the buttons with the right controller to push them. The button highlighted in green is the mode that the right controller is currently in.

Five participants stated that pushing the buttons on the arm menu was difficult or awkward to do. It was noted that some participants did not recognize how the arm menu was parented to the controller, and had difficulty moving their hand and arm in such a way as to see the menu comfortably. One participant actually punched their arm with the controller trying to push the bottom button of the arm menu. The locations of the top buttons relative to each other was also a problem, as two participants hit both buttons at once by mistake.

Another point of confusion about the arm menu was the laser pointer on the right controller interacting with the top buttons. Originally, the WW team planned on players being able to interact with the buttons by either touching them with the right controller or pointing with the laser and clicking with the trigger. The team did not implement the point and click functionality in time, but mistakenly left the laser interaction on the top buttons. Therefore, quite a few participants attempted to point and click the top buttons to no avail.

The arm menu also received praise from participants. Three participants expressed satisfaction using the arm menu, and it seemed easy for participants to remember how to use, as very few people asked how to use it after being told the first time. When participants first saw the arm menu in the WW application, many expressed interest in it.

5.3.1.4 Overall

The WW application as a whole garnered great responses from user study participants. Two participants exclaimed that they felt "powerful" playing WW. Two participants relished in the creativity WW provided and attempted to create extravagant structures. One participant called the application "enthralling", and another said "I'm having the time of my life" while playing. While these responses may have stemmed from experiencing VR, the participants who made these statements were some of the more creative out of the user study participants – the people who spent the time to make their own elaborate structures in WW during our user testing. They very well could have been responding to having fun using the application itself.

Contrary to the praise, the main points of concern with the application overall were the general control scheme, the user interfaces, and the bugs that were discovered.

5.3.2 Trials

The WW team decided to time participants as they completed the trials in order to get a sense of how well they understood the controls and how well the controls were designed. During the trials, we allowed participants to ask questions about the controls in hopes that this would alleviate any frustration they had while still adding time to their total to get an accurate representation of their level of understanding.

The average times for each trial were close to what the WW team expected. The trials were ordered by difficulty, with the easiest being the first trial, so naturally the average time for each trial increased for each subsequent trial. The average time for each trial is shown in Figure 39. Every participant was able to complete every task.

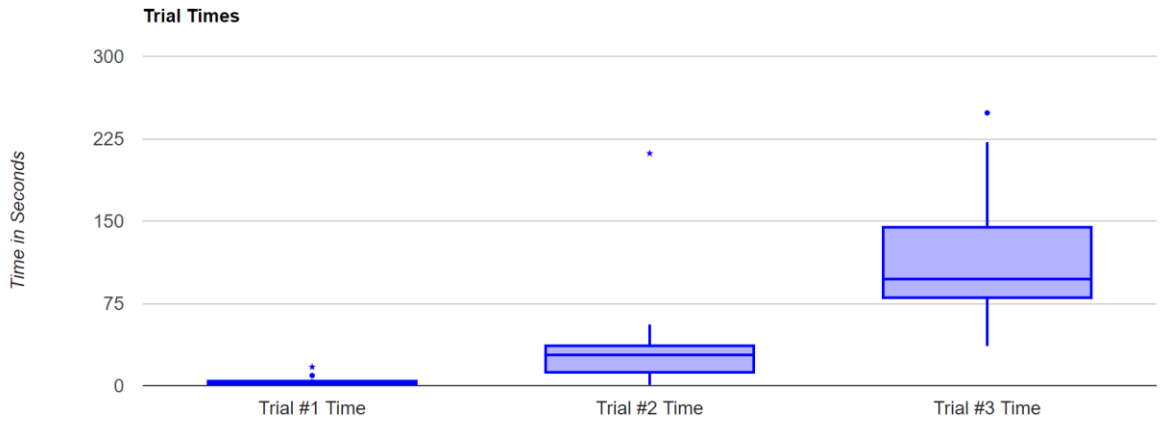


Figure 39: A boxplot of the times (in seconds) of all three trials

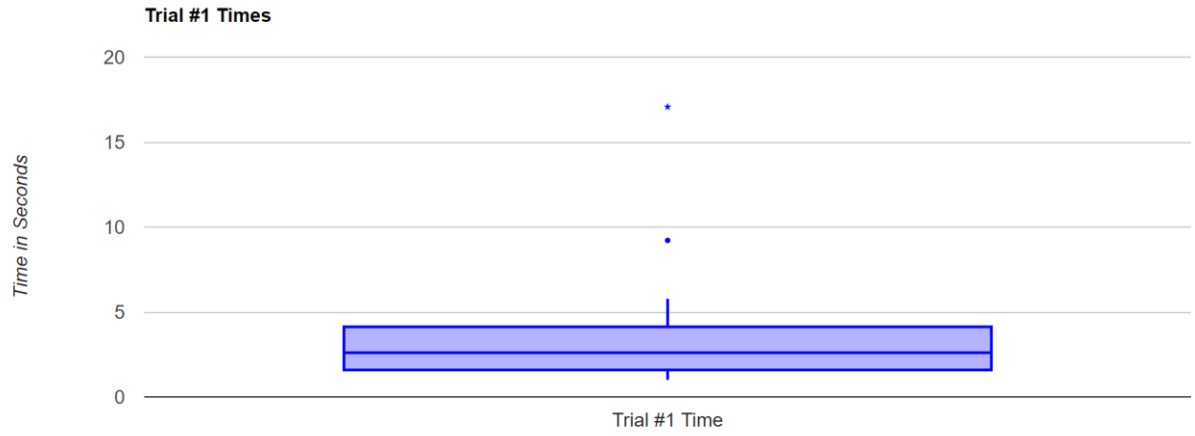


Figure 40: A boxplot of the times (in seconds) for trial #1

There was no obvious relationship between hours of VR played previously and trial completion time. There were 6 participants in the "0 hours" group, 5 participants in the "Less than 1 hour" group, 6 participants in the "Between 1 and 5 hours" group, 1 participant in the "Between 5 and 10 hours" group, and 2 participants in the "More than 10 hours" group. Note that only one participant chose "between 5 and 10 hours" of VR played, so the average trial completion time data points for that category correspond to only one participant and is an outlier in the data.

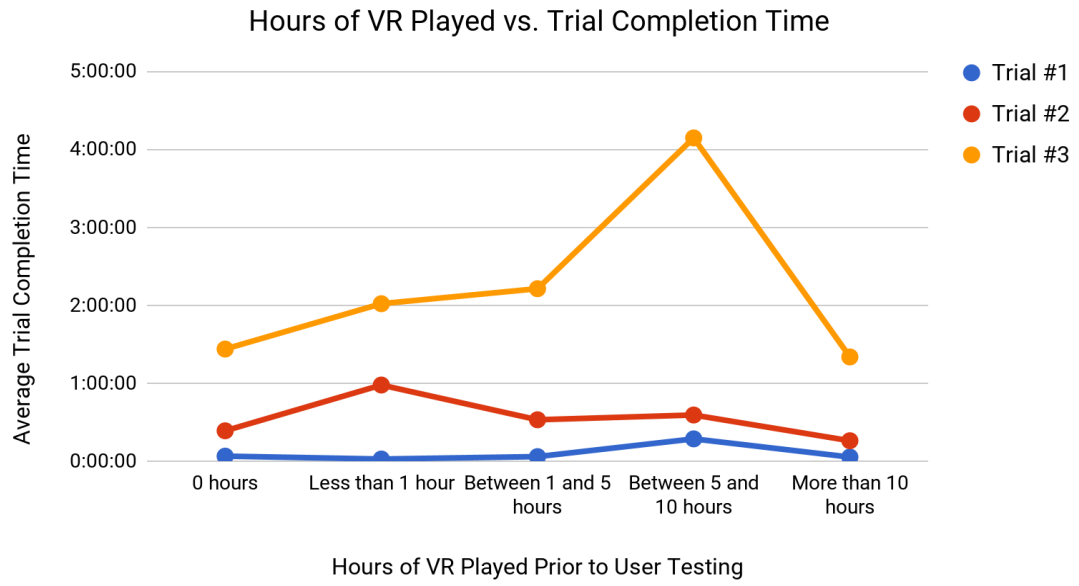


Figure 41: Comparison of hours of VR played previously and trial completion times

Note that our survey asked participants to self-identify their gender. Every participant responded with either "woman" or "man" when asked to self-identify, hence why we refer to only women and men throughout this section. The survey responses to that question are explained in further detail in the next section, with Figure 55.

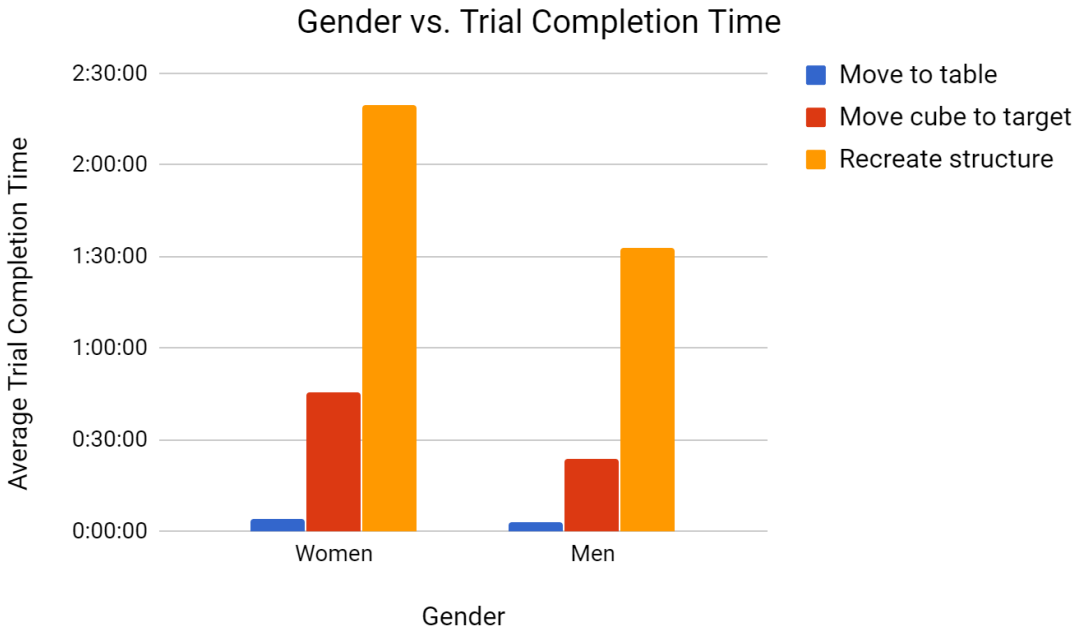


Figure 42: Comparison of gender and trial completion time

5.3.3 Survey

The usability evaluation survey was created in order to gather quantitative data about the WW application. Here, each section of the survey will be discussed, including the rationale behind and responses for each question. Note that our usability evaluation consisted of only 20 participants, so results should be interpreted with a mild amount of caution.

5.3.3.1 Likert Scale Questions

The first section of the survey consisted entirely of Likert scale questions - where participants must state their level of agreement with each statement. The WW usability test survey used a Likert scale that had 5 answers: strongly agree, agree, neutral, disagree, or strongly disagree.

The first question of the survey asked participants how nauseous they felt while playing WW. The WW team chose to ask this for a few different reasons. The first reason is that VR applications have a tendency to make some people feel sick [14]. Also, the team got informal feedback prior to formal user testing regarding the player movement in WW making some people feel ill, and wanted to see if the changes we made improved upon that.

Based on the results, more than three quarters of participants either disagreed or strongly disagreed that they felt nauseous while playing WW. Only one person said that they agreed with the statement, and nobody strongly agreed.

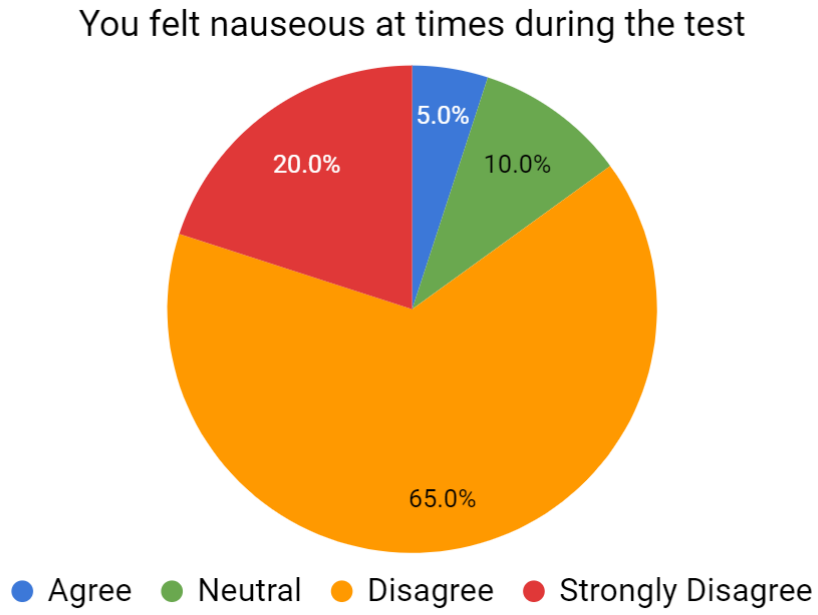


Figure 43: Nauseated-ness survey question results

There was no significant difference between the responses of self-identified women and men when asked to rank how nauseous they felt while playing WW. In fact, calculating the mean of the responses shows that the average response for both groups was exactly the same.

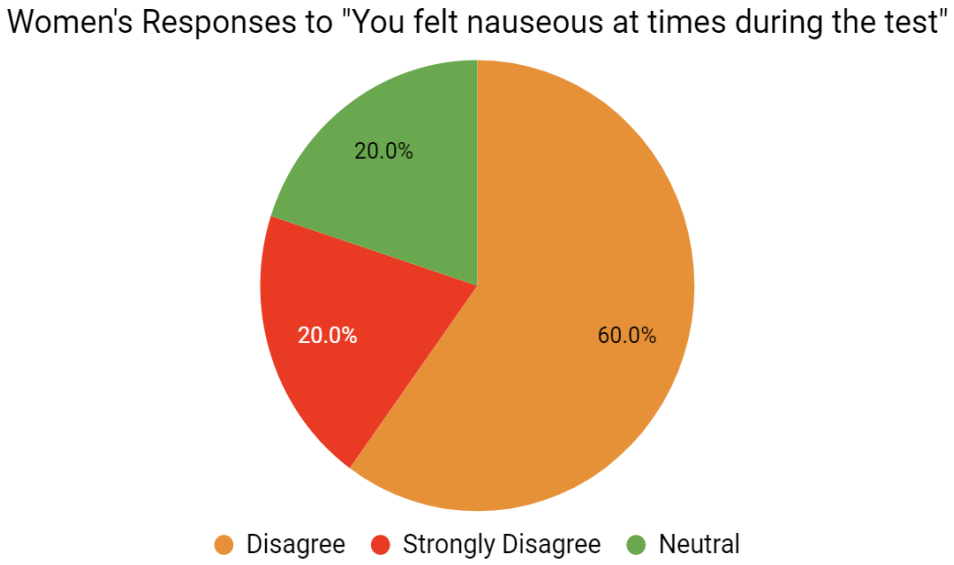


Figure 44: Women participants' response to nauseated-ness survey question

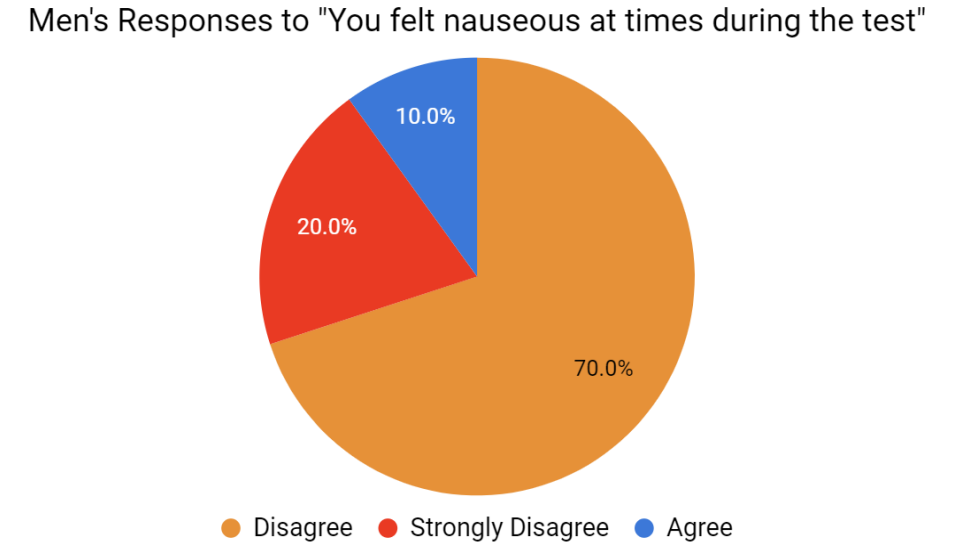


Figure 45: Men participants' response to nauseated-ness survey question

The next question was aimed at the participants' overall experience using a VR application. We wanted to ensure that participants in our user study had a positive experience with VR overall. The responses show that this expectation was met, as every participant either agreed or strongly agreed that they would try a VR application again after their experience in our user study.

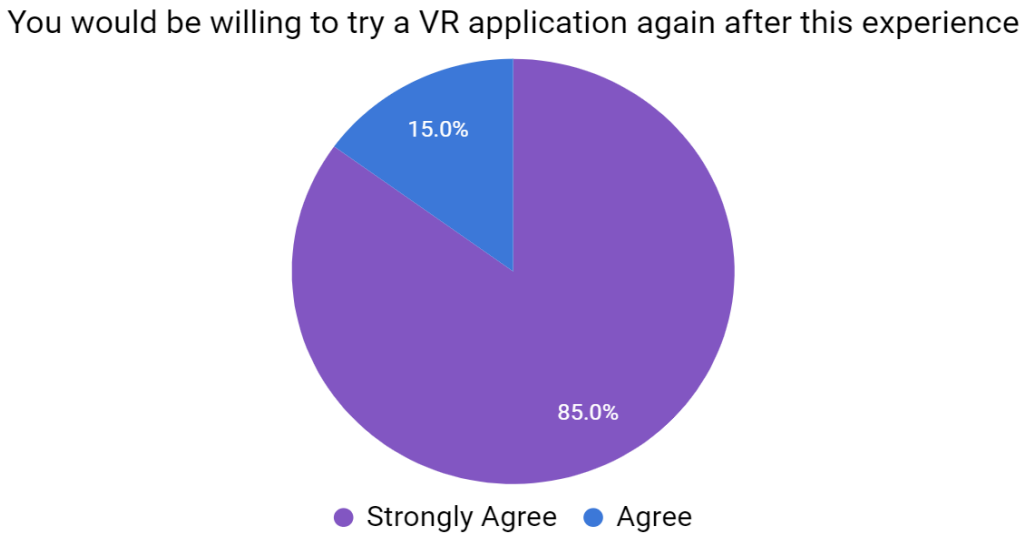


Figure 46: VR experience survey question results

Next, participants were asked about their thoughts regarding the control scheme. The WW team was most concerned about this aspect of the project, as a lot of debate was had over how the controls should be mapped and they were changed many times. We expected that participants would have a lot of trouble with the controls of WW. To our surprise, 75% of participants responded that the controls were intuitive and easy to use, with 20% agreeing and 55% strongly agreeing.

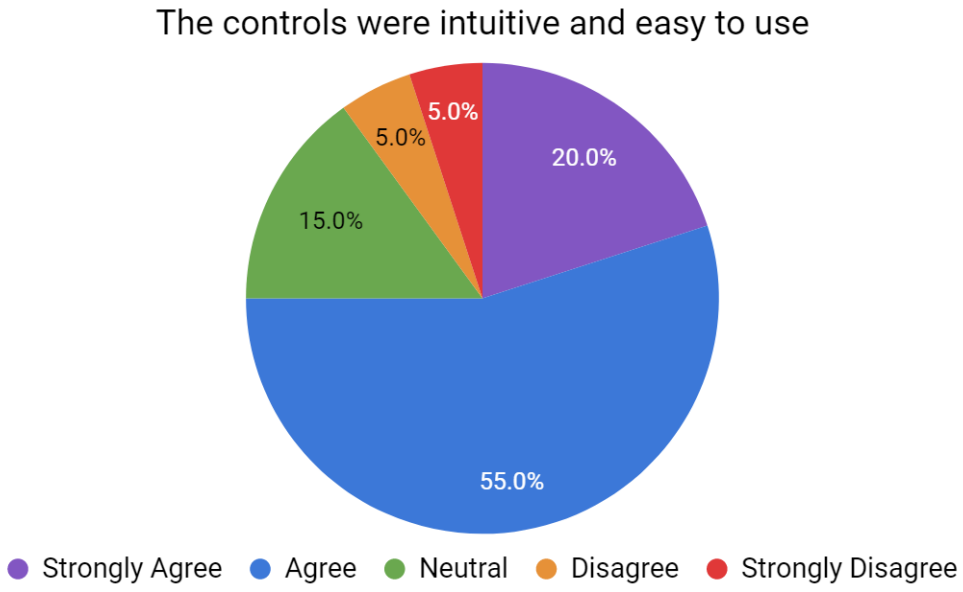


Figure 47: Controls survey question results

The next question asked participants to rank how uncomfortable it was for them to move vertically in WW. During preliminary testing we noticed that many people commented on the vertical movement being nausea-inducing, causing the WW team to make some changes regarding how it worked. We asked this question in order to gauge how well our fixes worked. According to the results, 65% of participants either disagreed or strongly disagreed that moving vertically was uncomfortable. The WW team interpreted the results as a good sign in terms of the vertical movement improvements we made, but that there are most likely further improvements that can make it an even better experience.

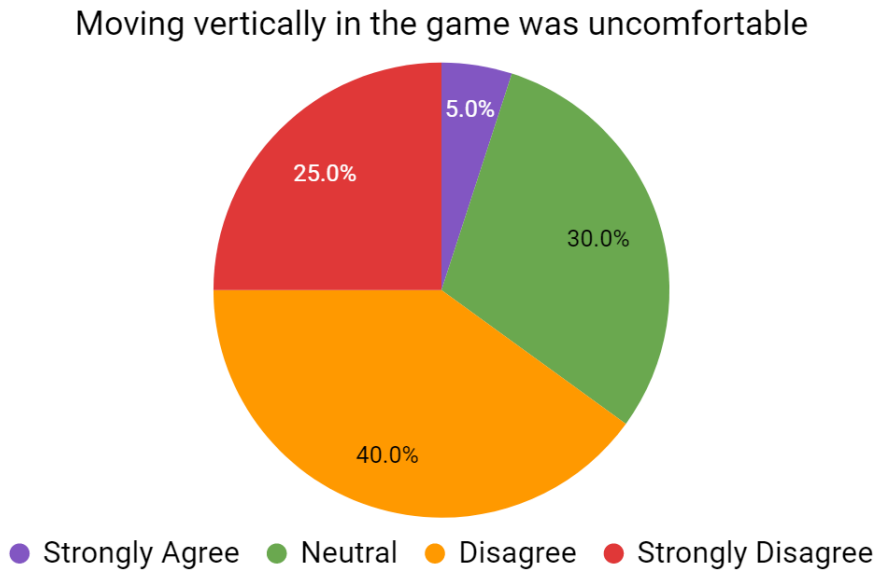


Figure 48: Vertical movement survey question results

There was a slight difference in response to the previous question between self-identified men and women participants. As shown in Figures 48 and 49, men had a tendency to think that moving vertically was slightly more uncomfortable than women thought.

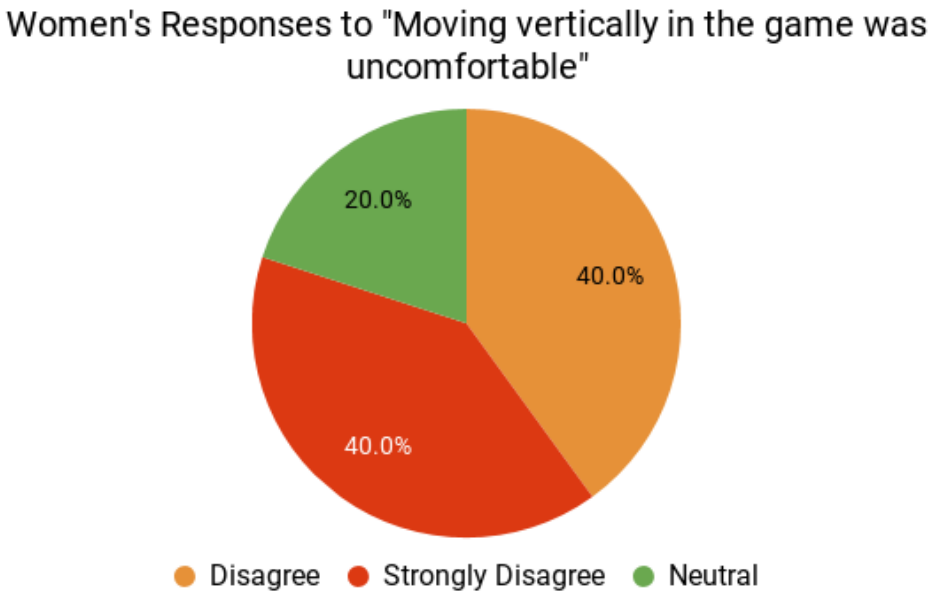


Figure 49: Women participants' response to vertical movement survey question

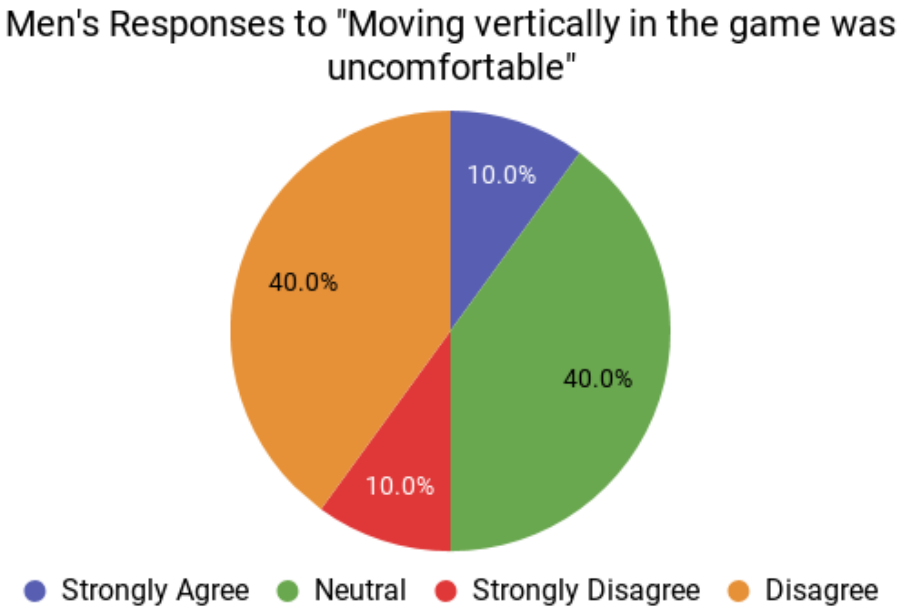


Figure 50: Men participants' response to vertical movement survey question

When compared to number of hours of VR played previously, there was an obvious trend. The more hours of VR the participant had played previous to participating in our user study the more comfortable they were moving vertically. In the chart below, 1 maps to "strongly agree" and 5 maps to "strongly disagree".

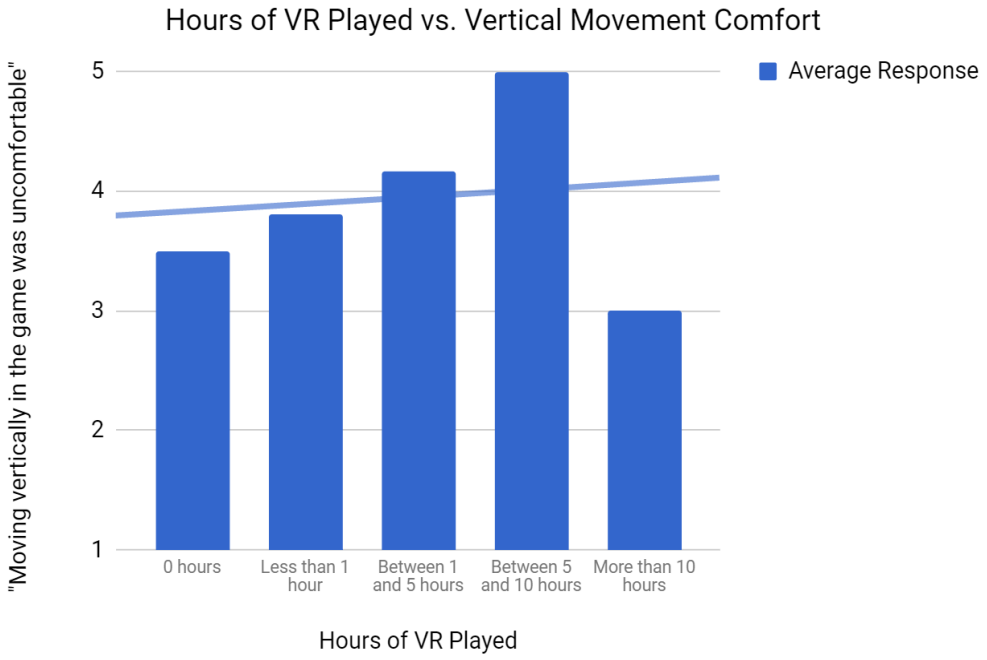


Figure 51: Comparison of hours of VR played and level of comfort moving vertically

We then asked participants to rank whether they thought the tools provided in WW were enough to build a "small game world". This question was asked in order to determine if there was any major functionality we had missed in regards to world building. The results were extremely positive - 95% of participants either agreed or strongly agreed that WW provided enough functionality.

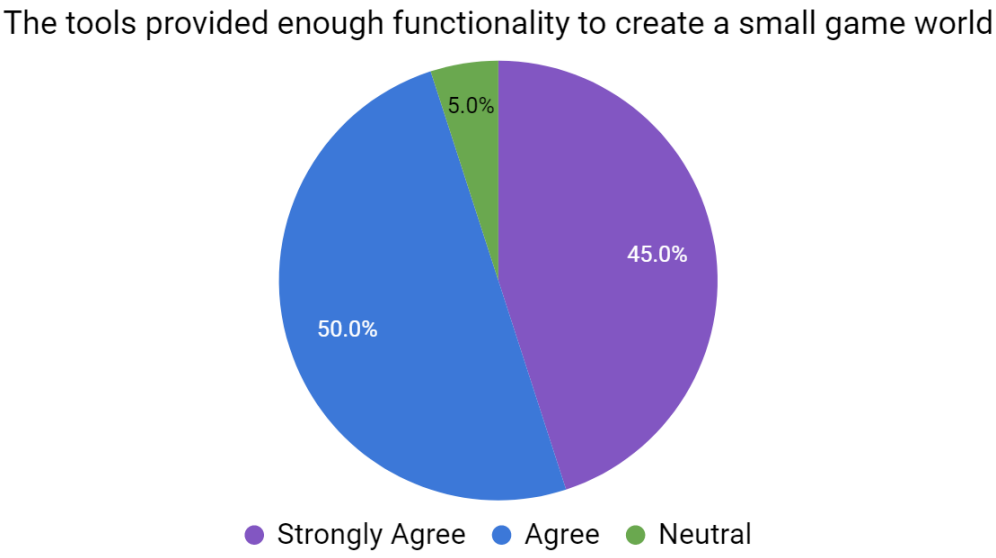


Figure 52: Tool functionality survey question results

The final Likert-scale question asked participants if there were enough tiles and props to build with. The WW team did not have an artist to work with, so we did not have very many tiles and props. We expected that participants would not be satisfied with the number of building blocks. However, based on the results, most participants thought that there were enough tiles to build with. We believe the explanation for this is that given the limited amount of time participants spent playing WW, they didn't have the time to be creative enough to want more to work with.

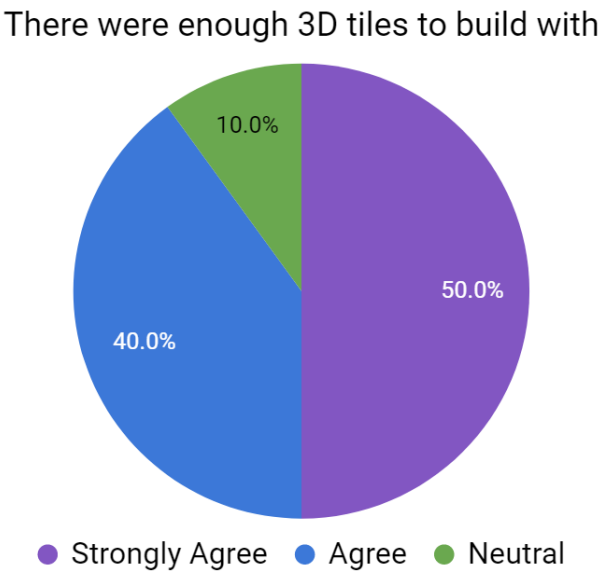


Figure 53: Building survey question results

5.3.3.2 Open Response Questions

After completing the Likert scale questions, participants were asked three open response questions:

1. What was the most frustrating aspect of this test?
2. What was the most satisfying aspect of this test?
3. Do you have any comments or suggestions?

The WW team designed these questions to let us know what participants thought the best and worst parts of the WW application were and to allow participants to give honest feedback anonymously, in case they were not comfortable telling us something directly while they were testing.

The responses to the first question contained about 14 differing opinions, but the most prevalent was that learning the controls was the most frustrating part of the testing. This did not come as a surprise to the WW team, as we realized early on that the controls were difficult for us to map

comfortably to the controllers and we had not implemented any in-game directions. However, these results are in stark contrast to the notes we took on the auditory feedback we got from participants. We attribute this differing of results to the fact that we weren't specific enough when writing the question. We wanted to know if the controls were easy to learn, not if they were easy after already learned - we think participants were answering with respect to how easy the controls were to use *after* they had learned them.

The rest of the responses were split among many different feedback points. Some were frustrated by various bugs in the application, including a bug where selections get changed based on head movement during multiple selection, and another bug where rotating a selection was not centered correctly. A few responses mentioned that it was uncomfortable that when moving vertically you also sometimes moved horizontally at the same time, and that the movement itself was too fast.

The second question responses had clear patterns. Almost half of the responses praised the movement controls in WW, saying that teleporting and "flying" (vertical movement) were the most satisfying aspect of the testing. About a quarter of participants mentioned that they enjoyed the creativity the application allows. As a surprise to the WW team, quite a few participants said that the trials during the testing were the most satisfying, as it gave them a sense of accomplishment. The trials, which included moving to a specific location, moving a cube onto a target point, and recreating a structure, were meant to give the WW team a sense of how well participants were able to learn the controls. Additionally, they seem to have given participants goals that otherwise doesn't exist in WW because it is a sandbox application.

The final open response question, asking for any additional comments and suggestions, gave the WW team a lot of suggestions for future iterations of the game. Many of the responses iterated things that participants had said were frustrating, but did not provide any further detail as to how those issues should be fixed. For example, one participant suggested that the WW team "remap the keys", because they had found the controls unintuitive. Despite the unhelpfulness of some responses, many actually provided interesting ideas for future iterations of WW. The most common suggestions were to move the arm menu buttons to less awkward locations and to change the button mapping, especially for movement.

5.3.3.3 Demographic Questions

The final section of the survey asked participants for demographics information, including past VR experience, gender, age, and major. These questions were asked in order for the team to ensure that we were gathering a range of opinions from different groups of people. Of the people who participated in the user study, the gender split was exactly 50-50, and the previous time spent playing VR was relatively spread out. The age demographic was significantly skewed towards typical college aged people (18-22) because we were only allowed to have WPI students participate. Participants' major was also skewed towards mostly Computer Science and

Interactive Media and Game Development majors, most likely because we advertised our study to students within the departments we were working within.

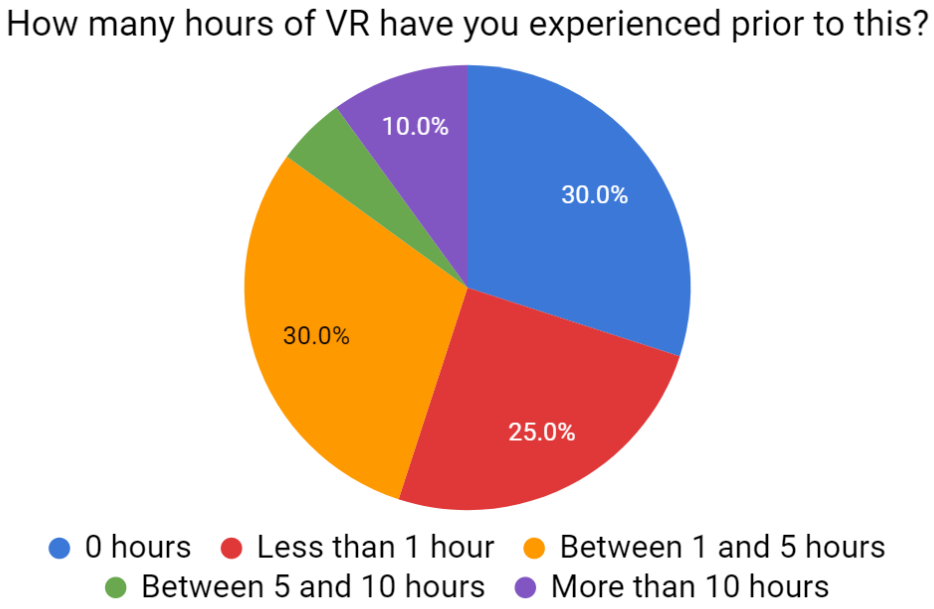


Figure 54: Hours of VR played survey question results

We asked participants to self-identify their gender as part of our demographic section of the survey. The results say that 50% identify as women, 50% identify as men, 0% identify as non-binary, 0% preferred not to say, and 0% chose to write in a different response.

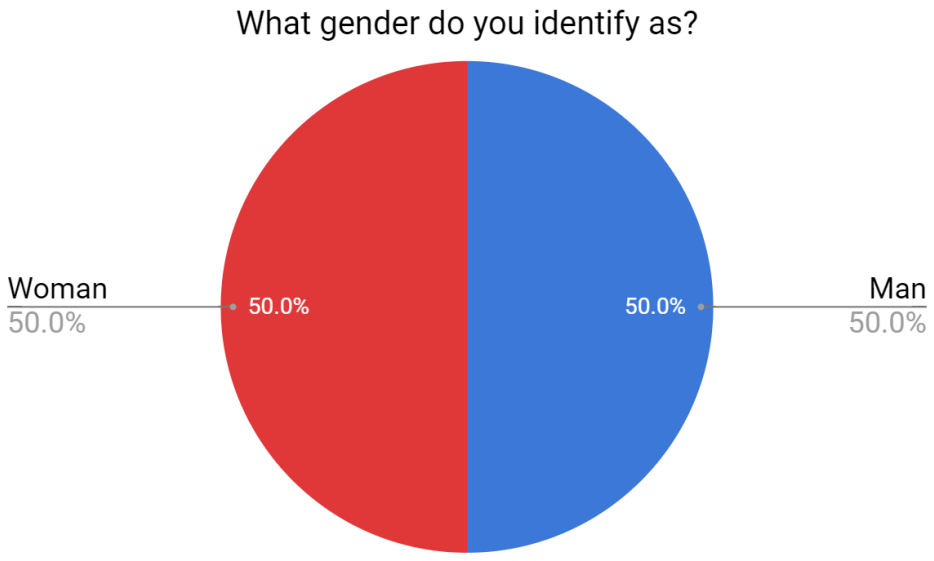


Figure 55: Gender demographic survey question results

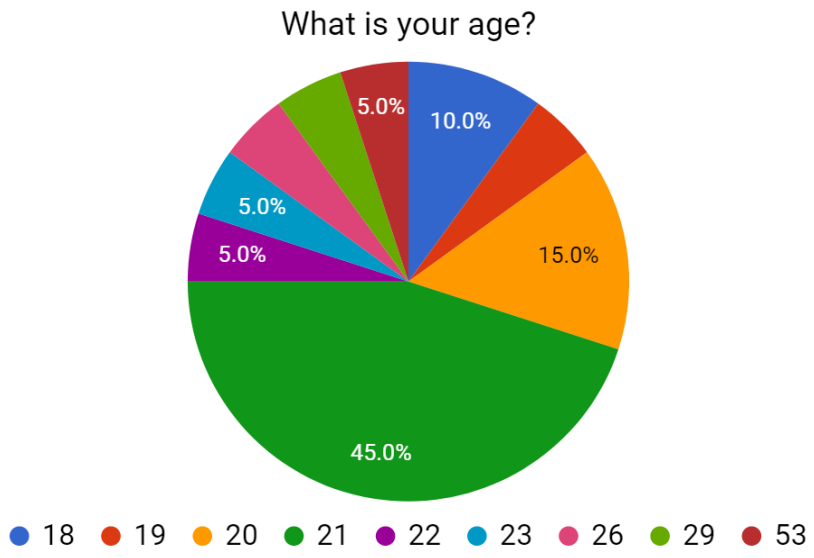


Figure 56: Age demographic survey question results

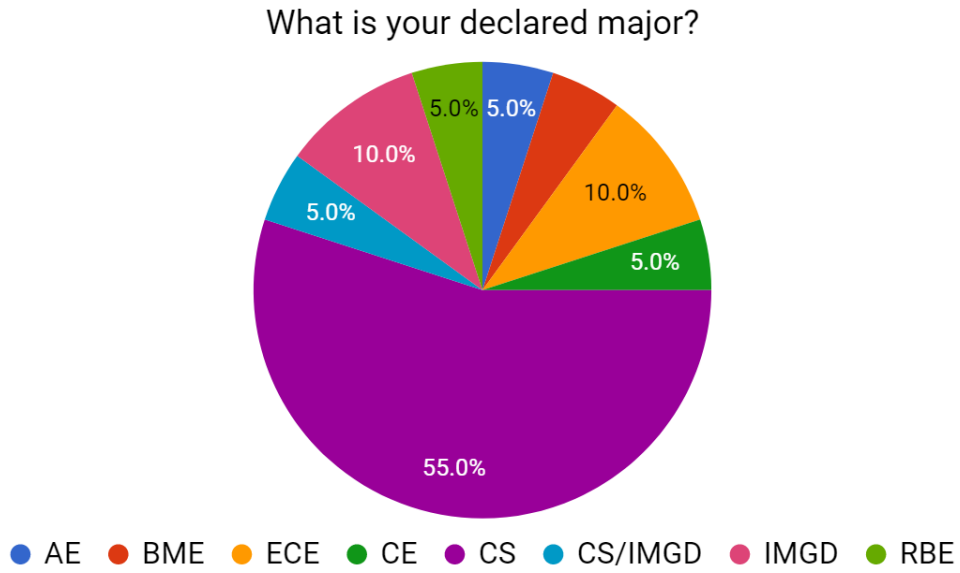


Figure 57: Major demographic survey question results

5.4 Discussion

The data we collected from our usability evaluation led the WW team to discover many different ways that WW can be improved. These changes include both improvements to existing features and suggestions for new features.

5.4.1 Fixes

5.4.1.1 Controls

Much of the feedback we received from our user study was centered on our controls. While some users claimed that they were intuitive after learning them, most had trouble remembering what actions were mapped to each button. A significant restructuring of the control scheme is in order. Less buttons should be used, making the tools more defined and less confusing to learn. In particular, the grip should not be used as much if possible, as it is rather awkward to use. With a more refined arm menu, allowing more tools to be accessible, individual tools could have very specific purposes with less inputs. This would make them less confusing than they are currently, though it might be obnoxious if users need are constantly swapping tools.

5.4.1.2 Movement

During our user study, many players had issues moving up and down. Because the horizontal movement was mapped to touches on the track pad and vertical movement was mapped to the trackpad buttons, many players would move forward when they tried to move upwards or backwards when they tried to move downwards. This problem could be solved by having a slight delay before moving forward or back, by ramping up horizontal velocity over time, or by having another button act as a modifier so that horizontal and vertical movement are not possible at the same time.

5.4.1.3 Trackpad

The issues with movement were present in other systems where touches and presses on the track pad were both mapped. In the future, these two inputs should not both be used by an individual tool. Distinct actions could be mapped to presses in certain regions of the track pad (such as left, right, up, and down) and continuous ranges could be mapped to touches along the axes of the track pad. For example, if rotating a block is done in 90 degree increments, it should be done with presses, but if a block can be rotated freely, it should be done with touching and sliding on the track pad. Existing tools should be redesigned with this is mind.

5.4.1.4 Bugs

Throughout user testing, we came across many bugs in WW that we had previously been unaware of. A full list of all the bugs we found (and the bugs we knew about beforehand) can be found in Appendix G.

5.4.2 Additional Features

5.4.2.1 Collisions

One of the common complaints during our user testing sessions was that the player did not collide with objects. This led to some minor discomfort when passing through objects in some cases. Collision data is already used by the tile system to allow multiple tiles to occupy the same

cell, but it is not considered when the player moves. This change would greatly increase the realism of the virtual worlds and would eventually be required to play the game anyway. Props do not currently have any collision data, but this would also be added as an optional parameter in the meta data. Whether the prop collisions would be based on the geometry or approximated with capsules is up for future debate.

5.4.2.2 Object Menu

One suggestion from a participant was a menu that lets you see each every object at once while you scroll through, in order to make it easier to find what object you're looking for. We think implementing this menu would greatly improve WW, as many participants had a difficult time scrolling through the objects the way it currently works.

6. Post Mortem

Throughout the design and development process, many challenges arose that the WW team had to tackle, and a lot of lessons were learned. WW user testing also showed us what changes should be made to both the design and implementation of WW. This chapter will discuss the main challenges we faced, the lessons we learned, and the proposed changes to WW based on our user testing.

6.1 Challenges

We faced a few challenges during our development of WW. The challenges we faced were both ordinary and expected. As a team, we worked together, overcame, and dealt with the following challenges:

6.1.1 Scoping

The most obvious challenge was dealing with project scoping and feature creep. WW is an ambitious project that has a lot of ground to cover with its various systems. We dealt with the challenge of scope by prioritizing just the level editor and tile system components of WW. We helped deal with this challenge by communicating with our advisors, who are far more experienced with scoping and determining realistic goals. Through their guidance and understanding we were able to prioritize and focus on implementing the core mechanics for an asset pipeline and a working level editor for WW.

6.1.2 Lack Of An Artist

Our team consisted of three technical contributors. WW has a large visual component, but because of our team structure, we did not focus on the visual aspect. One of our team members, Brian, was able to create a set of primitive tiles that were good enough to test out the tile system and asset pipeline, but were not aesthetically outstanding. Due to our lack of art assets, the full impact of the system we are building is difficult to communicate during demos, presentations, and testing.

6.1.3 Source Control Issues

One minor challenge we faced was figuring out what would work for source control. We settled on Git but faced an initial challenge where the metadata files that Unity automatically generates, which contain a GUID for each code file, were not in sync with our local versions of the repository. In other words, while we all had the same codebase, the GUIDs for each code file stored in our local repositories metadata files were different. This caused missing references for components in our Unity scenes. To solve this problem, we started tracking the metadata files

and synced them to the GUIDs that were on one of our machines. Future teams working on this project should not have this issue, as the .gitignore file is setup to track metadata files.

6.2 Lessons Learned

6.2.1 Test Frequently With VR Hardware

We learned a very important lesson in C term right before our first demo at the IMGD Alphafest, an event hosted by the IMGD department where students can user test IMGD MQPs. We learned that when developing an application for VR hardware, it is crucial to frequently test the application on the actual target VR hardware. While we were developing over the course of this school year, we largely developed and tested in the Unity Editor. We had a primitive but working set of desktop controls which we used to quickly and iteratively test out features for WW as we developed them. However, features that worked as expected in the Unity Editor on the desktop did not always work the same when running WW on the HTC Vive.

For example, the marquee selection tool we developed for selecting multiple objects worked perfectly on Desktop. However, in VR the rectangle GUI did not render properly and the position of the corners of the box were not as precise. Another feature that did not work on VR was an outline effect we wanted to use to visually indicate to the user which objects were selected by the marquee. The outline effect we originally implemented utilized Unity's Command Buffer, which allows the rendering pipeline to be extended, was not supported in VR. We realized this only when we went to test out the selection and highlighting of objects in VR.

6.2.2 Communication Is Critical

Like any project that has multiple contributors and several components to work on, communication is critical. We leveraged the communication platform Slack heavily to communicate remotely, and we met in person multiple times a week. The constant stream of communication helped us stay on track with the features we each took ownership for to develop. Our communication as a team was relatively strong and definitely improved over the course of the project as we learned how to communicate more effectively and further developed our team dynamic.

6.2.3 Focus Development Time On Making Impact

Overall, we did not spend or invest much time in developing features that did not work. One decision that we made early on that helped reduce incomplete features from contaminating complete features was to create a directory in our code base for experimental code. This was an area that we could quickly create and try out small bits of functionality or features without worrying about the quality of the code or the performance of the implementation. This experimental area allowed us to quickly prototype features like object placement early on in A

term. Features that looked promising were developed further and brought out of the experimental directory.

6.3 What Did Not Work

6.3.1 Working Too Closely

We created the codebase for WW from scratch. We started with nothing. Because we initially had no pre-existing foundation of code, it did not make sense to immediately divide up work in tasks right away. Instead we spent the first two weeks of A term coding together in person to make sure we were developing the basic architecture in a manner that made sense to all of us. However, once we had a foundation of code, this method soon started to lose its effectiveness. We realized that we needed to start dividing up work into self-contained tasks that would have minimal overlap with each other. It was important that we started being able to work independently to boost productivity, and to develop features that would integrate and mesh nicely with the code being produced as other features were developed. The transition from working together to working more independently took longer than expected but was necessary for our long term development.

6.3.2 Incomplete Features

There are still several incomplete features that either have not been integrated fully or do not have controls to trigger interactions within WW. For example, there is some functioning code for door placement, the automatic generation of walls along the perimeter edge of a group of contiguous tiles, and code for the smart placement of tiles that will automatically rotate a tile into a configuration that fits. Due to time constraints, these feature did not make it into our demo.

6.4 Rationale of Project Decisions

6.4.1 Focusing On Just A Few Core Features

When we planned out the project in our PQP, which was during the final term of our junior year, we scoped out many features at a high level. In hindsight, we were clearly overly ambitious with the features that we scoped out. When we began development of WW, we quickly found it important to focus on creating the pipeline that allowed players to create, share, and import custom art assets into WW. In parallel, we focused on creating the level editor, along with a set of tools a player could use to interact with the level editor. We did not want to spread ourselves too thin during production and recognized those features as the most important to constructing the foundation of WW. We ignored the more complex and secondary features like navigation mesh generation for pathfinding, character creation, terrain creation, in-game scripting,

procedural level generation, and interactable game elements that we had planned out during our PQP.

6.4.2 Demo On Hardware That Has Been Tested

One decision that benefited us immensely was our decision to use our own hardware, a laptop, when we demonstrated the application at the IMGD Alphafest. We wanted to control every variable possible to ensure that our demo went according to plan. Because of our preparation, testing, and our decision to use our own hardware, we were the only VR MQP that was successful in demoing at the IMGD Alphafest.

6.5 Future Work

In addition to the fixes and suggestions based on our user testing results in section (section number), the WW team came up with more ideas for how we think WW could be improved in the future.

6.5.1 Additional Features

6.5.1.1 Interactables

From the start, we planned on having `Interactables` as the third type of object alongside `Tiles` and `Props`. The list of `Interactables` would include levers, doors, destructible objects, chests, portals, and invisible triggers. In addition to the current process for adding custom assets, users would also need to determine what the result of an interaction would be. Custom behavior would be controlled with a scripting system that will be discussed in a later section. As far as placement goes, `Interactables` would act much like `Props`. They would not be restricted to the grid like `Tiles`. Multiple interaction types could be defined, such as `onDestruction`, `onCollision`, or `onButtonPress`.

6.5.1.2 Custom AI

The worlds created in our application feel rather empty at the current time. Players can create rather elaborate environments, but have no animals or people to fill them. Actors would be another type of asset alongside `Props`, `Tiles`, and `Interactables`, and they would have a preset behavior or a custom behavior script attached to them. Along with this, the ability to map out specific areas for use by the AI would most likely need to be added. Without any kind of mapping, it would be impossible to create realistic behavior aside from random wandering. The AI system would also eventually utilize the scripting system which will be discussed in a later section.

6.5.1.3 Scripting

For things such as interactable objects, scripted events, cinematics, or custom enemy AI, some scripting system would be required. Scripts could be attached to an object's meta data and be consulted when, for instance, a player walks over a trigger. Eventually, we would like to have the ability to apply scripts to entire levels, allowing for things such as weather systems or timed gameplay events. Scripting would allow players to create immersive, reactive environments with fully customized behaviors. It would also allow for incredible customization of gameplay, storytelling, and environments with endless possibilities.

6.6 Conclusion

We faced constant challenges of scope but were successful in focusing on architecting the core world building systems and art asset pipeline for WW. We learned that working too closely as a team did not work for long term productivity and we were able to correct this by dividing and conquering the work. While we developed a few incomplete features, like smart tile placement, which did not make it into the final build of WW, we were able to avoid incurring design debt. The code we produced for the main package is well documented and should be extendable by future MQP teams and other contributors as features like Interactables, Custom AI, and Scripting are implemented.

References

1. Aaron Souppouris. 2016. How HTC and Valve built the Vive. (March 18, 2016). Engadget. Retrieved April 24, 2018 from https://www.engadget.com/2016/03/18/htc-vive-an-oral-history/.

2. Andrés Torres, Telmo Zarraonandia, Paloma Díaz, and Ignacio Aedo. A Comparative Study of Menus in Virtual Reality Environments. (2017), 294-299.

3. Atman Binstock. 2015. Powering the Rift. Oculus.com. (May 15, 2015). Retrieved March 28, 2018 from https://www.oculus.com/en-us/blog/powering-the-rift/

4. Ben Lang. Latest HTC Vives Are Shipping with Tweaked Base Stations, Redesigned Packaging – Road to VR. Road to VR. (April 13, 2017). Retrieved March 28, 2018 from https://www.roadtovr.com/latest-vive-shipping-with-tweaked-base-stations-redesigned-packaging/

5. Chris Baker. 2017. Nintendo Flashback: The Disastrous Power Glove. (March 3, 2017). Rolling Stone. Retrieved April 24, 2018 from https://www.rollingstone.com/glixel/news/nintendo-flashback-the-disastrous-power-glove-w470178.

6. Chuck Martin. 2018. HTC Vive Leads VR Market, Oculus Rift Gains In Popularity. (January 2, 2018). Retrieved April 24, 2018 from https://www.mediapost.com/publications/article/312368/htc-vive-leads-vr-market-oculus-rift-gains-in-pop.html

7. Electronic Arts. 1986. Commodore-Amiga Adventure Construction Set. (1986). Retrieved April 24, 2018 from mocagh.org/ea/acsuk-refcard.pdf

8. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1994. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, Reading, Mass.

9. Evan Lahti. 2016. Valve puts in $1 million for all future major CS:GO tournaments. PC Gamer. (February 23, 2016). Retrieved April 24, 2018 from http://www.pcgamer.com/valve-all-major-csgo-tournaments-will-be-1-million/

10. HTC Corporation. 2018. Vive VR System. (2018). Retrieved April 24, 2018 from https://www.vive.com/us/product/vive-virtual-reality-system/.

11. Joe Donnelly. 2017. Best total conversion mods. Rock, Paper, Shotgun. (February 2, 2017). Retrieved April 24, 2018 from https://www.rockpapershotgun.com/2017/02/10/best-total-conversion-mods/

12. John Funk. 2013. MOBA, DOTA, ARTS: A brief introduction to gaming's biggest, most impenetrable genre. Polygon. (September 2, 2013). Retrieved April 24, 2018 from http://www.polygon.com/2013/9/2/4672920/moba-dota-arts-a-brief-introduction-to-gamings-biggest-most

13. Jon Brodkin. 2013. How Unity3D Became a Game-Development Beast. Dice Insights. (June 3, 2013). Retrieved March 28, 2018 from http://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/

14. Lawrence Hettinger and Gary Riccio. Visually Induced Motion Sickness in Virtual Environments. Presence: Teleoperators and Virtual Environments 1, 3 (1992), 306-310.

15. Michael Walbridge. 2018. Analysis: Defense of the Ancients - An Underground Revolution. Gamasutra. (June 12, 2018). Retrieved April 24, 2018 from https://www.gamasutra.com/view/news/109814/Analysis_Defense_of_the_Ancients__An_Underground_Revolution.php

16. Mike Stubbs. 2017. Dota 2's $100 million milestone, visualised. Redbull.com. (June 23, 2017). Retrieved April 24, 2018 from https://www.redbull.com/se-en/dota-2-100-million-milestone-visualised

17. Modbox. 2016. Alientrap. PC. (April 5, 2016). Retrieved April 24, 2018 from http://store.steampowered.com/app/414120/Modbox/.

18. Mojang. 2017. Minecraft End User License Agreement. (2017). Retrieved April 24, 2018 from https://account.mojang.com/documents/minecraft_eula.

19. Mojang. 2018. Explore Minecraft in VR. (2018). Retrieved April 24, 2018 from https://minecraft.net/en-us/vr/.

20. Nikolai Bockholt. 2017. VR, AR, MR and What Does Immersion Actually Mean?. Google. (2017). Retrieved April 24, 2018 from https://www.thinkwithgoogle.com/intl/en-cee/insights-trends/industry-perspectives/vr-ar-mr-and-what-does-immersion-actually-mean/

21. Richard Trenholm. 2015. The history of Samsung's Gear VR virtual reality headset. (2015). CNET. Retrieved April 24, 2018 from https://www.cnet.com/news/the-history-of-samsungs-gear-vr-virtual-reality-headset/.

22. Rift Info. 2015. Oculus Rift History - How it All Started. (2015). Retrieved April 24, 2018 from https://riftinfo.com/oculus-rift-history-how-it-all-started.

23. Samuel Horti. 2018. Minecraft had 74 million active players in December, a new record for the game. (January 21, 2018). PC Gamer. Retrieved April 24, 2018 from https://www.pcgamer.com/minecraft-had-74-million-active-players-in-december-a-new-record-for-the-game/.

24. Sibaem. 2017. The Sims 1: Gameplay #1 (No Commentary). (2017). YouTube. Retrieved April 24, 2018 from https://youtu.be/gi0AB5ksxuY.

25. Sophie Charara. 2015. Virtual reality: Then and now - why it won't fail this time. Wareable. (July 14, 2015). Retrieved April 24, 2018 from https://www.wareable.com/vr/virtual-reality-then-now-why-it-wont-fail-this-time

26. Southern California Earthquake Center. 2018. How Do I See Depth?. (2018). Retrieved April 24, 2018 from http://scecinfo.usc.edu/geowall/stereohow.html.

27. Steam. 2018. Virtual Reality on Steam. (2018). Retrieved April 24, 2018 from http://store.steampowered.com/vr.

28. Strong, The. 2018. View-Master. (2018). Retrieved April 24, 2018 from http://www.toyhalloffame.org/toys/view-master.

29. Tilt Brush. 2018. Google. PC. (2018). Retrieved April 24, 2018 from https://www.tiltbrush.com/.

30. Tiny Town VR. 2017. Lumbernauts. PC. (2018). Retrieved April 24, 2018 from http://store.steampowered.com/app/653930/Tiny_Town_VR/.

31. Tom Hall Press, Inc, The. 1998. Doom Bible Appendices. (1998). Retrieved April 24, 2018 from http://5years.doomworld.com/doombible/appendices.shtml

32. Turing Institute, The. 1996. History of Stereo Photography. (1996). Retrieved April 24, 2018 from http://www.arts.rpi.edu/~ruiz/stereo_history/text/historystereog.html.

33. Unity. 2016. New Unity products and prices launching soon. (May 31, 2016). Retrieved March 28, 2018 from https://blogs.unity3d.com/2016/05/31/new-products-and-prices/

34. Unity. 2018. Unity - User Interfaces for VR. (2018). Retrieved April 24, 2018 from https://unity3d.com/learn/tutorials/topics/virtual-reality/user-interfaces-vr.

35. Verge, The. 2015. Valve's VR headset is called the Vive and it's made by HTC. (March 1, 2015). Retrieved March 28, 2018 from https://www.theverge.com/2015/3/1/8127445/htc-vive-valve-vr-headset

36. Virtual Reality Society. 2017. History Of Virtual Reality. (2017). Retrieved April 24, 2018 from https://www.vrs.org.uk/virtual-reality/history.html.

37. Virtual Reality Society. 2017. VPL Research Jaron Lanier. (2017). Retrieved April 24, 2018 from https://www.vrs.org.uk/virtual-reality-profiles/vpl-research.html.

38. Virtual Reality Society. 2018. What is Virtual Reality?. (2018). Retrieved April 24, 2018 from https://www.vrs.org.uk/virtual-reality/what-is-virtual-reality.html.

39. Yaphets. 2018. A History of Dota: Part 1. Facebook. Retrieved April 24, 2018 from https://www.facebook.com/notes/yaphets-pis/a-history-of-dota-part-1/377203832338260.

Appendix A: Original WW Design Document

Abstract / Idea High Level

We are creating a World Builder for the World Wizards project. The World Builder will support users creating levels from tilesets and also allow the placement of props and interactable objects in the scene. The World Wizards Level Builder (WWLB) will be developed for Virtual Reality (VR) platforms, specifically the HTC Vive. WWLB will be developed largely using the Unity Game Engine.

Goals

1. Intuitive tile and prop placement, leveraging the power of VR.
2. Multi-level support with Portals to traverse them.
3. Fully fledged and sophisticated, yet simple, level editor.
4. Import and setup custom tiles and props using Unity Editor extension and then export as asset bundles to be used in the World Wizards Application.
5. Place interactable objects in the world such as doors, levers, traps.
6. Gesture-based object transformations and controls.
7. Group objects together for multi-object editing capabilities.
8. Portable file system to save levels and user setting. Use a common format like JSON.

Controls

Menu Traversal:

Large Cinema Menus:

Move your head to look at what you want to select, look long enough to select it. Swipe to scroll menu to see other options.

Radial Menu:

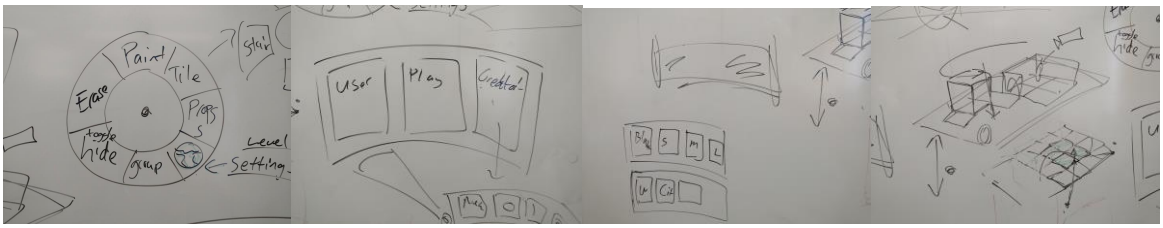
Workbench Dome: Rather than an actual menu, a portable workbench that follows the player around, allows the player easy access and placement of all possible tools.

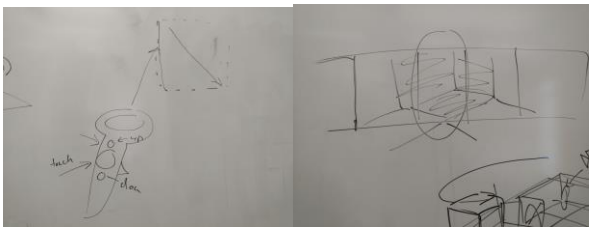
Tile Editing:

Gestures for rotation, scaling, and translating.

Smart placement of tiles use contextual filters, and volumetric cognisance (only placing props and tiles that fit in the available space).

UI / Interface Mockups





Camera Modes

First Person Player Perspective:

Allows the user to drop into the world and see it up close and personal. Useful for testing things from the player's perspective and getting a true sense of scale and proportion or fine tuning prop and interactable placement. Users can toggle between modes to either fly or walk through the level.

Workbench Perspective:

The world sits upon a table in the center of the user's VR space. The player can walk around the table and edit the level as they see fit from this higher level perspective. Useful for high-level level design and sweeping changes.

Architecture

See github: https://github.com/brianjkd/worldWizardsDesign

The Core World Wizards Project contains three packages:

1. The **entity package** contains data types and structures for World Wizards.

Package	Type	Name	Description
Common/User	class	UserProfile	Contains a user's general global settings as well as their different projects (worlds)
Common/User	class	UserSettings	Contains a user's global configuration settings

Common	enum	InteractionType	The types of interactions (trigger, input, event, etc)
Common	class	IntVector3	A specific type of Vector3 which contains 3 integers rather than floats
Common	class	MetaData	Contains any metadata associated with a WWObject (environment group tags, etc)
Common	enum	WorldWizardsMode	The mode that the application is currently in (Builder, Player)
Common	enum	WorldWizardsType	The WWObject types (Tile, Prop, Interactable)
Coordinate/Utils	class	CoordinateHelper	A helper class for conversions between WW's Coordinate System and Unity's System
Coordinate	class	Coordinate	The WW coordinate type. It contains an IntVector3 for the tile index and a Vector3 for the offset within that tile
GameObject	class	Interactable	An interactable WWObject
GameObject	class	Prop	A WWObject that is placable within a tile
GameObject	class	Tile	WWObject which takes up one coordinate box and acts as a base
GameObject	class	WorldWizardsObject	Base class for all WWObjects. Inherits from Monobehavior so it acts as the connection to Unity's gameobject system
Level	class	Level	Contains the level settings and the scene graph associated with it
Level	class	LevelSettings	Holds a number of settings specific to a level such as global lighting settings

Level	class	SceneGraph	Data structure which holds all of the WWObjects in the current level
World	class	World	Contains the world settings and a list of levels associated with it
World	class	WorldSettings	Holds a number of settings specific to the world
World	class	EntityPackageInfo	Contains the data types and data structures for the WW core project

2. The **controller package** contains the manager/ controller classes which exist in the Unity scenes. The controllers will be accessed through the user controls and menus to manipulate the entity objects.

Package	**Type**	**Name**	**Description**
Builder	class	ManipulateObjectController	Handles translation, rotation, and scaling events for WWObjects
Builder	class	ToolController	Manages the work space of tools that the player can interact with, like the grouping tool, eraser, paint prop brush, etc.
Camera	class	CameraController	Controls the camera
Camera	enum	CameraMode	The different camera modes (player, free, overview)
Common	class	WorldWizardsMasterSingleton	Singleton class that persists for the duration of the entire World Wizards application, and manages overall

			application settings
Level/Utils	class	WorldWizardsObjectFactory	Responsible for instantiating gameobjects into the unity environment
Level	class	LevelController	Loads and saves levels
Player	class	PlayerController	Controls player movement through level
User	class	UserController	Contains user profile information
World	class	WorldController	Contains reference to the world and loads it

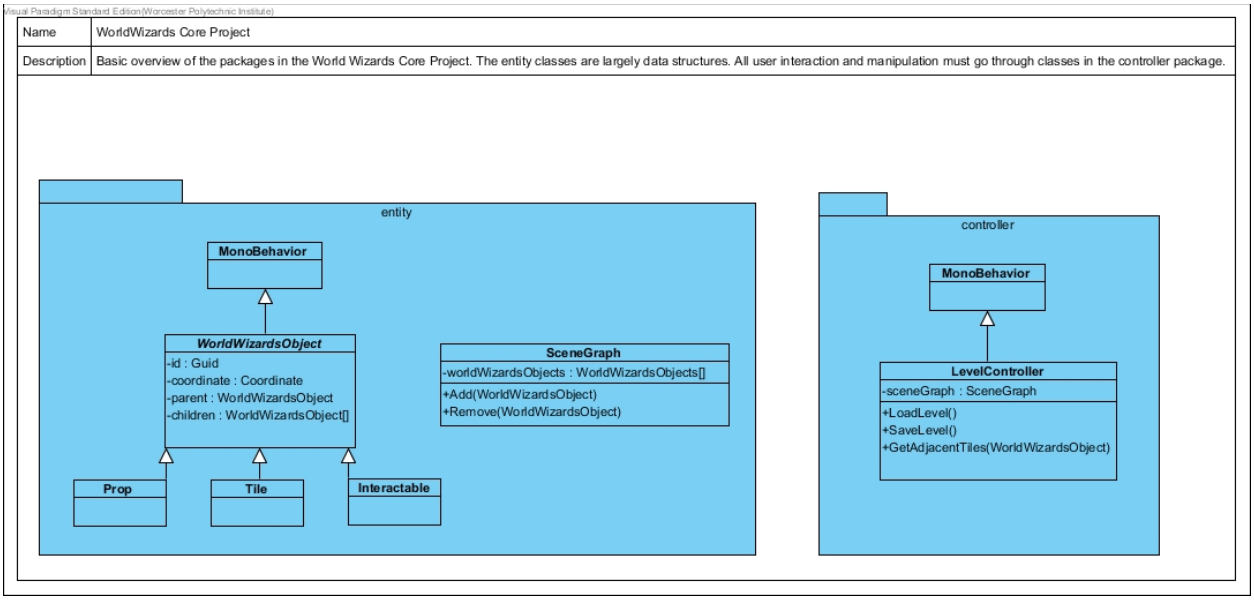
3. The **user package** will be responsible for handling the UI menus and user controls and interactions and interface to the controller objects.

Type	**Name**	**Description**
class	BuilderMenu	Rotary menu containing options related to world building such as object placement
class	LevelSelectorMenu	Once within a world project, this menu contains the world's levels along with the option to add additional levels
class	MainMenu	Contains options to select a project to edit, switch user profiles, or change user settings
class	UserPackageInfo	Contains all the UI and user controls

Control Flow:

User interacts with UI and other controls. These interactions interface with the controller objects which mutate the World Wizards data and data structures.

SceneGraph:
The scene graph will be a flat data structure implemented with a hashmap where the key is a Guid and the value is the WorldWizardsObject. Each implementation of a WorldWizardsObject will have a bidirectional connection to a parent and a list of children. When a child is removed, it must first be removed from the parent's list of children, and then deleted to ensure no dangling references.



Coding Standards

Capitalize Method Names (PascalCase)

Capitalize Class Names

100 character line width limit

camelCasing

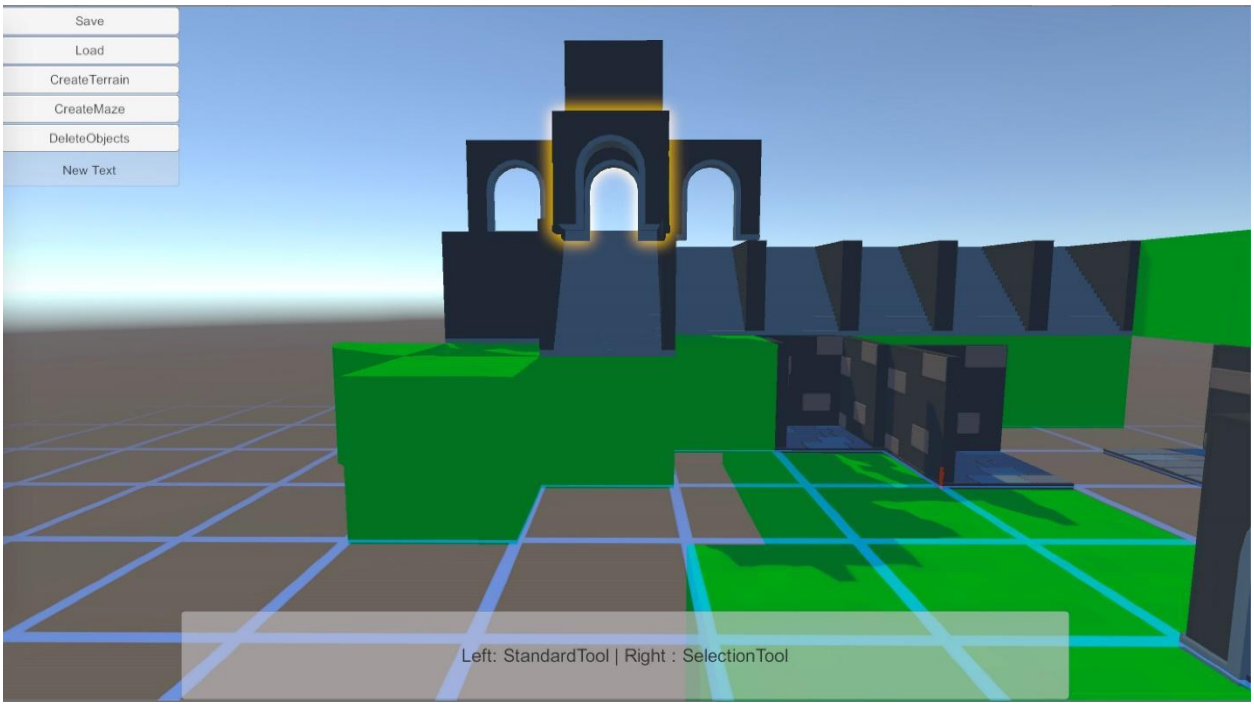
Curly Brackets on the next line

All expressions following an if or else statement must be wrapped in { }

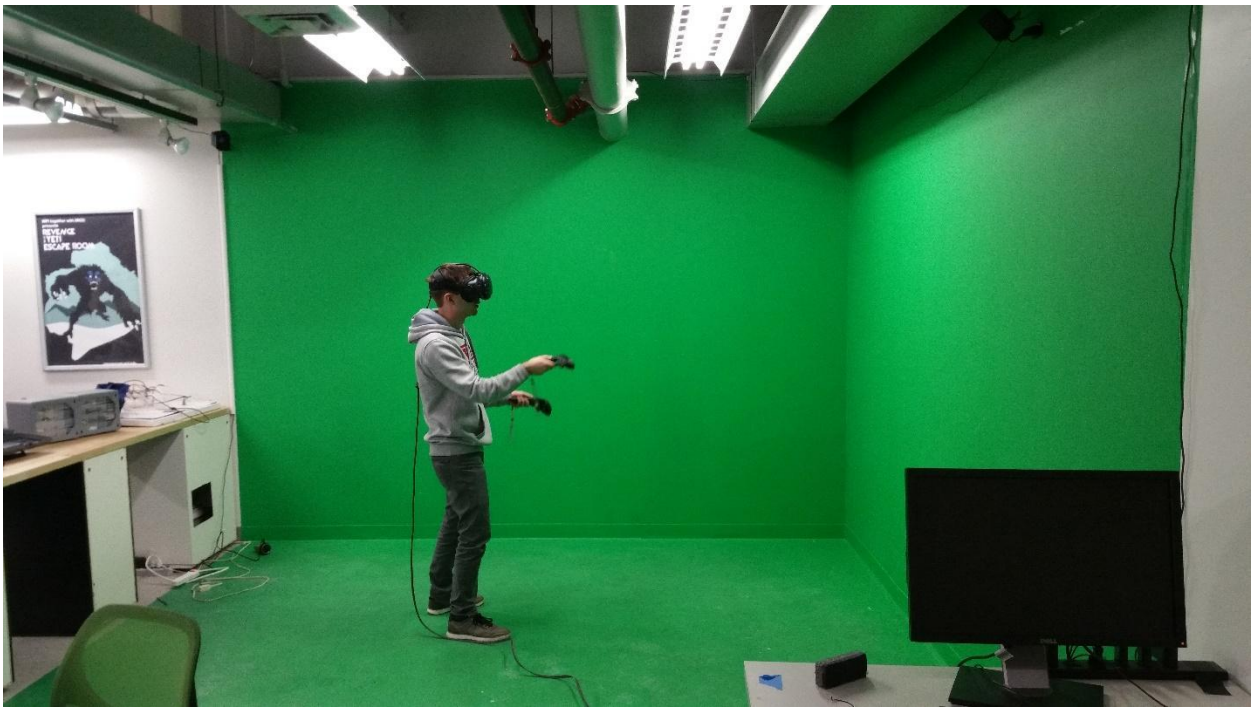
Appendix B: WW Description/User Study Methodology

General Game Description

World Wizards is a virtual reality (VR) game where players create their own virtual world. Players are able to place and interact with objects in the virtual world, and move around in the virtual world. During gameplay, players walk around a confined space (about 10ft x 10ft) in the real world with a headset on and use two controllers (one in each hand) to play.



World Wizards gameplay from the player's perspective



World Wizards gameplay from an observer's perspective. (Image used with permission from person shown)

Virtual Reality Components

World Wizards uses an HTC Vive controller. The HTC Vive consists of a headset, two controllers, and two sensors.

HTC Vive handheld controllers (front left and front right), sensors (back left and back right) and headset (middle)

The HTC Vive requires a real world space of about 10ft x 10ft. The sensors are set up in two opposite corners of this space and allow the game to track the locations of the headset and controllers.

The room where the study will take place. The area allowed for movement during gameplay is green. The HTC Vive sensors are circled in red.

The headset is placed on the player's head and covers their eyes. The player's eyes are facing a screen inside the headset. The headset is connected to a computer through a cord that runs down the player's back. The headset is what allows the player to see what they are doing in the virtual world they are interacting with during gameplay.

A person wearing an HTC Vive headset and holding the HTC Vive controllers. The headset's cord runs down the person's back. (Image used with permission from person shown)

The player holds two controllers, one in each hand, during gameplay. Interacting with the objects in the game requires players to push the buttons on the controllers and move the controllers around real world space.

Methodology

During our study, we will follow this guideline with every participant:

1. Preparation
    a. Explain the goals of the study to the participant
    b. Explain all potential hazards to the participant
    c. Go over consent form with the participant and get participant's signature
    d. Explain to the participant the virtual reality setup
    e. Explain the controls for the game to the participant
    f. Put the VR headset on the participant and give them the VR controllers
2. Gameplay
    a. Ask participant to play the game and explain what they are thinking as they play
        i. One MQP group member will be taking notes
    b. Should the participant be confused or unsure of what to do, we will provide tasks for them to attempt to complete such as:
        i. Place an object in the virtual world
        ii. Move an object that already exists
        iii. Rotate an object that already exists
        iv. Move around in the virtual world (through either real world walking or in-game teleportation)

The study will take a maximum of 30 minutes for each participant. Participants are able to ask questions and voice their concerns or opinions at any point in time.

Potential Hazards

Potential hazards during gameplay include nausea, dizziness, and falling. Virtual reality games can make players motion sick and/or dizzy. A player's dizziness may cause the player to fall. Falling may also be caused by the player tripping on the cord behind the player that is attached to the headset.

We are going to be very clear with our participants about the safety hazards and will do all we can to prevent them. We will ask participants in our study to tell us if they feel sick and/or dizzy and allow them to leave the study at any point in time. There will also be an additional person (outside of the MQP group) shadowing each participant throughout the entire study to ensure that

none of the participants fall. Shadowing will consist of following the participant, watching them closely, guiding them away from hazards, and providing support should they start to fall or feel dizzy.

Appendix C: User Study Survey

1. You felt nauseous at times during the test.
    a. Strongly Agree
    b. Agree
    c. Neutral
    d. Disagree
    e. Strongly Disagree
2. You would be willing to try a VR application again after this experience.
    a. Strongly Agree
    b. Agree
    c. Neutral
    d. Disagree
    e. Strongly Disagree
3. The controls were intuitive and easy to use.
    a. Strongly Agree
    b. Agree
    c. Neutral
    d. Disagree
    e. Strongly Disagree
4. Moving vertically in the game was comfortable.
    a. Strongly Agree
    b. Agree
    c. Neutral
    d. Disagree
    e. Strongly Disagree
5. The tools provided enough functionality to create a small game world.
    a. Strongly Agree
    b. Agree
    c. Neutral
    d. Disagree
    e. Strongly Disagree
6. There were enough 3D Tiles to build with.
    a. Strongly Agree
    b. Agree
    c. Neutral
    d. Disagree
    e. Strongly Disagree
7. What was the most frustrating aspect of this test?
    a. Open response

8. What was the most satisfying aspect of this test?
    a. Open response
9. Do you have any comments or suggestions?
    a. Open response
10. How many hours of VR have you experienced prior to this?
    a. 0 hours
    b. Less than 1 hour
    c. Between 1 and 5 hours
    d. Between 5 and 10 hours
    e. More than 10 hours
11. What gender do you identify as?
    a. Female
    b. Male
    c. Nonbinary
    d. Prefer not to say
    e. Other (fill in)
12. What is your age?
    a. Short answer
13. What is your declared major?
    a. Short answer

Appendix D: User Study Notes

Participant #1:

- "Point with the right controller and pull the left trigger?"
- While cycling objects, couldn't figure out where the object was because they weren't pointing the controller at a surface.
- Hard to differentiate between clicking and touching to change objects (accidentally moved grid and rotated object instead of changing object)
- Should probably have a list thing showing what objects are being cycled through (like little pictures that show objects)
- Confusing that changing the tool was done by physically hitting the button (laser was confusing)
- Forgot filtering was on, affected selection
- "So there's no copy and paste?"
- Lower arm menu

Participant #2:

- "That's cool." referring to teleporting
- Has used VR before
- "That's easy enough" referring to selecting and moving objects

Participant #3:

- Generally this user completed tasks at a good rate. Was comfortable using the controllers. Was able to rotate objects after placing them with the edit object tool

Participant #4:

- "Teleported" object that was selected by clicking in a totally different spot, confusing because it seemed to disappear by moving really far away
- When trying to place a rock, it was colliding with stuff under the grid and trying to place on top of that stuff. Grid collider should take precedence
- Some issues deleting/selecting (issue with some objects not being tracked properly)
- Got slightly nauseous at the end

Participant #5:

- "Is there any way I can cancel the laser?" (teleportation)
- Changed objects by accident sometimes
- Issue with props not being constrained to grid, confusing
- Issue rotating, object flew away because of the center not being calculated properly
- "Weird when I wanna go up but it's moving me forward, it's throwing me off"
- Nothing is deleting?

    - Objects that are part of the loaded scene cannot be deleted
- Laser pointer on right controller is confusing with arm menu
- “Only weirdness I felt was moving forward and going up at the same time”

Participant #6:

- Rotation would not work when pressing the right button
    - Corrected, was not pressing close enough to the right edge of the button
- Tried deleting a preview object but could not
    - Explained that can only delete objects that have been placed
- Learned on their own how to move objects before placing by holding down trigger before releasing
- Expressed satisfaction using the physical arm menu
- Experimented with moving selection up and down and rotating

Participant #7:

- Moved grid by accident
- A little bit nauseating to look down while moving up

Participant #8:

- Started with object placement
- Moving up and down and vertical felt bad
- Moving via the laser/ teleport “feels a lot better”
- The z fighting looks weird when objects are overlapping
- Scrolls through objects where you point
- Menu vs scrolling through objects
    - Menu would be preferred
- Movement feels weird
    - Slightly blurry but mostly fine
- Enjoys destroying objects

Participant #9:

- Tried to teleport to sun
- Seemed a little bit dizzy, almost lost footing
- Grid got stuck when it moved too high, came back when they teleported
- Doors are still invincible
- Hit the hide menu button instead of changing tool
- Up and down movement is the most nauseating
- Hit arm when using arm menu

- Trouble figuring out whether an object was just a preview or actually placed, thought an object was placed when it wasn't
- Moving head made marquee selection not work right

Participant #10:

- Might feel more comfortable with movement being controlled by your right hand
- Really enjoys moving
- Pushing down on center button feels like you're pushing sometimes even when you're not, makes it seem like you should be moving down and you don't
- Preview object is confusing, might need to make it slightly transparent or something so people know it's not actually placed yet (leads to issues with scene graph as well)
- Flying above objects makes it easier to build/manipulate
- Object editing is "pretty intuitive"
- Controls seemed to be difficult to remember, except for movement
- "Not too disorienting"

Participant #11:

- Has played our application before at Alphafest
- "Kind of enthralling"
- "Definitely a learning curve"
- "Wouldn't call the controls intuitive", but picked them up after a while

Participant #12:

- Not dizzy when moving up and down
- Tried to grab object without selecting it first
- Thought the preview object had been placed

Participant #13:

- A little weird that you move forward when you want to move up
- Couldn't select an object very easily
- When attempting to move the construction plane up, cycled objects

Participant #14:

- Played during Alphafest
- "A little awkward that you can't go up or down the stairs"
- Flying feels better than at Alphafest (had issues with flying making them nauseous before)
- Had issue with arm menu rotating with controller rotation (maybe have the arm menu always face the headset)
- Confusing that the laser on the right controller causes the arm menu buttons to highlight

- Feels better to hold down trigger and cycle through objects
- "Definitely better than last time"

Participant #15:

- Up and down movement more comfortable not inverted
- Accidentally hit both arm menu buttons at the same time
- Issue cycling through objects without holding trigger (holding trigger and cycling worked fine)
- "Much more convenient" to use teleporting as movement
- "Having the time of my life"
- Grid system is convenient, makes everything line up nicely

Participant #16:

- "A little tricky at first, but it's relatively intuitive"
- Menu button placement is a little low, punched her own arm
- Moved construction plane before trying to place the floor tile during the 3rd trial

Participant #17:

- Tilted her head to get a better view of the 3rd trial
- Had some trouble moving down

Participant #18:

- "Freaking cool dude"
- Feels powerful
- Moving forward while moving up felt weird
- "Woahhh" (in reference to z fighting)
- "Very satisfied"
- Confusing that the blue laser highlights the arm menu buttons
- The only person to move their position while also moving an object
- Made menu switch to prop filtering by accident
- Most creative person we've had
- "Bro, I feel like I have too much power right now"
- "This is insane" (in reference to the trees conforming to the ground below them)
- Kind of hard to cycle, when trying to swipe instead of touch
- Moving grid up and down cycles objects (because touch cycles objects)
- "The controls were actually pretty intuitive. The only thing that was confusing was tapping to change objects"

Participant #19:

- Felt more comfortable changing tiles while holding the trigger

- Doesn’t like that previews cannot be deleted
- “I like that you added the sun”
- Had some trouble changing modes. Had to do some weird contortions to hit the button
- Had trouble using the grip
- Controller is too big
- Said issues mostly had to do with their disability

Participant #20:

- Acceleration is a little scary
- Had difficulty pressing arm buttons
- Encountered bug with filtering menu
- Wanted to place a green cube on top of a floor tile, but couldn’t
- Said it felt pretty good, but the control scheme was a bit weird

Appendix E: User Study Timed Trials Results

Participant #	Trial #1	Trial #1 Time	Trial #2	Trial #2 Time	Trial #3	Trial #3 Time	Notes
1	Complete	0.3.0	Complete	3.31.59	Complete	3.24.14	Trial #2: We forgot that filtering affected selection, made time a lot longer
2	Complete	0.2.8	Complete	0.7.2	Complete	1.19.0	
3	Complete	0.1.0	Complete	0.73	Complete	2.57.32	
4	Complete	0.4.25	Complete	0.37.10	Complete	0.51.96	Trial #2: Accidentally deleted green cube, had to recreate it
5	Complete	0.9.21	Complete	0.11.98	Complete	1.11.50	
6	Complete	0.5.1	Complete	0.5.3	Complete	0.58.54	
7	Complete	0.1.74	Complete	0.32.06	Complete	1.25.66	
8	Complete	0.3.65	Complete	0.24.20	Complete	1.21.22	
9	Complete	0.1.56	Complete	0.35.58	Complete	2.17.33	
10	Complete	0.17.07	Complete	0.34.89	Complete	4.08.47	
11	Complete	0.1.61	Complete	0.44.12	Complete	1.36.60	
12	Complete	0.1.23	Complete	0.55.73	Complete	2.08.12	
13	Complete	0.1.0	Complete	0.6.5	Complete	0.36.1	
14	Complete	0.1.2	Complete	0.13.7	Complete	1.31.6	
15	Complete	0.3.9	Complete	0.32.0	Complete	2.01.5	
16	Complete	0.1.7	Complete	0.14.6	Complete	1.39.75	
17	Complete	0.5.77	Complete	0.17.68	Complete	1.37.56	
18	Complete	0.2.08	Complete	0.43.18	Complete	2.31.23	Trial #2: Had to turn off filtering

19	Complete	0.2.40.0	Complete	0.12.3	Complete	1.25.42	
20	Complete	0.4.0	Complete	0.31.76	Complete	3.41.85	Trial #2: Moved cube up a level and had it on the wrong level

Appendix F: Survey Responses

Open Responses

What was the most frustrating aspect of this test?

- Learning controls
- I wouldn't say that any of it was frustrating. I felt like most of the controls were relatively intuitive. The only thing that was kind of a pain was the multiple selection mechanic and the object rotation.
- initially getting used to the controls
- Trying and failing to build a rock castle.
- the fact that the block you were using reset after you moved the plane up
- The grip control to editing things. Moving up and down could be mapped to different buttons so you don't move forward and backward too a little.
- Just getting used to the controls because it is so different from any kind of controls I have used before but that only lasted a couple minutes before it got a lot easier.
- Slightly dizzy from accidentally touching the touchpad. Also I wanted more player feedback when I was interacting with objects in the world. (i.e objects highlighting when hovering pointer at it in edit mode)
- Trying to select objects without accidentally making multiple copies, using the touch to cycle through objects (not tactile enough)
- I would like to see a HUD of some kind. It was frustrating but understandable that you could not delete mistakes.
- Just the learning curve for the buttons
- Trying to move things when I am in a different mode
- When moving vertically, you have a moment of moving forwards or backwards while touching the pad but not clicking the pad just yet.
- Trying to cycle through objects without holding the trigger. Selecting multiple objects felt a little awkward, would be better if you could select objects while moving your head.
- Getting slightly used to the controls
- My thumb would rest on movement and it would cause me to move left and right quickly.
- Movement maybe was a little too "zippy" for me. As I was learning the controls, it could be a little frustrating when I wasn't in the right mode or I ran into a bug.
- Sometimes I wanted to move vertically by clicking but instead would move forward because of the tap before the click (idk if that makes sense).
- The maneuver to place items was not frustrating at all but a little confusing at first.
- The hardware was too touch sensitive and the controller was too large for my hands. the key mapping

What was the most satisfying aspect of this test?

- Having multiple objects to choose from
- It's always fun to build things. So, that. I also felt like a minecraft character in creative mode.
- teleporting was dope
- The trials were a fun way to get used to the mechanics.
- teleporting rather than moving
- there was a lot of creativity. It was a lot of fun
- Once I got the hang of how to move around and achieve the objectives because it is so different to start out. Just being able to fly around in the world without having to think much about it was really satisfying.
- The trials were very fun. I liked the part where I had to replicate a structure.
- Moving objects and transporting
- The mechanics at times were intuitive based on my previous game experience. I really enjoyed flying and moving around. Teleporting was cool.
- The virtual reality experience is enjoyable and immersive, and building things is also satisfying, also completing the challenges is great
- Fly up and teleport
- I liked to teleport around since it was faster than moving with the d-pad
- The ability to teleport and move around the world easily.
- Being able to complete the tasks and having fun while doing it :)
- The most satisfying part was when I finally pretty much understood the controls and was able to complete the trials.
- It was very intuitive and easy to learn.
- Completing the tasks
- I really liked the flying it was a neat way to get a different perspective on the landscape.
- vr is fun

Do you have any comments or suggestions?

- Place menu lower so you don't have to reach your other arm up and over
- It would be helpful if there was a box drawn in space when you would selecting multiple objects so that you could physically see the space that you were working with.
- menu to choose blocks rather than switching through maybe?
- walking around in real person translating into walking around in game would help with the nauseating aspect i feel like
- Super cute game! I'm super conflicted with flying movement. One part of me really enjoys it because it speeds up building and I feel Powerful and Cool but it slightly hurts my head when I do it.
- It would have been nicer for objects to snap to vertical grids as well instead of having to manually change the z axis.
- Keep going and let me know when you are done. :) Love it.

- It is an enthralling experience, I would recommend to all my friends
- Rotating a group selection was a little wonky
- Maybe have a game play mode. In this mode, you won't be able to go through walls, and can go up and down the stairs.
- Loved that the blocks are aligned in the grid once placed. Maybe more wizard theming maybe. Overall, great work. Really enjoyed it and wanted to stay longer to create stuff. Good Luck!
- I think maybe movement with the trackpad should be a little bit slower, if possible. Also, the menu should be placed a little bit better because I was punching my arm to change input mode.
- Maybe there's a way to differentiate the user's desire to fly or move forward
- Maybe have an interface that automatically adjusts grid level to where you are facing. That can be toggled on and off.
- Move the buttons for mode selection to a different location or something.
- Remap the keys

Appendix G: Known Issues and Bugs

Id	Description
1	The door prop in the ww_basic_assets asset bundle cannot be deleted or selected once added to a level. This is due to the door asset not having any colliders.
2	When cycling through objects in the `CreateObjectTool`, the reference to the preview object can be lost if you change your `Tool` to `EditObjectTool`, leaving a dangling object in the scene. The object is not added to the scene graph and cannot be interacted with.
3	The selection marquee box does not render in VR for the `EditObjectTool`.
4	The rotation of objects is not always at the center, and causes objects to move when being rotated, instead of being rotated in place
5	Z fighting can occur when attempting to move a selection of tiles into a cell that is already occupied by tiles.
6	If an asset bundle contains more than one asset with the same name, only the first asset is loaded. This occurs even if the assets have different directory paths. Resource tags need to take into account the relative path of an asset.
7	Sometimes props are not placed properly on top of surfaces. This may be due to the collisions of some tiles being incorrect, or an issue with the tool code.
8	The shift access modifier is not always released properly on the desktop controls.
9	The ww_basic_assets asset bundle is hardcoded into the `WWObjectGunManager.cs`, meaning that a user needs to modify the code to use their own asset bundles.
10	Currently, World Wizards only saves to a single save file named test.json.

Appendix H: Project Setup

World Wizards lives in a public code repository on Github at https://github.com/tbeaupre/worldWizardsCore. The repository contains both the source code and the resources needed to use World Wizards in its host engine, Unity. The resource folder contains various Unity prefabs that are used by World Wizards. The core folder contains all of the source code, written in C#. Bundled within the core folder are third party dependencies for the Unity HTC Vive SDK and JsonDotNet.

To setup the project in Unity, clone the repository into the Asset folder of a Unity Project.

Two other folders are necessary for setting up World Wizards in Unity. In the root of your Unity Project folder, a folder named AssetBundles and a folder named SaveFiles must be created. Asset bundles must be placed in the AssetBundles folder.