UNIVERSITY OF TARTU

Institute of Computer Science

Software Engineering Curriculum

Toomas Tamm

DeltaVR Multiplayer 2.0

Master's Thesis (30 ECTS)

Supervisor(s): Mark Muhhin, MSc

Tartu 2023

DeltaVR Multiplayer 2.0

Abstract:

This thesis presents an improved version of DeltaVR, building upon its predecessor, DeltaVR Multiplayer, by addressing game-breaking bugs, enhancing the virtual Delta Building, and introducing new experiences. The updated application features a tutorial, an improved archery range experience, and two new experiences: a drawing experience and a virtual spacewalk. Usability and performance testing showed that the new version is more stable, user-friendly, and visually accurate than its predecessors. While some usability issues and bugs remain, the application runs faster, and it should have a reduced incidence of nausea among users. Overall, the new version of DeltaVR offers a better experience with further scope for improvements.

Keywords:

Computer Graphics, Virtual Reality, Unity, User Testing, Game Design, Software Development

CERCS: P170, computer science, numerical analysis, systems, control

DeltaVR mitmikmäng 2.0

Lühikokkuvõte:

Lõputöö kirjeldab täiustatud versiooni DeltaVR-ist, ehitades selle eelkäija DeltaVR mitmikmängu peale. Töö käigus parandati mängu rikkuvaid vigu, täiendati virtuaalset Delta hoonet ja lisati uusi kogemusi. Uuendatud rakendusel on sisseehitatud õpetus, parandatud vibulaskmise kogemus ja kaks uut kogemust: joonistamine ja kosmoses kõndimine. Kasutaja- ja jõudluse testimise tulemused näitasid, et uus versioon on stabiilse, kasutajasõbralikum ja visuaalselt täpsem kui eelkäijad. Kuigi mõned kasutatavusprobleemid ja vead on veel alles, töötab rakendus kiiremini ja kasutajatel peaks olema vähem iiveldustunnet. Kokkuvõttes pakub uus DeltaVR kasutajatele paremat kogemust, millel on veel arenguruumi.

Võtmesõnad:

Arvutigraafika, Virtuaalreaalsus, Unity, Kasutajatestimine, Mängudisain, Tarkvaraarendamine

CERCS: P170, arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Table of Contents

Introduction

In recent years, the virtual reality (VR) industry has experienced remarkable growth, with increasing numbers of people embracing VR technology for various applications. According to Statista, the number of active VR headsets worldwide reached almost 20 million units in 2022, demonstrating the growing interest in experiences offered by this technology. [1]

Furthermore, the Road to VR reports that Valve's Steam platform, one of the largest digital distribution platforms for PC gaming, has seen a steady increase in VR users, further underscoring the potential for VR applications in various fields.[2] Given this context, it becomes crucial to explore innovative applications of VR technology and understand how they can be enhanced to provide users with better experiences.

This thesis aims to build upon the work of two previous theses, *DeltaVR* and *DeltaVR Multiplayer,* by addressing bugs and issues, expanding the virtual environment with the entire Delta Building, and providing more experiences for users. By drawing from the experience gained from previous projects, this thesis seeks to improve the application's overall performance, usability and experience for users.

Chapter 1 covers similar works to this application, its predecessors, and their demonstration at expos. Chapter 2 explains the goals and design of this thesis in detail. Chapter 3 explains the implementation of the goals covered in the previous chapter. Chapter 4 talks about the demonstration of this application at an expo and performance and usability testing.

The appendix contains a dictionary of technical terms used in this thesis, tables for issues discovered in predecessors, communications sent to testers, a feedback questionnaire used during testing, a guide on how to use the application and pictures of the application.

The attached materials include the build utilized in usability and performance testing, along with the results and code for generating graphs. Additionally, it has usability testing data, an Excel file containing graphs, and various videos and images showcasing the application.

[1] https://www.statista.com/statistics/677096/vr-headsets-worldwide/

[2] https://www.roadtovr.com/valve-fix-steam-survey-vr-population/

The source code for the application can be found in the following link: https://gitlab.com/UT-CGVR/deltavr/.

This thesis has been written using AI-based tools, including Grammarly[3] and ChatGPT[4], to enhance its readability. Grammarly facilitated in-text refinements, while ChatGPT was provided with paragraphs and instructed to improve them in accordance with thesis writing guidelines.

[3] https://app.grammarly.com/

[4] https://chat.openai.com/

1 Background

This chapter overviews the background and context necessary to understand virtual reality (VR) games with experiences. As the demand for engaging and interactive multiplayer experiences continues to grow[5], exploring the design and functionality of existing VR games and platforms is crucial.

This chapter specifically focuses on four experiences. Subchapter 2.1 explains *The Lab* by Valve Corporation. Subchapter 2.2 explains *VRChat*. Subchapter 2.3 explains another university's VR tour. Subchapter 2.4 explains *DeltaVR Multiplayer* by Joonas Püks and its demonstrations during expos.

1.1 The Lab

The Lab[6] is a virtual reality game developed by Valve Corporation, the company behind popular games such as *Half-Life*, *Portal*, *Counter-Strike*, and *Dota 2*. *The Lab* is a collection of VR experiences designed to showcase the capabilities of the HTC Vive, a VR headset developed in collaboration between HTC and Valve.

These experiences include physics-based puzzles, tower defence, a retro-style arcade, immersive experiences in scenic locations, educational explorations of the human body and solar system, a robot repair interactive movie, and a *Dota 2*-inspired secret shop. An example of the tower defence game mode is given in Figure 1.

[5] https://www.roadtovr.com/valve-fix-steam-survey-vr-population/

[6] https://store.steampowered.com/app/450390/The_Lab/

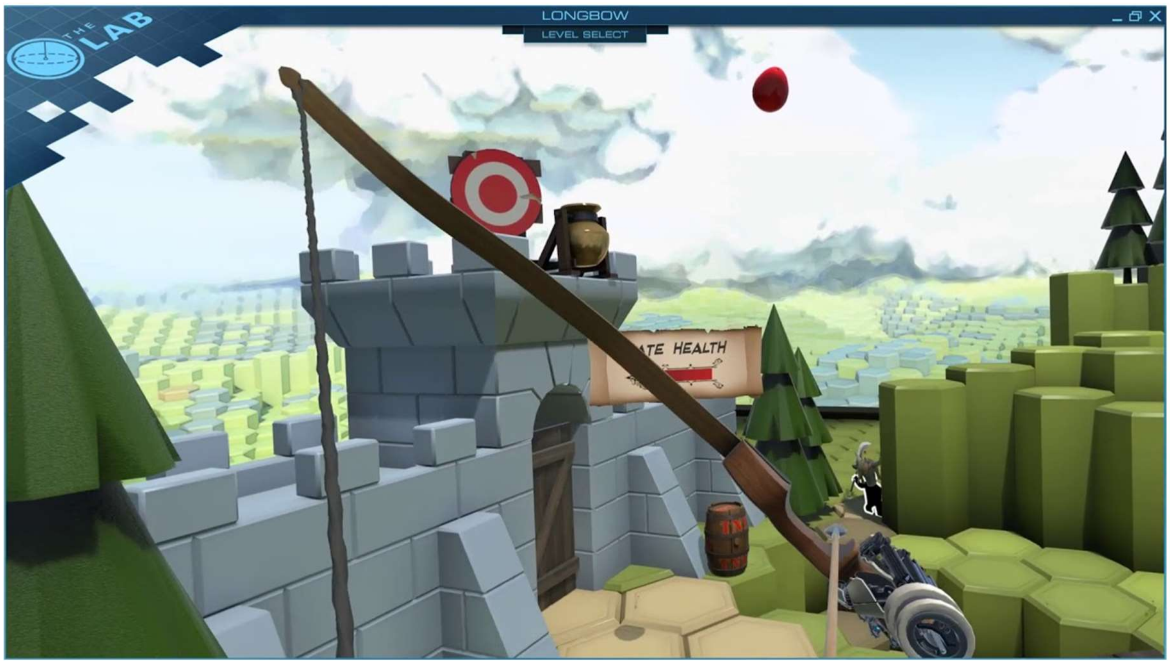


Figure 1. *The Lab*'s tower defence game mode.

In the context of *DeltaVR*, *The Lab* serves as a highly relevant example due to its implementation of various VR experiences. However, the main difference between *DeltaVR* and *The Lab* is that the first takes place in the Delta Building and has multiplayer support.

1.2 VR Chat

VRChat[7] is a free-to-play online virtual reality social platform that allows users to create, share, and explore user-generated virtual worlds and avatars with others worldwide. The platform supports VR headsets such as Oculus Rift, HTC Vive, Valve Index, Windows Mixed Reality, and desktop mode.

[7] https://store.steampowered.com/app/438100/VRChat/

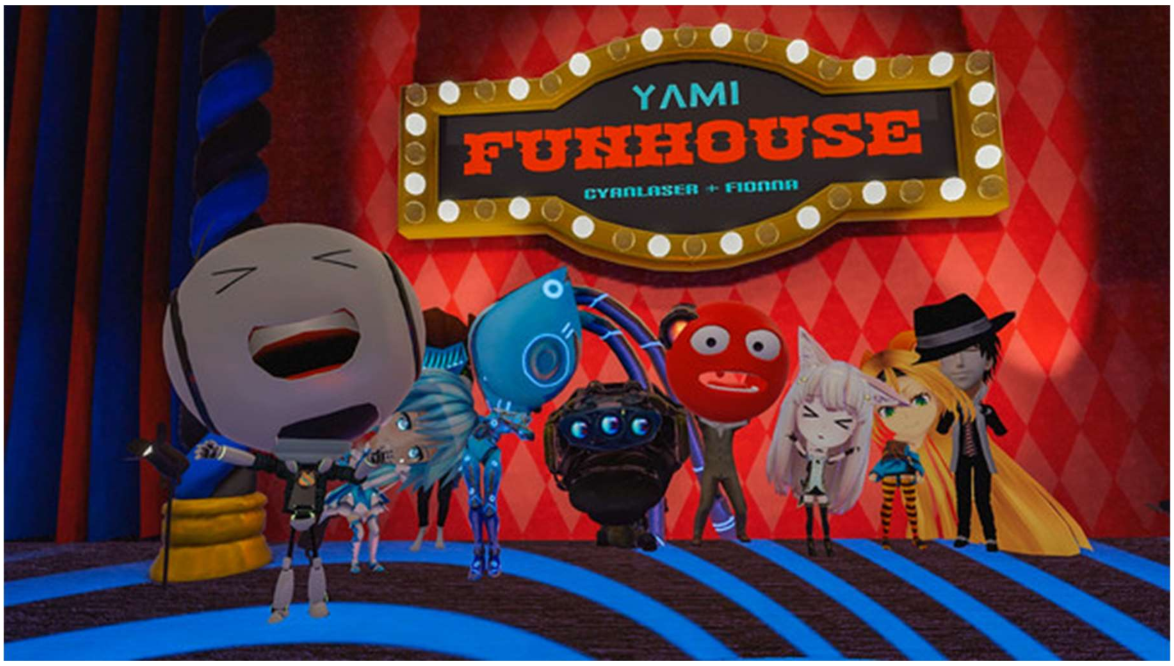


Figure 2. One of *VRChat*'s worlds

VRChat has gained popularity for its open and creative community, where users can express themselves through their avatars and worlds, participate in events, and explore diverse social experiences. One of these worlds is shown in Figure 2.

In the context of *DeltaVR*, *VRChat* is a relevant example due to its focus on being able to explore a virtual environment together. Although *DeltaVR* does not provide the same level of customization, its multiplayer functionality enables social interaction between users. Additionally, *DeltaVR* aims to create a more controlled and guided experience, focusing only on exploring the Delta Building and its various experiences.

1.3 UNSW Business Classroom VR

The *UNSW Business Classroom VR*[8] is an Android VR application that displays a virtual replica of one of the University of New South Wales's classrooms. It was introduced as part of the university's Open Day. Within the classroom, users can watch videos and experience what studying there might be like. Figure 3 presents an image of the application.

[8] https://newsroom.unsw.edu.au/news/business-law/virtual-reality-unsw-open-day

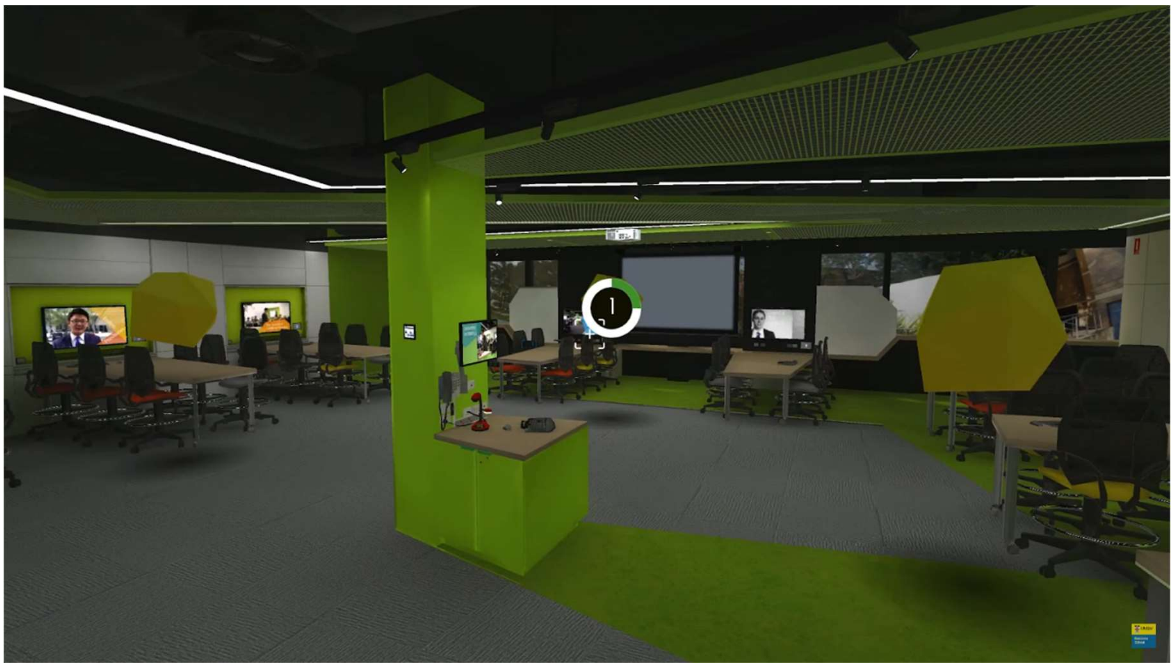


Figure 3. The classroom shown in the application.

In the context of *DeltaVR*, both applications aim to showcase their respective buildings. However, *DeltaVR* extends beyond a single classroom and aims to provide an entertaining experience.

1.4 DeltaVR Multiplayer

DeltaVR Multiplayer[9] by Joonas Püks is a multiplayer version of the original *DeltaVR* experience. It extends the original version by allowing players to explore the building together in virtual reality or the new non-VR version. The multiplayer features are synced players and proximity voice chat. (Püks, 2022, pp. 13–19)

Besides the new multiplayer features, two other notable features were added to the experience. One is a new material collection experience, where the materials are hidden inside the building. The second one is the addition of a teleport menu, where the player can teleport to existing experiences. (Püks, 2022, pp. 13–19)

[9] https://gitlab.com/UT-CGVR/deltavr

1.4.1 DeltaVR Multiplayer at Expos

On September 17th, 2022, *DeltaVR Multiplayer* was presented[10] at MängudeÖÖ[11], which posed several challenges during preparation and presentation. The team had initially planned to use three Quest 2's, but only two were utilized due to laptop performance issues. In addition, the two presenters struggled to adequately explain the experience to users, as the application lacked a tutorial. Nonetheless, approximately 30 people were able to try the application. A picture of the setup is shown in Figure 4.

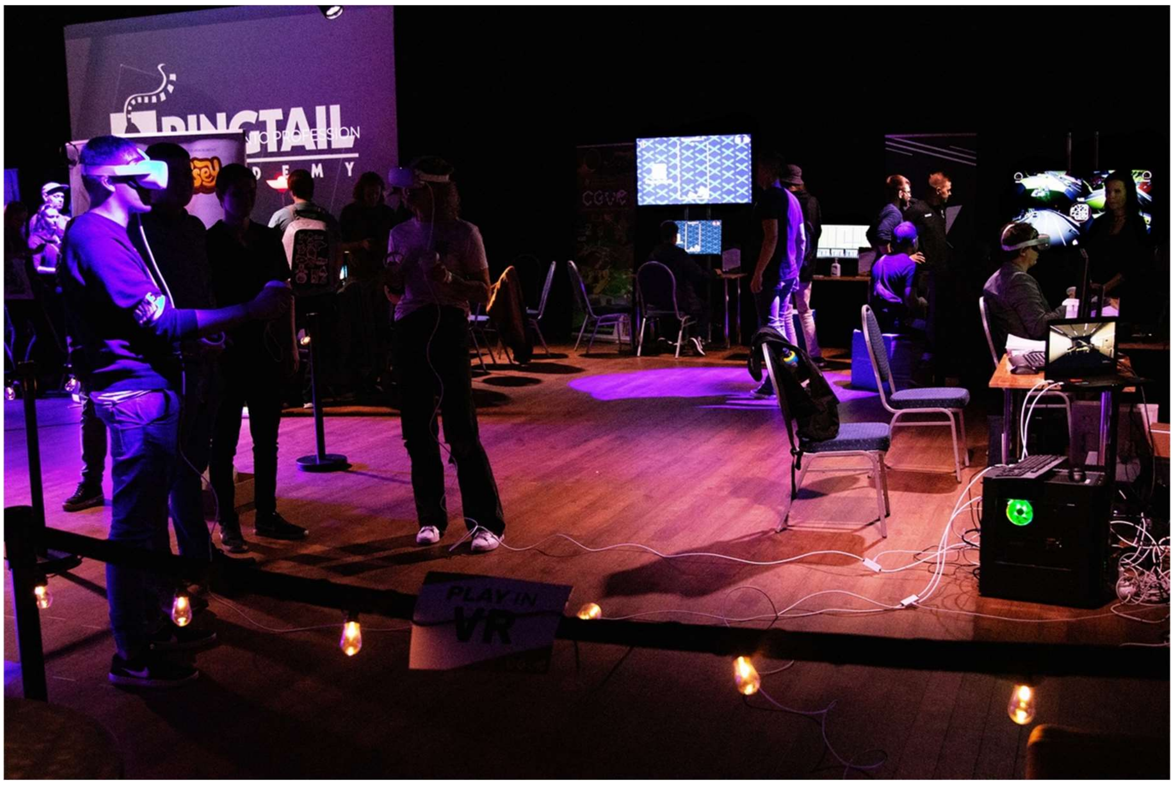


Figure 4. DeltaVR Multiplayer at MängudeÖÖ.

A second demonstration of *DeltaVR Multiplayer* was held during the "Õpi Tartus" ("Study in Tartu") Higher Education Day event[12] on January 25th, 2023. The team demonstrating had previous experience presenting the application, which made the event run more smoothly. Roughly 15 people tried the application during this event. A picture of the setup is shown in Figure 5.

[10] https://cgvr.cs.ut.ee/games-at-mangudeoo-2022/

[11] https://www.mangudeoo.ee/

[12] https://opitartus.ee/

Figure 5. DeltaVR Multiplayer at Study in Tartu event.

During both demonstrations, a list of issues was compiled by explaining how to use the application, observing users, and asking for verbal feedback. The issues were recorded in a virtual notebook and sorted into three tables in the appendix. Table 4 lists performance issues, Table 5 lists usability issues, and Table 6 lists functional issues. Each table has three columns: issue ID, name, and description. Solutions to these issues are explained in the Goal overview paragraph.

2 Goal Overview and Design

This chapter outlines the goals established during the thesis and the design decisions made to achieve those objectives. Subchapter 3.1 focuses on multiplayer, addressing its issues in the predecessor and proposing solutions. Subchapter 3.2 discusses the tutorial objectives. Subchapter 3.3 covers player movement alterations. Subchapter 3.4 details desired modifications to the Virtual Delta Building. Subchapter 3.5 examines changes to the archery range experience. Subchapter 3.6 explores the tours offered in one of the university's labs. Lastly, subchapter 3.7 considers different experiences to incorporate into the application.

2.1 Multiplayer

Many issues concerning multiplayer (see Appendix II) are due to various networking issues. The main problems are that over time objects become de-synced between players (UI-6, FI-8), and the multiplayer library requires an active internet connection due to proprietary libraries used (UI-12, UI-13).

2.1.1 Issues in DeltaVR Multiplayer

The primary cause of the first issue is likely a defect or missing sync functionality in the implementation, resulting in objects becoming de-synced between players. The issue appears to be the underlying reason for other related issues, including the bow range being only playable by one player (FI-2), bow range targets becoming stuck on the building (FI-3), targets not getting hit (FI-4), players being unable to start the bow range mini-game (FI-5), and doors teleporting around the building (FI-6).

The second main issue is that the multiplayer library uses a client-server topology, which hosts the server in the cloud. Because the server is hosted in the cloud, an external internet connection is required to demo the application. Depending on the number of people participating at the venue, this may be unavailable, costly, or unreliable in expos. Consequently, the experience cannot be demoed at the location or cause synchronization issues and stuttering for the player.

Additionally, since the current multiplayer library called Photon is proprietary and needs to be hosted on the cloud, the multiplayer may stop working if Photon makes any breaking changes or licensing changes in the future. It would be necessary to stay up to date with any modifications Photon makes and be prepared to update the experience accordingly to avoid these issues. Due to the previously mentioned issues, it has been decided to switch the networking library used in *DeltaVR*.

2.1.2 Network Topologies and Architectures

Before selecting a library, potential network topologies are evaluated for *DeltaVR*. A more suitable option might exist besides the client-server model. Glazer & Madhav discuss two topologies: Client-Server and Peer-to-Peer (2015, pp. 166–169). An illustration of the topologies is given in Figure 6.

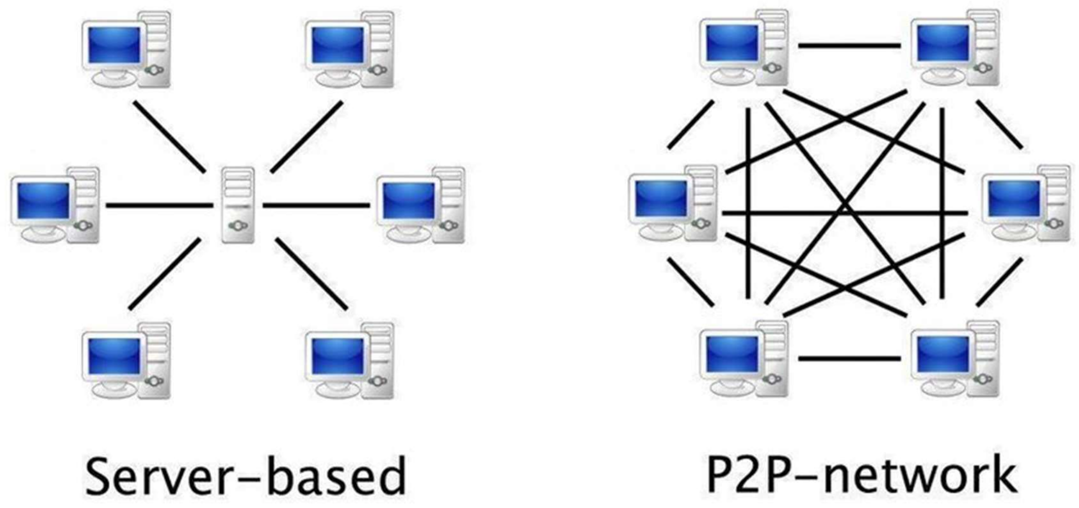


Figure 6. Client-Server vs Peer-to-Peer network[13].

The Client-Server topology has static bandwidth requirements for clients and allows server-authoritative game logic. However, it can increase latency for clients. In contrast, the Peer-to-Peer topology connects all instances, raising bandwidth requirements and making authoritative game logic harder to implement. (Glazer & Madhav, 2015, pp. 166–169)

For *DeltaVR*, bandwidth requirements are less significant due to the few simultaneous players. Cheating is not a concern as the game is typically played on our computers.

[13] https://www.researchgate.net/figure/Server-based-network-vs-Peer-to-Peer-network-Digital-Impact-Labs-2017_fig1_344233205

However, a more powerful server could enhance performance for weaker clients, such as expo laptops. Thus, the Client-Server topology remains the choice.

Glazer & Madhav present two server types: dedicated server and listen server. The dedicated server handles game state and client communication. In contrast, the listen server allows one client to participate actively while managing the game state for all the other clients. (2015, pp. 166–169) For *DeltaVR*, the listen server is the preferred choice due to its ease of use and not requiring a separate executable for the server.

In summary, the chosen Unity library should support the Client-Server topology and have a listen server option.

2.1.2 Unity Libraries

Unity lacks native multiplayer support but allows plug-ins and libraries for this functionality. There are several official and unofficial Unity network libraries, as shown in Table 1. Comparison of different Unity networking libraries.

Table 1. Comparison of different Unity networking libraries.

	Photon PUN[14]	**Netcode for GameObjects**[15]	**Mirror**[16]	**DarkRift**[17]	**FishNet**[18]
Client-Server topology	Yes	Yes	Yes	Yes	Yes
Listen Server	No	Yes	Yes	No	Yes
Free	Limited	Limited	Yes	Yes	Limited
Open Source	No	Yes	Yes	Paid	Yes
Unity 2022 Support	Yes	Yes	No	Unknown	Yes

[14] https://www.photonengine.com/PUN

[15] https://docs-multiplayer.unity3d.com/releases/netcode/1.0.0/index.html

[16] https://github.com/MirrorNetworking/Mirror

[17] https://www.darkriftnetworking.com/

[18] https://fish-networking.gitbook.io/docs/

Photon PUN is currently used in *DeltaVR Multiplayer*, but its issues and lack of listen server support make it unsuitable.

Netcode for GameObjects, created by Unity, meets our needs, but its novelty[19] raises concerns about community support and undiscovered bugs.

Mirror fits our networking requirements but is not compatible with Unity 2022, which *DeltaVR* requires for specific XR features.

DarkRift, while older than Netcode for GameObjects, suffers from sparse documentation, making troubleshooting challenging.

Fishnet supports our desired topology and architecture. It is more established than Netcode for GameObjects and has broader community usage. Although some features are paid[20], they are not essential for *DeltaVR* and can be manually implemented. Therefore, *DeltaVR Multiplayer 2.0* will utilise the Fishnet library.

2.2 Tutorial

In *DeltaVR Multiplayer*, game controls posed a significant challenge for users, often requiring a dedicated helper to guide players. At the first expo, this issue was exacerbated by having only two helpers for three gameplay stations. Implementing an in-game tutorial could provide a self-guided introduction to game controls, as done in games like *Until You Fall* (Figure 7) and *Half-Life Alyx* (Figure 8).

[19] https://docs-multiplayer.unity3d.com/releases/netcode/1.0.0/index.html

[20] https://fish-networking.gitbook.io/docs/master/pro-and-donating

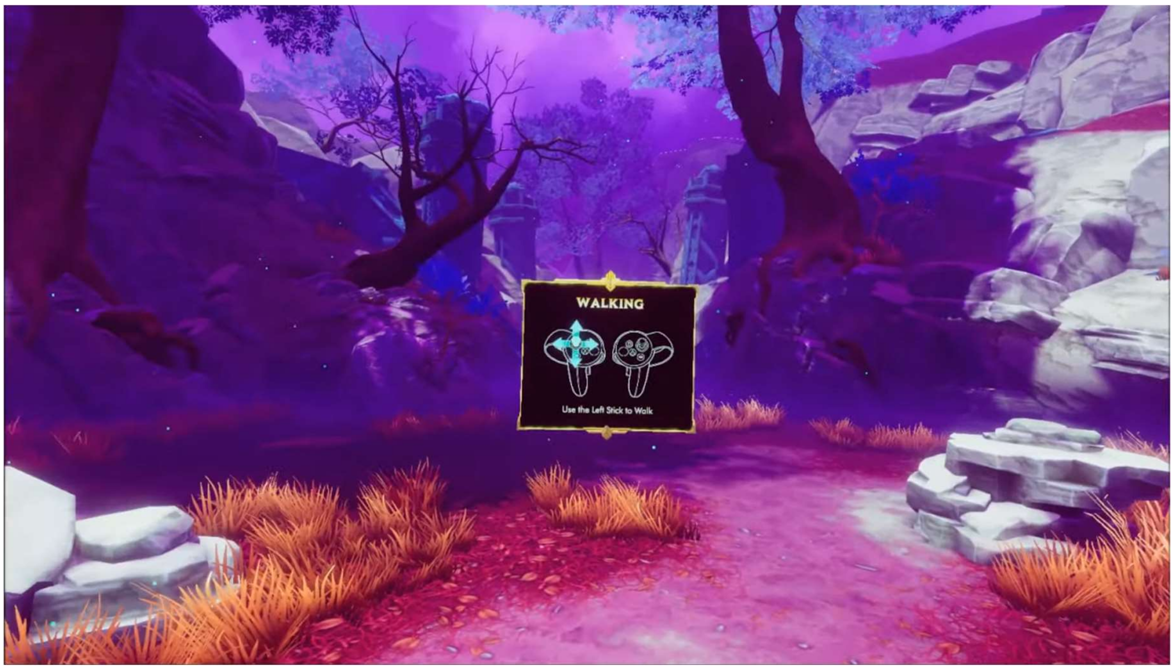


Figure 7. Text and Diagram modality used in Until You Fall.

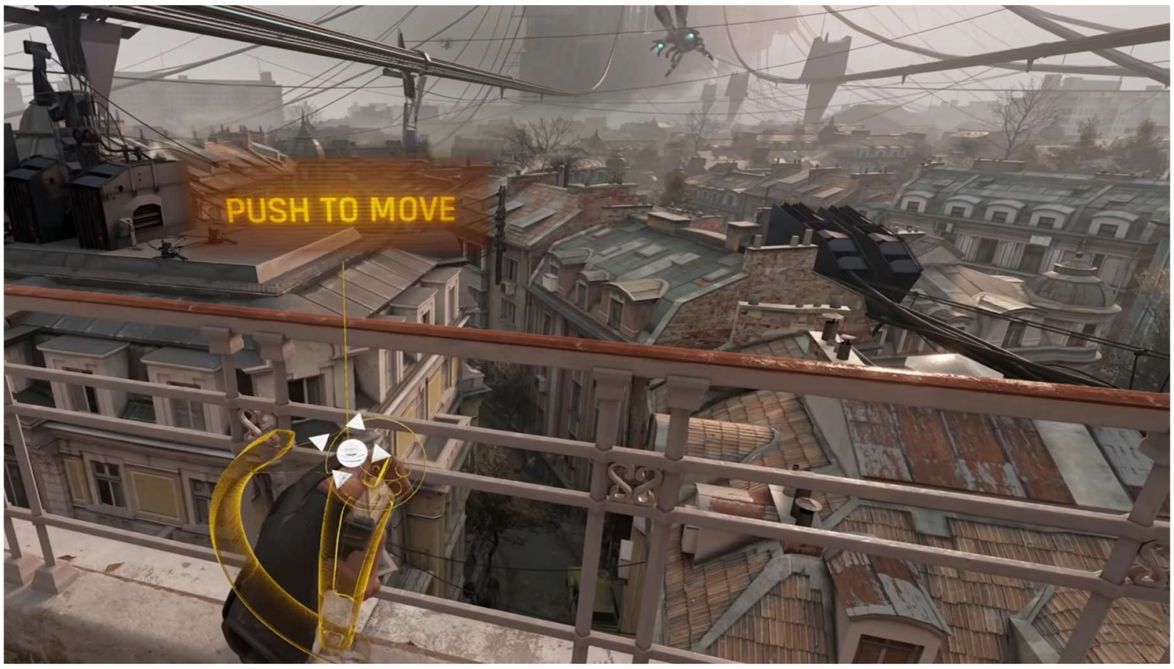


Figure 8. Text and Spatial modality used in Half-Life Alyx.

Kao et al. identify three common modalities for teaching game controls: Text, Text and Diagram, and Text and Spatial. Text solely relies on written instructions, while Text and Diagram include a controller diagram. Text and Spatial presents a virtual controller with overlaid instructions. Overall, Text and Spatial proved the most effective. (2021, pp. 16–20)

Therefore, *DeltaVR*'s tutorial should employ the Text and Spatial modality for optimal comprehension. The tutorial should also cover the most used controllers when demonstrating *DeltaVR*: Vive Wands, Quest 2 controllers, and Valve Index controllers.

2.3 Locomotion Changes

Locomotion refers to the ability to move oneself around. During expos, several problems were identified with the implementation of *DeltaVR Multiplayer*. One issue involves turning controls. VR headsets can be connected with a cable. To prevent users from twisting around it, *DeltaVR* includes virtual turning controls.

Users reported nausea when using these controls. In DeltaVR Multiplayer, players turn smoothly while the control is held down. Jerald suggests that instantaneous turns cause less sickness compared to smooth turns (2015, p. 211). Therefore, smooth turning should be replaced with snap turns to reduce nausea.

Glazer & Madhav mention that a lower field of view and fading scenes in and out are associated with less motion sickness (2015, pp. 200–201, 344). A vignette effect (Figure 9) could be added to all locomotion methods, smoothly fading over the user's vision during movement to reduce perceived motion.

Figure 9. Example of a vignette.

In *DeltaVR*, teleportation is another locomotion method. In *DeltaVR Multiplayer 2.0*, users can toggle the teleportation ray's visibility with a button and teleport to the ray's endpoint by pressing another button. However, users frequently forget to turn off the ray, obstructing their view while interacting with objects. To enhance usability, the ray should be modified so that users must hold the button to aim and release it to teleport to the ray's endpoint.

Finally, users faced issues with the building setup, including one-way walls and non-functional doors. To address these problems, meshes should be fixed, and broken doors must be repaired. Moreover, some surfaces were mistakenly marked as teleportable, leading players to teleport inside walls or unintended locations. To resolve this, teleportable surfaces should be reviewed and restricted to floors and stairs only.

2.4 Building Improvements

The previous two theses on this project only included the first two floors, which were optimised and furnished based on the DBV (*Delta Building Visualization*) project. The first *DeltaVR* thesis improved on the DBV project by adding lamps, better walls, etc. In this thesis, the building should be further improved. The missing floors should be readded, along with distinctive features that make the building easily recognizable, such as the external ribs shown in Figure 10.

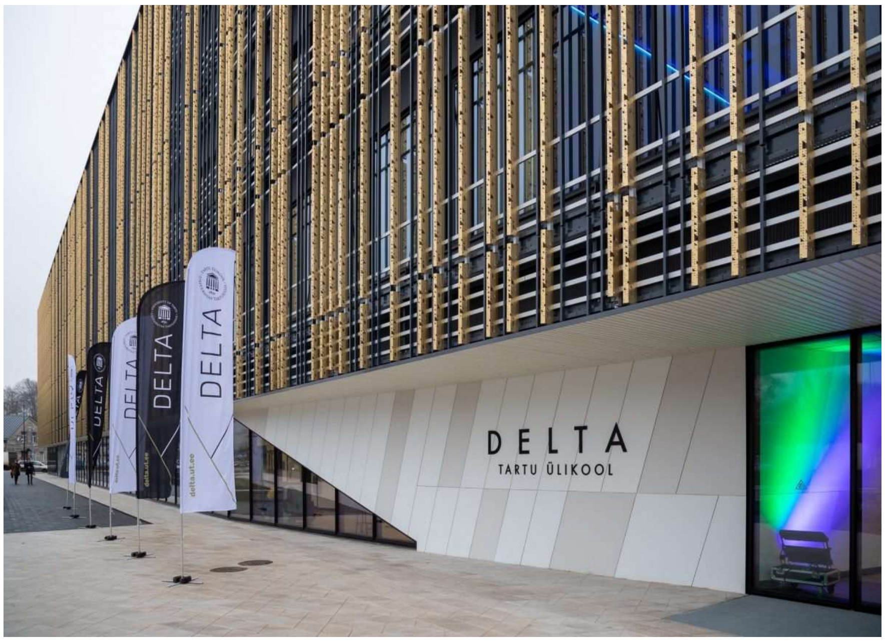


Figure 10. Golden and black metallic ribs surround the Delta Building.

Furthermore, issues still need to be addressed with the existing floors, such as missing mesh faces and lighting artefacts in certain areas.

2.5 Archery Range Modifications

The archery range (Figure 11) was the most popular experience in the original DeltaVR thesis (Tamm, 2021, p. 30) and was frequently showcased at expos. Nonetheless, the latest version encounters issues related to networking and gameplay design. As discussed in Chapter 2.1, multiple networking problems hindered players from playing this experience. These issues must be resolved to enable simultaneous gameplay for multiple participants.

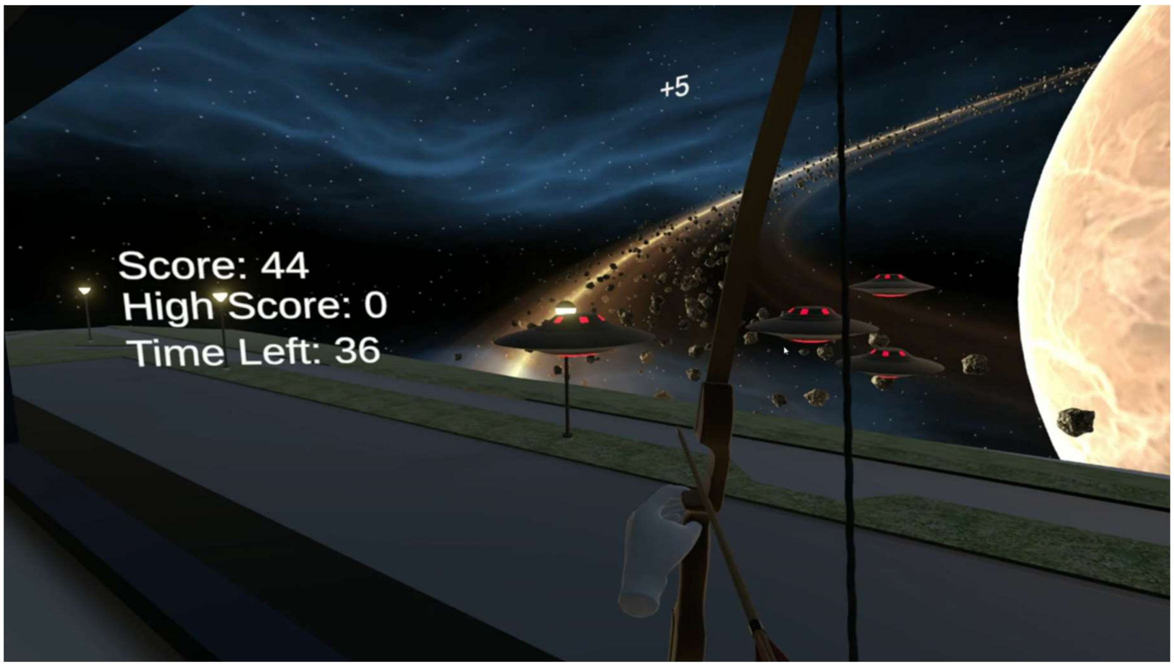


Figure 11. Picture of the archery range experience from the first thesis.

From a usability standpoint, users faced difficulties when operating the bow. The sequence of holding the bow, picking up an arrow, loading, aiming, and releasing proved confusing, particularly for first-time users unfamiliar with the controller.

To improve bow usage, the shooting process should be streamlined. Rather than having a separate step for loading the bow, it could remain perpetually loaded. Furthermore, the grip could automatically adjust to the proper end of the bow based on whether the player is already holding the bow with one hand. This modification would also permit a larger collider to grab, enhancing usability.

2.6 CGVR Lab Tour

The University of Tartu Computer Graphics and Virtual Reality Study Lab (CGVR) provides tours[21] on its website, offering the opportunity to try out VR applications. Typically, individuals have 5-6 minutes to test an application. Although some suitable options like The Lab (see Chapter 1.1) are available, they do not precisely fit the 5–6-minute timeframe and do not showcase CGVR Lab's unique offerings.

Demonstrating an application created by the University of Tartu students could highlight potential learning opportunities for those who choose to study there, aligning with the tour's goals.

The current version of *DeltaVR* is unsuitable for the tour, as it lacks a tutorial and engaging content, as shown by expo feedback. A time-limited mode for *DeltaVR* could be developed to cater to the tour's needs. Additional experiences should be added to fill the 5–6-minute duration.

2.7 Additional Experiences

The current version of *DeltaVR* may be considered dull due to its limited selection of experiences. The following subchapters discuss new experiences and the bow range that will be incorporated into the application.

2.7.1 Non-Euclidean Space

A key distinction between real and virtual environments is that virtual spaces do not need to adhere to physical laws. One intriguing way to defy these laws is by using non-Euclidean geometry.

[21] https://cgvr.cs.ut.ee/teenused/

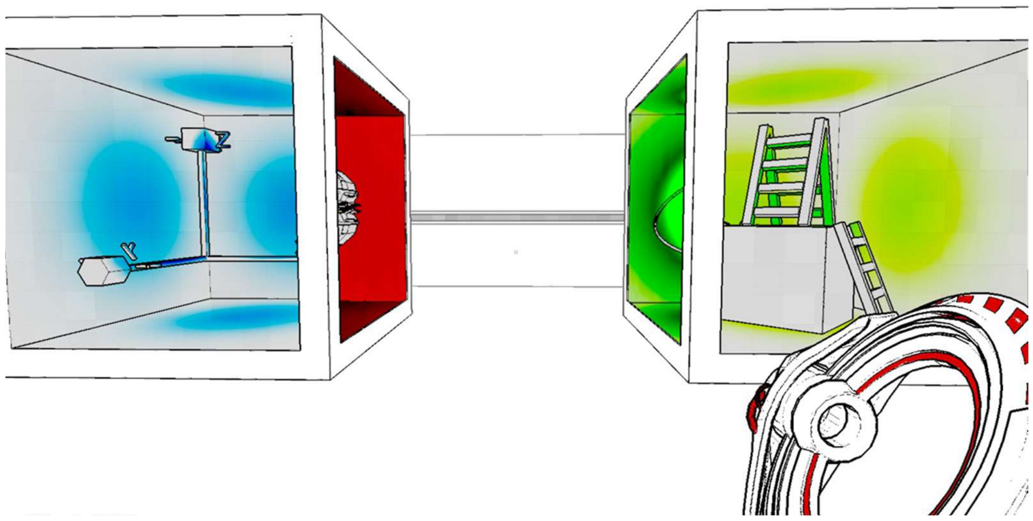


Figure 12. Non-Euclidean geometry in Antichamber.

Non-Euclidean geometry in video games enables the creation of worlds that defy classical geometry rules, allowing for illusions such as impossible architecture or teleportation portals. Implementing this technique in VR can offer players an immersive experience of impossible physics, as seen in successful games like *Antichamber*[22] (Figure 12).

2.7.2 Virtual Plank

Virtual plank experiences offer a safe way to simulate dangerous scenarios. By immersing users in the experience, they can feel the same emotions as if they were in real danger. A popular example of this is *Richie's Plank Experience*[23], a VR game that simulates walking on a virtual plank with a long fall below (Figure 13).

[22] https://store.steampowered.com/app/219890/Antichamber/

[23] https://store.steampowered.com/app/517160/Richies_Plank_Experience

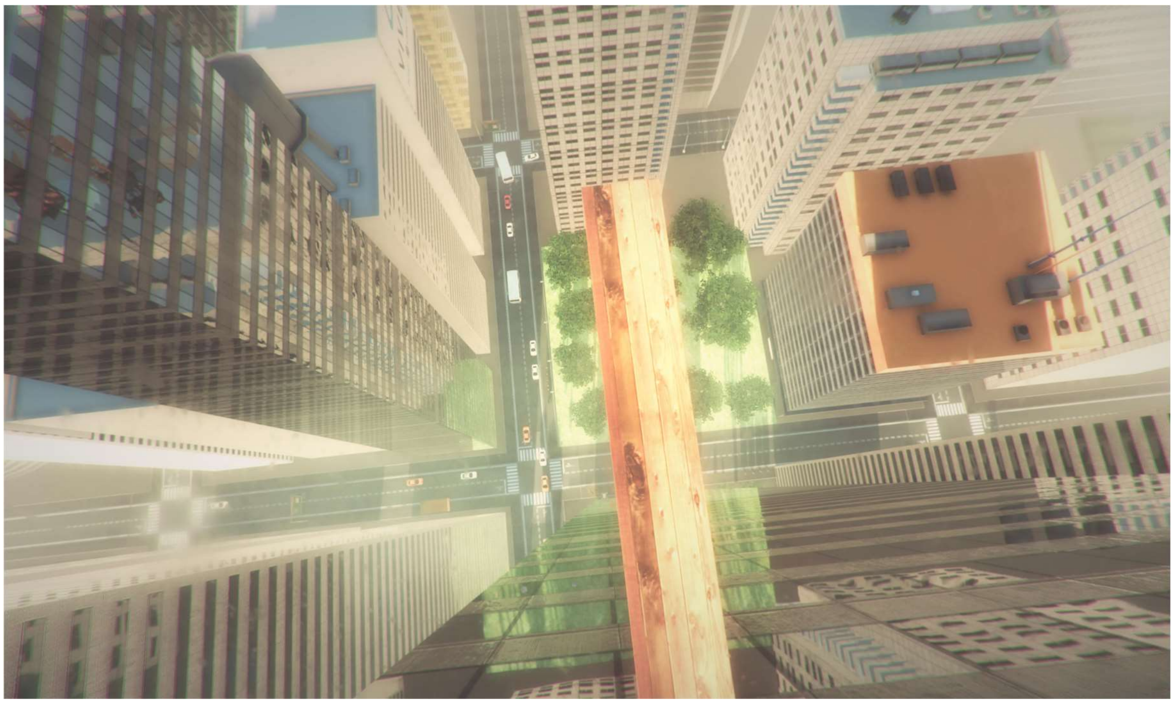


Figure 13. Richie's Plank Experience.

When adding this experience to *DeltaVR*, it could be combined with non-Euclidean space. For example, creating a portal to space where the user would walk across a plank while seeing the vast expanse of space below them. Additionally, this experience could help players practise locomotion controls.

2.7.3 Drawing

Another experience for *DeltaVR* is drawing with spray cans. Drawing is a creative activity that can be easily adapted to VR. This could be a fun and creative experience for users, allowing them to express themselves and even collaborate with other users in multiplayer. Additionally, players could practise interacting with simpler objects before moving on to more complex ones, such as the archery range with the two-handed bow.

3 Implementation

DeltaVR Multiplayer 2.0 was developed utilizing the interactive design process. Zimmerman characterizes this process as repeatedly prototyping, testing, and analysing as frequently as possible. Employing this approach allows for the early detection of bugs and issues within the application (2003, pp. 176–147). In the context of this thesis, multiple iterations of this application underwent testing during development by the author, at expos by visitors, and ultimately during usability testing.

Section 3.1 provides an overview of the technology employed during development. Section 3.2 discusses the implementation of player hands to support the tutorial. Section 3.3 delves into the implementation of the tutorial. Section 3.4 offers an overview of modifications made to the virtual Delta Building. Section 3.5 outlines the implementation of non-Euclidean portals and the plank experience. Section 3.6 details the implementation of the drawing experience. Section 3.7 highlights the changes made to the bow range. Lastly, Section 3.8 examines the implementation of multiplayer using the Fishnet library.

3.1 Technologies Used

This thesis builds upon the previous thesis on *DeltaVR*, using the Unity game engine with an upgraded version to 2022. This upgrade enables support for additional VR features with the XR Interaction Toolkit plugin, specifically version 2.3, which allows for two-handed object manipulation. The previous versions of DeltaVR used the 0.9 pre-release version of the plugin, which had less functionality.

To modify the virtual Delta Building, Blender[24] was used, as it is a 3D modelling software capable of changing the virtual building's FBX file, and it provides a wide range of tools and options for editing meshes. Additionally, Blender is open-source and free to use.

3.2 Locomotion Changes

As outlined in Chapter 3.3, DeltaVR had some usability issues with its locomotion system, particularly concerning nausea experienced by users. DeltaVR has three non-roomscale locomotion types: teleportation, smooth locomotion, and smooth turning. However, these

[24] https://www.blender.org/

locomotion types can cause discomfort and nausea for newer users. As a solution, it was decided to add a vignette effect to the player's vision during movement (Figure 14, Figure 15).

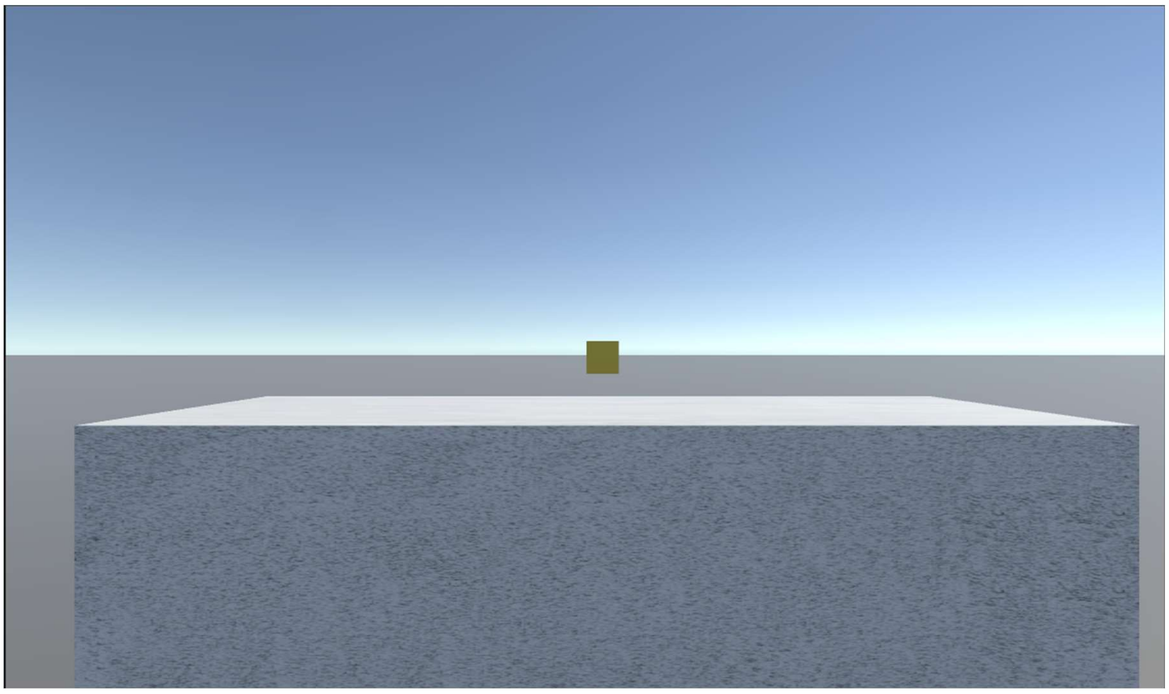


Figure 14. Player view without vignette.

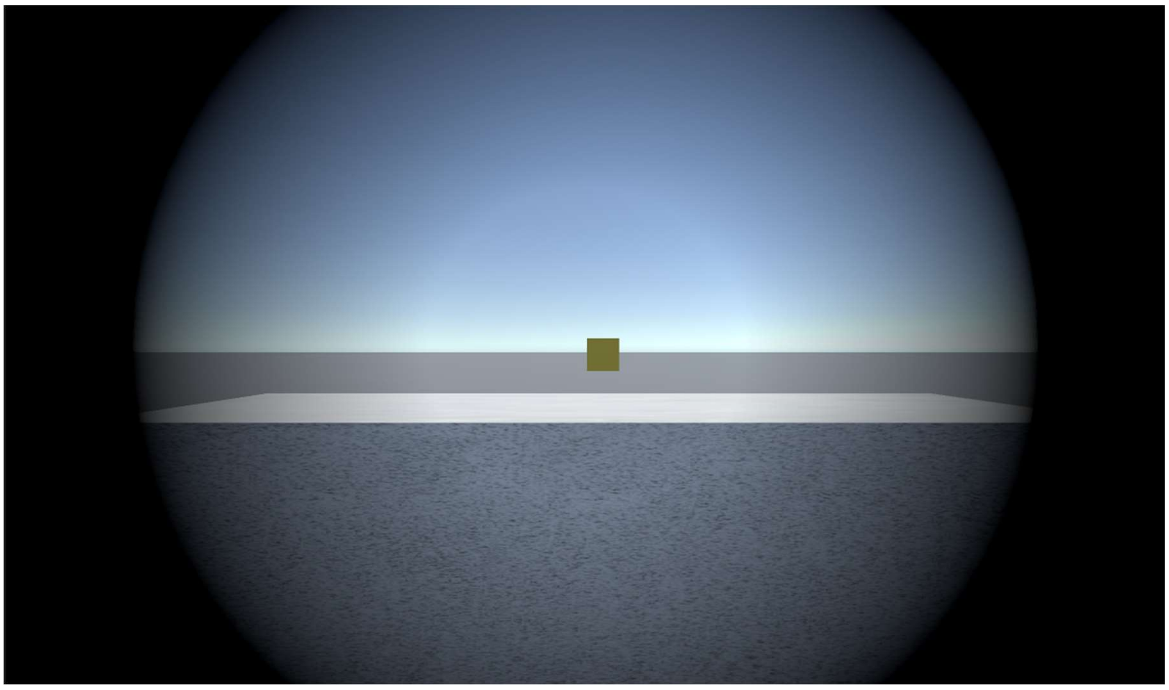


Figure 15. Player view with vignette.

The vignette has a configurable option on how much of the screen to cover and fade-in-out times. In the case of teleportation, the vignette covers the entire screen, as the player can choose where to move before the move takes place.

Only a portion of the screen is obscured when applying the vignette effect to turning and smooth locomotion, as depicted in Figure 15. This is done because the movement associated with snap turning is less abrupt than teleportation. With smooth locomotion, the user needs to see where they are going.

To ensure the vignette effect does not cause additional discomfort, a delay is introduced between the user pressing the snap turn or teleportation button and the vignette being displayed. This delay ensures that the vignette appears gradually, covering and uncovering the screen over time rather than abruptly flashing the screen black.

In response to user feedback, gravity was added to the user in *DeltaVR*. Previously, users could walk off the second floor and float in the air, unable to descend. This was described as disorienting and made the experience feel unnatural.

To address issues with the teleportation aiming toggle in *DeltaVR*, it was removed in favour of a new method that involves holding a button to aim and releasing it to teleport to the targeted location. This approach should be more intuitive and less complex than the previous toggle system.

The new method allows players to aim and interact with objects without the ray obstructing their view. Additionally, the new system allows teleportation to be cancelled by aiming at an invalid surface and releasing the button or pressing the grip button.

3.3 Virtual Hands and Controllers

In virtual reality, the headset completely covers the user's field of view, making it impossible to see the physical controllers they are holding. To address this, the application must display a virtual representation of the controllers, allowing the user to interact with the virtual environment and navigate the interface simultaneously.

At the CGVR Lab, a variety of controllers are available for use. However, during implementation, the focus was on the most used controllers: Quest Touch controllers, Valve Index controllers, and Vive Wands. Quest Touch controllers are frequently used during expos, as they are more portable than other options. In the lab, all these controllers are commonly used for demonstrations.

3.3.1 Controller Models

To display the controllers in *DeltaVR*, corresponding 3D models are required. However, at the time of writing this thesis, no suitable free packages were available for the specific controllers used in the application.

The Oculus integration plugin only provides models for Oculus devices, while the SteamVR Unity plugin only contains hand models. Although the SteamVR plugin can obtain models from the SteamVR application, this approach may not always be feasible, such as when running the application through the Oculus software instead.

One solution is to obtain the controller models directly from the SteamVR application, which can be found in the *rendermodels* folder. This folder contains OBJ files for the controller bodies and their buttons, thumbsticks, and other components.

However, two challenges arise when using these models. Firstly, the models may not be accurately positioned within the custom DeltaVR application. Secondly, these models lack any accompanying animations, making them appear static and less immersive.

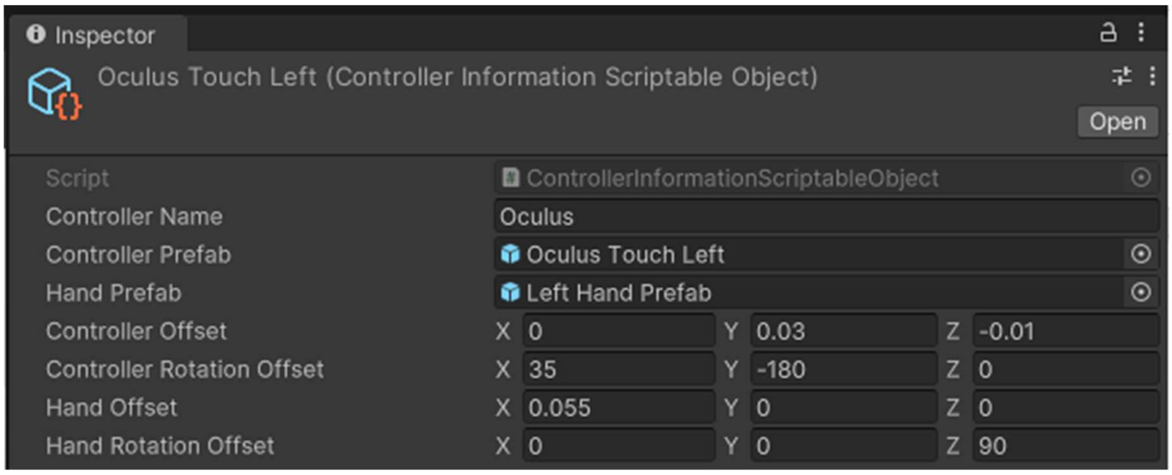
3.3.2 Offsetting Controllers

The OpenXR API[25], a cross-platform standard for virtual and augmented reality applications developed by the Khronos Group, provides information about connected controllers. To match this information with the corresponding controller models in DeltaVR, a new ScriptableObject was created (see Figure 16). This scriptable object contains information about a specific controller, including its name, prefabs[26], and offsets.

---

[25] https://www.khronos.org/OpenXR/

[26] https://docs.unity3d.com/2020.3/Documentation/Manual/Prefabs.html


Inspector

Oculus Touch Left (Controller Information Scriptable Object)

Script	ControllerInformationScriptableObject
Controller Name	Oculus
Controller Prefab	Oculus Touch Left
Hand Prefab	Left Hand Prefab
Controller Offset	X 0 Y 0.03 Z -0.01
Controller Rotation Offset	X 35 Y -180 Z 0
Hand Offset	X 0.055 Y 0 Z 0
Hand Rotation Offset	X 0 Y 0 Z 90

Figure 16. ScriptableObject for the Left Oculus Touch controller.

The controller's name is used to find the corresponding controller reported by OpenXR. When the controller is detected, prefabs for both hands and controllers are instantiated. Offsets are also included for both the hands and controllers used during prefab instantiation.

The position and rotation offsets were determined manually through trial and error. The Valve Index headset has cameras attached to it, and SteamVR offers a passthrough mode[27] that allows the user to see through these cameras while wearing the headset. The rotations and position offsets of the virtual controllers were adjusted using the passthrough mode until they matched the real-world position. These offsets were then stored in the ScriptableObject for future use.

4.3.3 Animating Controllers

The second problem with the controller models was the lack of accompanying animations. To address this, animations had to be manually created for each controller, covering three different inputs: thumbstick rotation, button movement, and grip and trigger button rotation and movement. Buttons were animated using Unity's built-in animation tool following the official documentation[28].

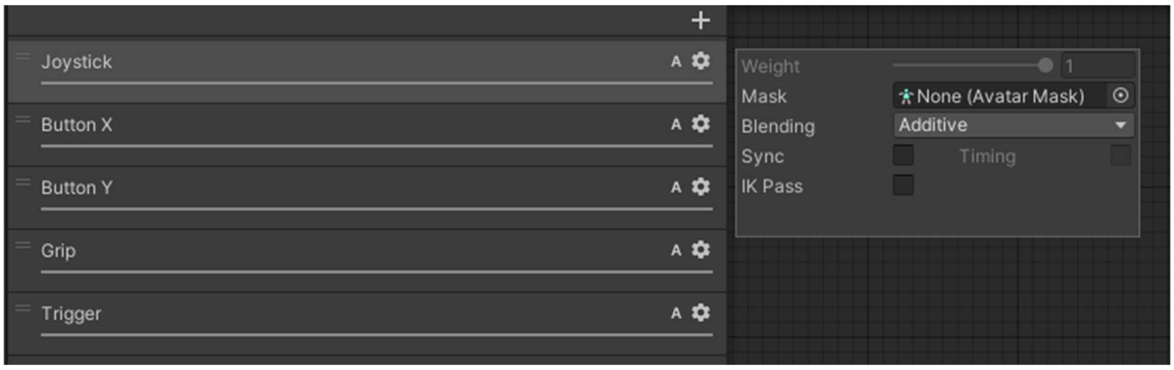
After creating the animations, each controller required a separate Animation Controller[29]. For each input, an animation layer was created (Figure 17), with all layers set to additive so

[27] https://www.youtube.com/watch?v=gzaVY8esYKg

[28] https://docs.unity3d.com/Manual/animeditor-CreatingANewAnimationClip.html

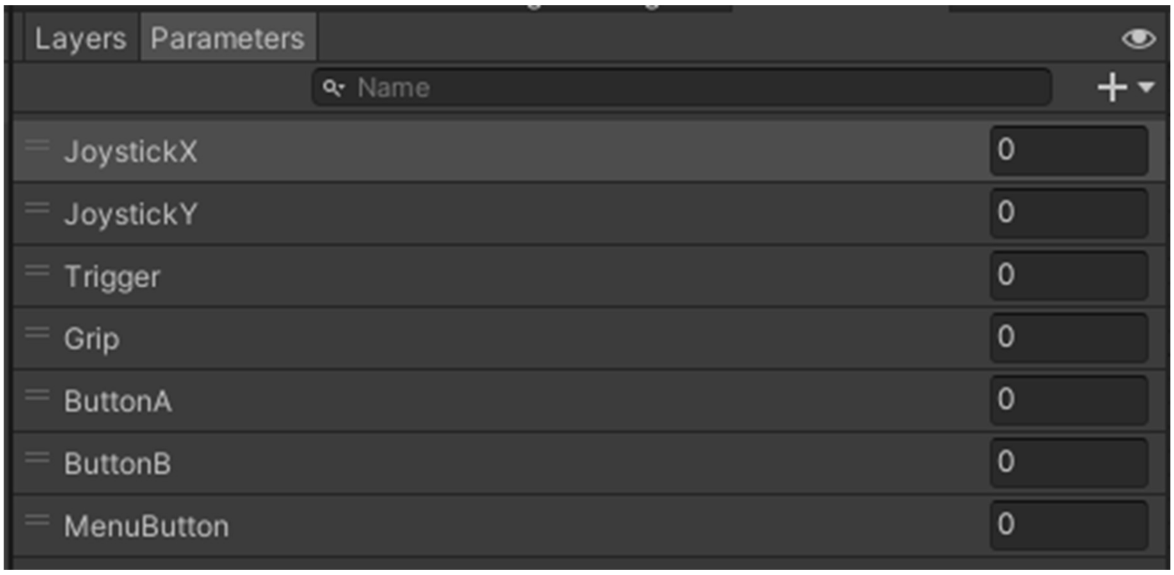
[29] https://docs.unity3d.com/Manual/class-AnimatorController.html

they can be played simultaneously. Additionally, the animation controller received a raw parameter (Figure 18) for each possible raw input provided by the OpenXR API. Each layer has a blend tree that converts the raw parameter values into specific animation keyframes (Figure 19).


Joystick
Button X
Button Y
Grip
Trigger

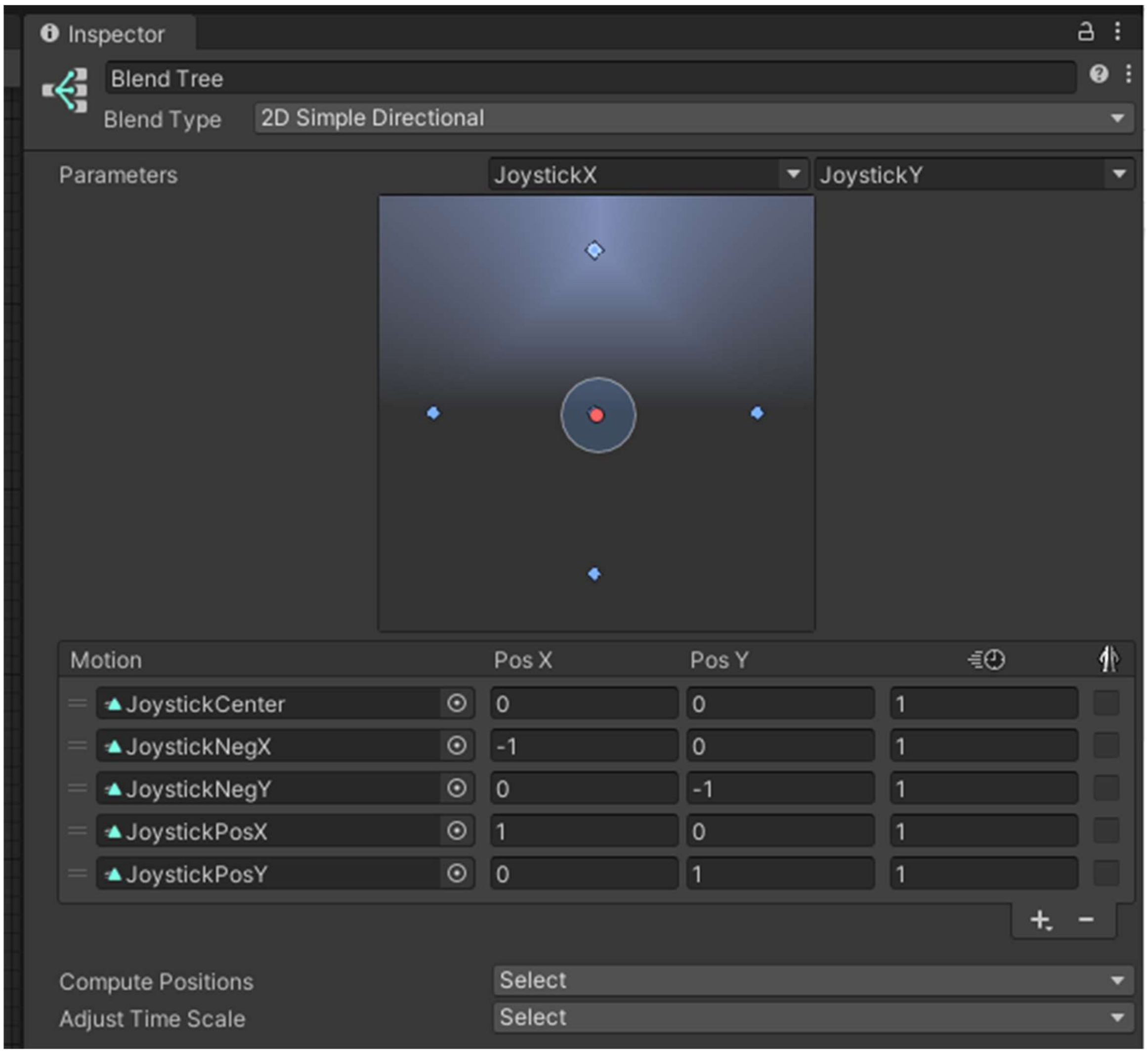
Weight: 1
Mask: None (Avatar Mask)
Blending: Additive
Sync
Timing
IK Pass

Figure 17. Layers in the animation controller.



JoystickX	0
JoystickY	0
Trigger	0
Grip	0
ButtonA	0
ButtonB	0
MenuButton	0

Figure 18. Parameters in the animation controller.


Inspector

Blend Tree

Blend Type: 2D Simple Directional

Parameters: JoystickX, JoystickY

Motion	Pos X	Pos Y	
JoystickCenter	0	0	1
JoystickNegX	-1	0	1
JoystickNegY	0	-1	1
JoystickPosX	1	0	1
JoystickPosY	0	1	1

Compute Positions: Select

Adjust Time Scale: Select

Figure 19. Example joystick blend tree in the animation controller.

A script was developed for each controller prefab. The script retrieves the current input values from the OpenXR API and passes them to the Animation Controller. This script ensures that any button presses on the physical controller is replicated by the virtual controller, making the controllers look less static and giving feedback if the user presses the right button.

4.2.4 Virtual Hands

To display the virtual hands in *DeltaVR*, the prefab from the first single-player version of the experience was reused, as it already had animations and support for all controllers used through OpenXR (Tamm, 2021, pp. 13–14). Although the hand prefab is the same for all controller types, the option to change it was still included in the ScriptableObject for future use in case specific hand animations are required.

The only modification to the hands was to make them transparent instead of white. Transparent hands reduce occlusion, making it easier for users to interact with the virtual environment (Jerald, 2015, p. 326).

3.4 Tutorial

After implementing the ability to display the controllers, they can be used to teach the player the controls in *DeltaVR* interactively. To achieve this, we use the Text and Spatial model described in Chapter 2.2.

Subchapter 3.4.1 describes how we achieve a highlight effect and show the text above it. Subchapter 4.4.2 describes how and when we use the combination.

3.4.1 Controller Outline and Text

To highlight the controller, a shader was made to outline a model (Figure 20). The shader was made using the Shader Graph tool[30] for Unity.

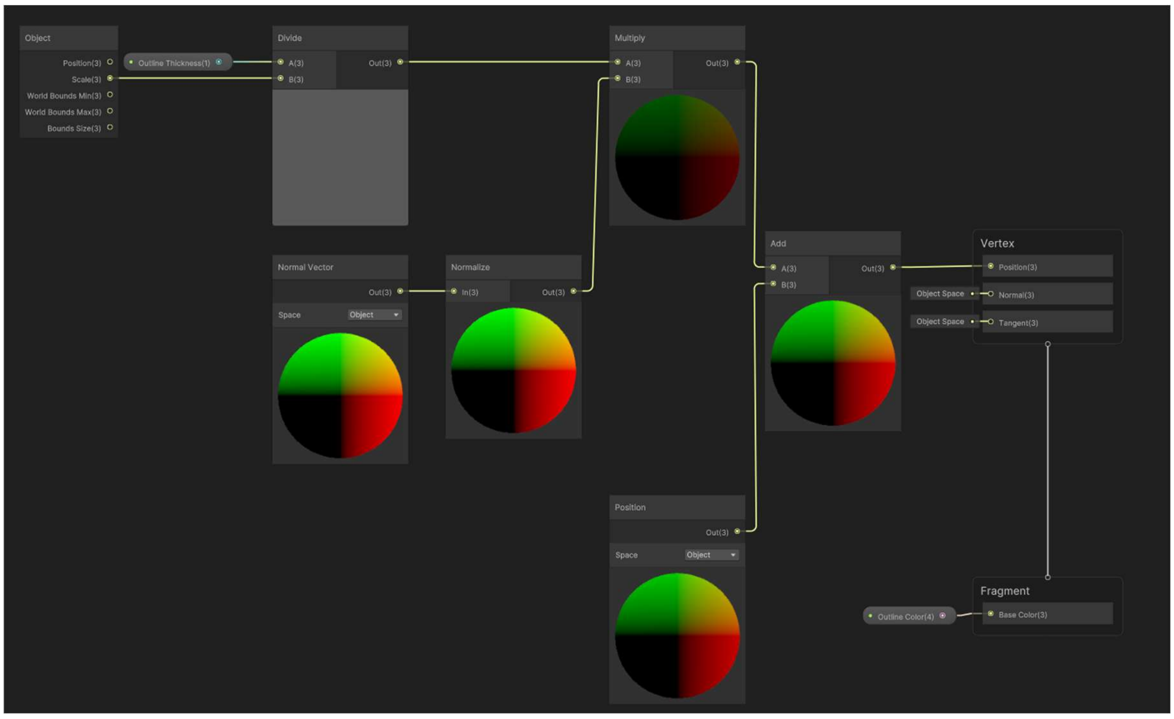


Figure 20. Shader graph for the outline shader.

[30] https://docs.unity3d.com/Manual/shader-graph.html

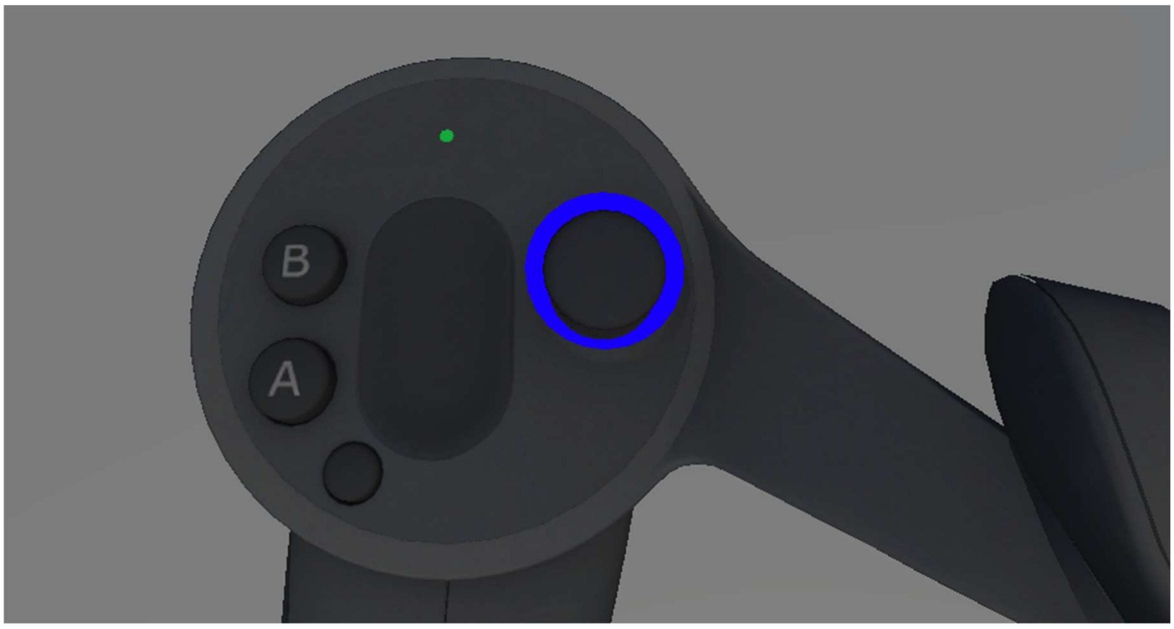


Figure 21. Outlined Valve Index thumbstick.

This Unity shader graph computes the position of the outline of an object by combining the object's position with the normalised object space normal vector multiplied by the desired outline thickness. The "normalise" function ensures that the normal vector has a length of one, which helps ensure that the outline's thickness is consistent across the object's surface. The result of this shader can be seen in Figure 21.

Sometimes other objects can obstruct the highlighted button, or the controller meshes itself. To ensure the player can always see the highlighted controller, we must always render it on top. Also, to avoid confusion if the mesh is still occluded, it should be rendered a solid colour when occluded.

First, another shader was made (Figure 22) that renders the mesh as a solid colour. To make the object always be rendered on top, the URP[31] (Universal Render Pipeline) “Render Objects” feature[32] is used. This feature allows configuring how and when specific layers are rendered.

[31] https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@16.0/manual/index.html

[32] https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@16.0/manual/containers/how-to-custom-effect-render-objects.html

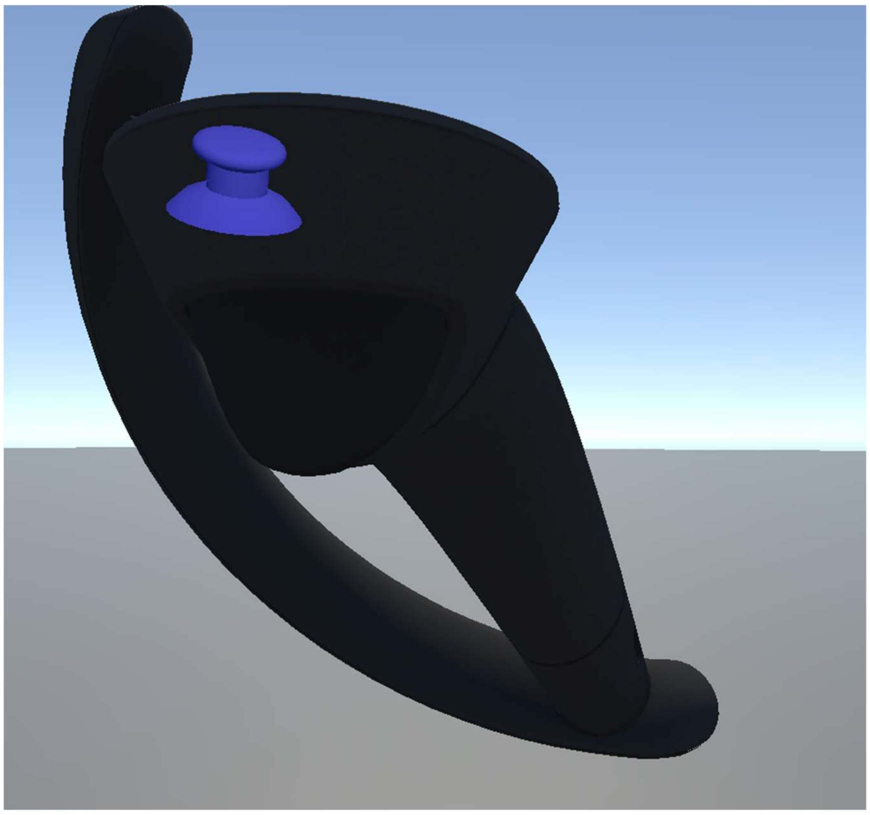


Figure 22. Valve Index thumbstick is visible through the rest of the controller.

A separate layer was made for objects that should be outlined and always on top and were configured not to be rendered by default. Then a new “Render Objects” feature (Figure 23) was added to the configuration and was set only to render the “Outlined Objects” layer.

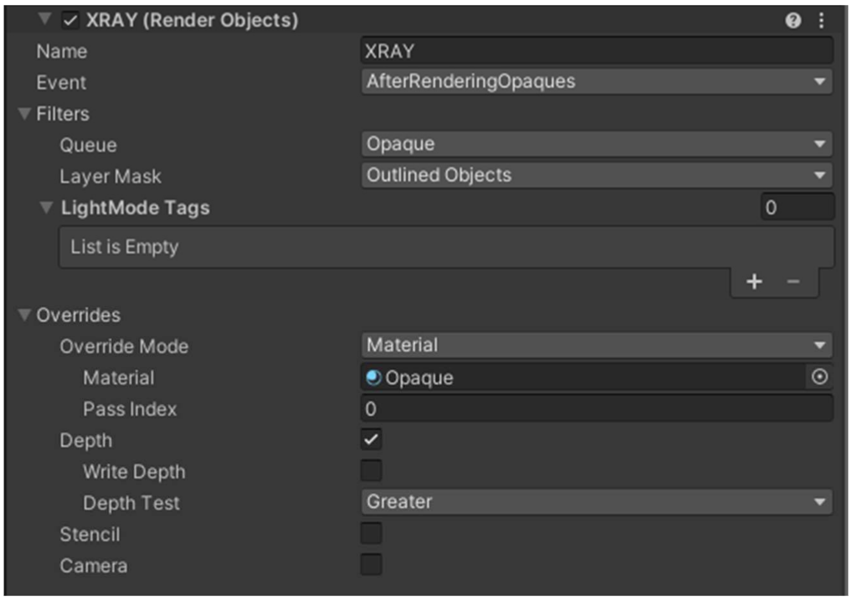
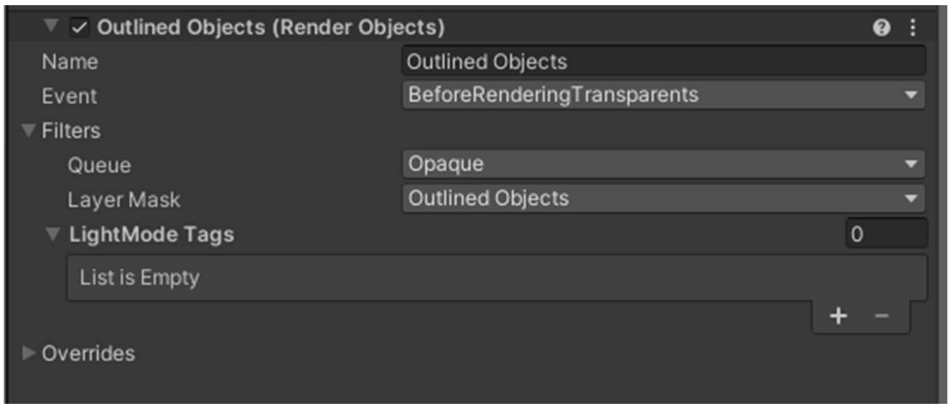

```
▼ ✓ XRAY (Render Objects)
  Name                  XRAY
  Event                 AfterRenderingOpaques
▼ Filters
    Queue               Opaque
    Layer Mask          Outlined Objects
  ▼ LightMode Tags                          0
    List is Empty
                                          + −
▼ Overrides
    Override Mode       Material
      Material          Opaque
      Pass Index        0
    Depth               ✓
      Write Depth       ☐
      Depth Test        Greater
    Stencil             ☐
    Camera              ☐
```

Figure 23. URP pipeline configuration for rendering opaque.

It was also configured to render the layer after rendering most of the scene. Then a new overwrite for the material was configured, setting it to use the solid colour shader. Additionally, it was configured only to render the material when the object has a greater depth than other possible overlapping objects.

However, this only renders the object parts occluded by other objects. Hence, any object in the layer that is not occluded is invisible. To fix this, another "Render Objects" feature (Figure 24) was added and was set to render these objects regularly.

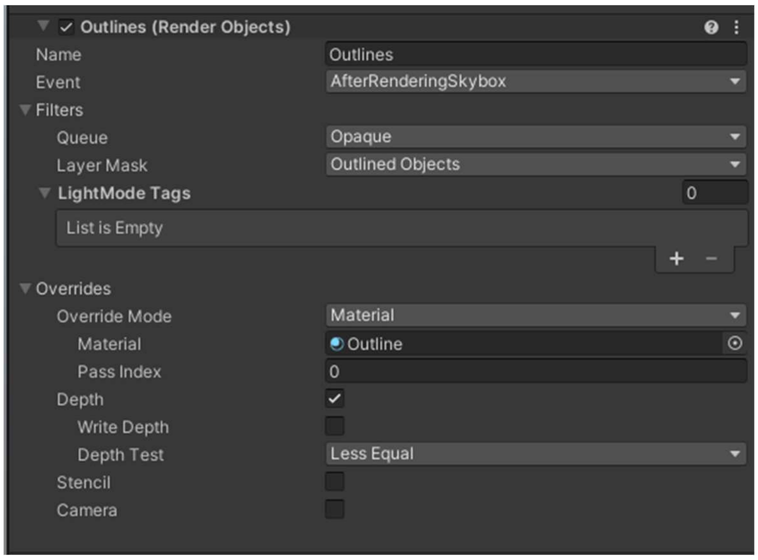


Outlined Objects (Render Objects)

Name	Outlined Objects
Event	BeforeRenderingTransparents
Filters	
Queue	Opaque
Layer Mask	Outlined Objects
LightMode Tags	0
List is Empty	
Overrides	

Figure 24. URP pipeline configuration for rendering outlined objects regularly.

The outline shader can be rendered similarly (Figure 25) but only rendering it when the object is above other objects in the depth buffer. It was added to a later pass of rendering, so anything that could block our outlined object is already rendered.



Outlines (Render Objects)

Name	Outlines
Event	AfterRenderingSkybox
Filters	
Queue	Opaque
Layer Mask	Outlined Objects
LightMode Tags	0
List is Empty	
Overrides	
Override Mode	Material
Material	Outline
Pass Index	0
Depth	✓
Write Depth	
Depth Test	Less Equal
Stencil	
Camera	

Figure 25. URP Render pipeline configuration for rendering outlines.

After URP was configured, the highlighted button was outlined and visible even when occluded. An example of this is shown in Figure 26.

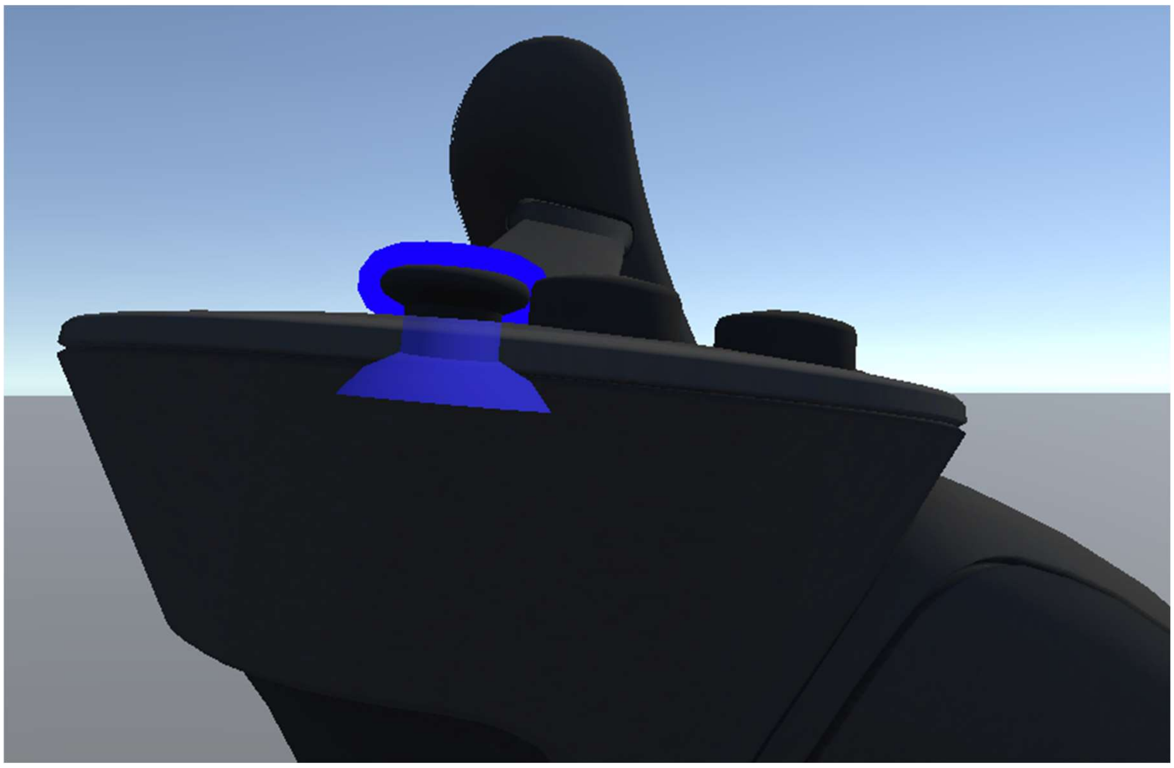


Figure 26. Combination of all pipeline configurations.

Next, the text needs to be shown above the controller. For this, a new prefab called billboard was created. The billboard prefab consists of a canvas and text (Figure 27). The script for the billboard makes the canvas always face the player camera and has methods to show, hide and change the text.

It was first necessary to map all available controls on the model to show a combination of both text and highlight. A new script (Figure 28) was created in the controller prefab that decides which object to highlight and what text to show. The script contains references to different inputs of the controller and the previously added billboard script.

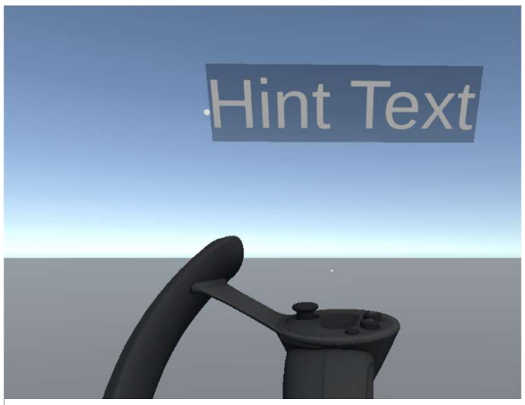
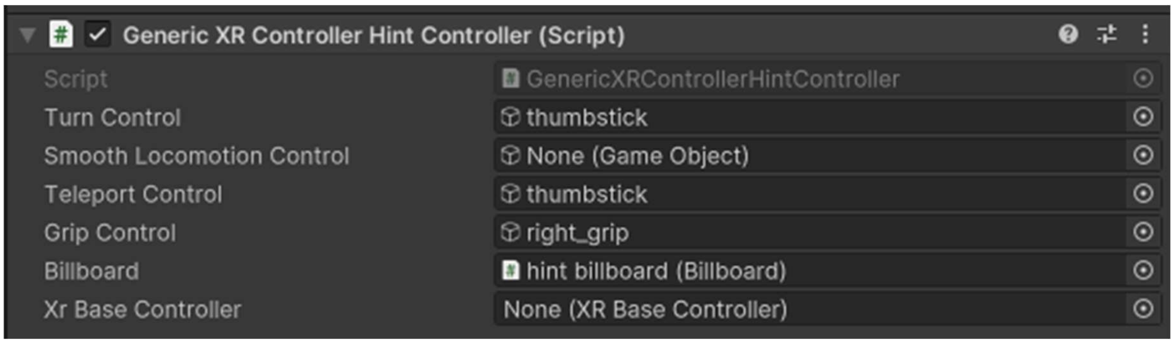


Figure 27. Hint text above the Valve Index controller.

The new `HintController` script has methods to highlight a specific control and display accompanying text. Text is displayed by calling methods in the billboard component, and buttons are highlighted by changing the layer to the outline layer. Additionally, the script has a reference to the controller itself, so when a new hint is shown, it vibrates the controller to draw attention to it.



Generic XR Controller Hint Controller (Script)

Script	GenericXRControllerHintController
Turn Control	thumbstick
Smooth Locomotion Control	None (Game Object)
Teleport Control	thumbstick
Grip Control	right_grip
Billboard	hint billboard (Billboard)
Xr Base Controller	None (XR Base Controller)

Figure 28. Example parameters in the hint controller script.

Now by calling methods on the `HintController` script, hints can be shown on the controller during the tutorial, which is explained in the next subchapter.

3.4.2 Tutorial Flow

The tutorial follows a specific flow to teach the user the game's controls. This is important to ensure that the user learns the controls in a logical and interactive way. Each step of the tutorial flow has a condition to enter and exit.

The tutorial currently has five states: initialising, turning, moving, teleporting, waiting for a grabbable object, grabbing and finished. The first state of the tutorial is initialising. During this step, the tutorial script waits for both user controllers to become visible to the application.

Once the controllers become visible, the tutorial uses the *Observer Pattern* to start listening to actions the player can perform, as in the future states, the script needs to know if the user successfully performs the action requested before moving onto the next state.

Once the action listeners are set up, the tutorial moves onto the Turn state, telling the previously created `HintController` script to show the matching text and button for the turn action. The next state is entered when the script observes that the player successfully turned. The moving, teleporting, and grabbing states behave similarly and are not explained separately.

The waiting for grabbable object state looks for grabbable objects in front of the user. Once it finds it, it highlights it and moves onto the grabbing state. Once the player grabs the object, the highlight is removed.

The Observer Pattern and `HintController` script allow for interactive instruction. At the same time, the condition-based states ensure that the user progresses through the tutorial in a logical and structured manner. By following this flow, users should be able to learn the controls without external assistance. The scripts for the tutorial and hands can be found in the *Assets/_PROJECT/Components/NewHandPresence* folder.

3.5 Virtual Delta Building Changes

To reintroduce the missing floors, they were imported from the original building’s IFC file. Architecture firm Arhitekt11[33] created the IFC file containing the building's models and materials. The IFC file was converted to a Unity-supported format FBX using IfcOpenShell[34]. However, models in IFC files are typically not optimized for game engines and require adjustments for real-time rendering.

The exported models were further optimized using Blender. First, the already existing first and second floors were removed from the export. On the new floors, unnecessary details were removed, overlapping meshes were adjusted or removed, and visible gaps between meshes were filled. After editing the model, the resulting FBX file was imported into the existing Blender file and merged with the rest of the scene. After re-importing the modified Blender file into Unity, the new additions were visible in the game engine (Figure 29).

[33] https://www.architect11.com/en/

[34] https://ifcopenshell.org/

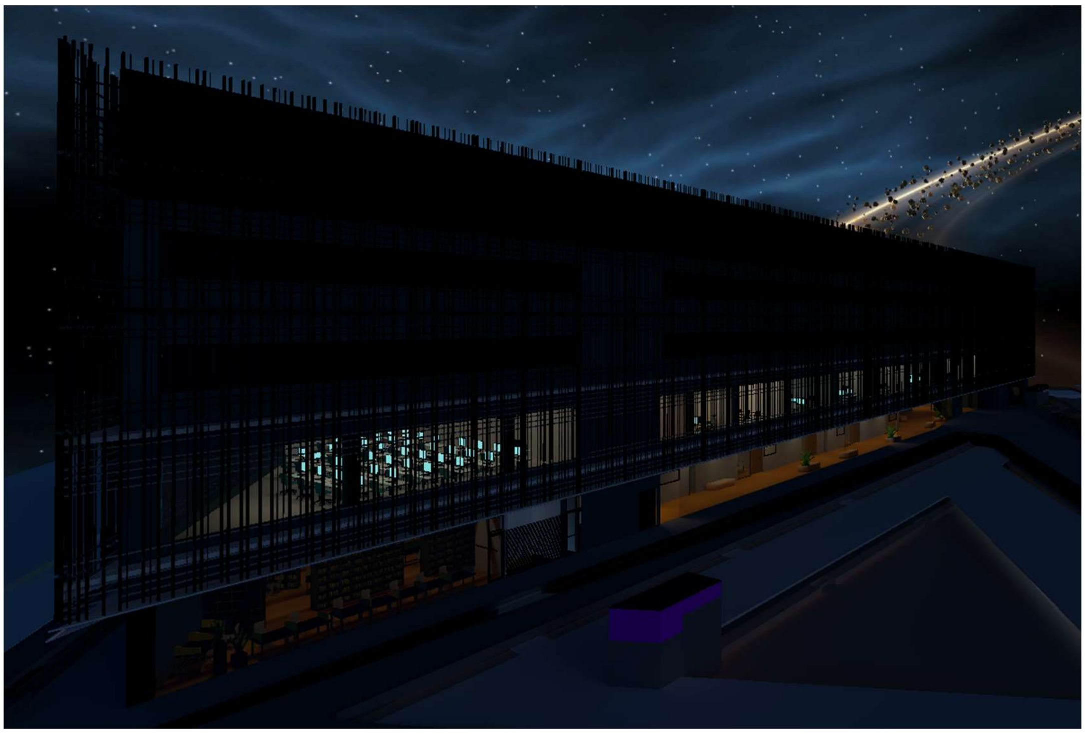


Figure 29. The new version of the virtual Delta Building visible in the Unity game engine.

Further optimization was done in Unity. One option is adjusting lightmapping. Since the new floors do not have lights, they do not require as much detail when pre-computing lighting using light baking. On the new floors, the lightmapping scale was lowered.

Another optimization technique in *DeltaVR* is occlusion culling, which does not render occluded objects, saving rendering costs. Some meshes, like floors and walls, were large and visible throughout the building. To make these meshes more easily cullable, they were split into smaller pieces in Blender.

As a result of these steps, the virtual building now includes all floors and external ribs without a visible impact on performance.

3.6 Spacewalk Experience

As discussed in Chapter 2.7.1, implementing non-Euclidean geometry in *DeltaVR* was an idea to explore. Two variations of this implementation were created. The first was based on Sellis's thesis, which used a virtual camera in the portal's target scene. The camera had to match the player's position in the origin scene to produce the correct image in the portal. Additionally, the image in the portal needed to be cropped to align with the player's view through the portal (2022, pp. 11–14).

Some adjustments were made to the solution, such as sharing the portal mesh between eyes instead of using two different meshes. In this case, the shader checked which eye was rendering the portal and switched the texture accordingly.

Furthermore, instead of having a hard-coded eye distance in the shader code, the new shader code used the eye distance from OpenXR, so it supported any OpenXR-compatible headset. Sellis mentioned a slight stutter with the portal (2022, p. 34), which was resolved by moving the camera matrix calculations to Application.onBeforeRender instead of LateUpdate. This solution can be found in the *Assets/_PROJECT/Components/Portals* folder.

However, even with the adjustments, the performance of this solution was not good enough. Each extra camera added 1-2ms to the time it takes to render a frame (frametime). So, with a pair of portals, the minimum number of cameras needed was 4, which added 4-8ms of extra frametime. Since the ideal frame rate is between 11.1ms and 6.94ms, depending on the headset, this solution is not ideal for *DeltaVR*.

The second variant was developed using the *stencil buffer*. The stencil buffer is a graphics feature that stores per-pixel information. It can be used to mask or selectively render parts of a scene.[35] This approach proved to be much faster, with no visible performance reduction. To implement this method, the Unity Universal Render Pipeline (URP) was used to render objects on a specific layer with a specific stencil value.

A shader was made to draw only the objects with the assigned stencil value. A duplicate of the target scene was created and positioned behind the portal. These copies were placed on a separate layer, making them visible only through the new shader.

As the portals needed to lead to space in our implementation, another shader was developed to mimic the space outside the Delta Building. This shader surrounded the duplicates, effectively creating a visible copy of the portal's target scene. An example of the result is in Figure 30.

[35] https://learn.microsoft.com/en-gb/windows/win32/direct3d9/stencil-buffer-techniques

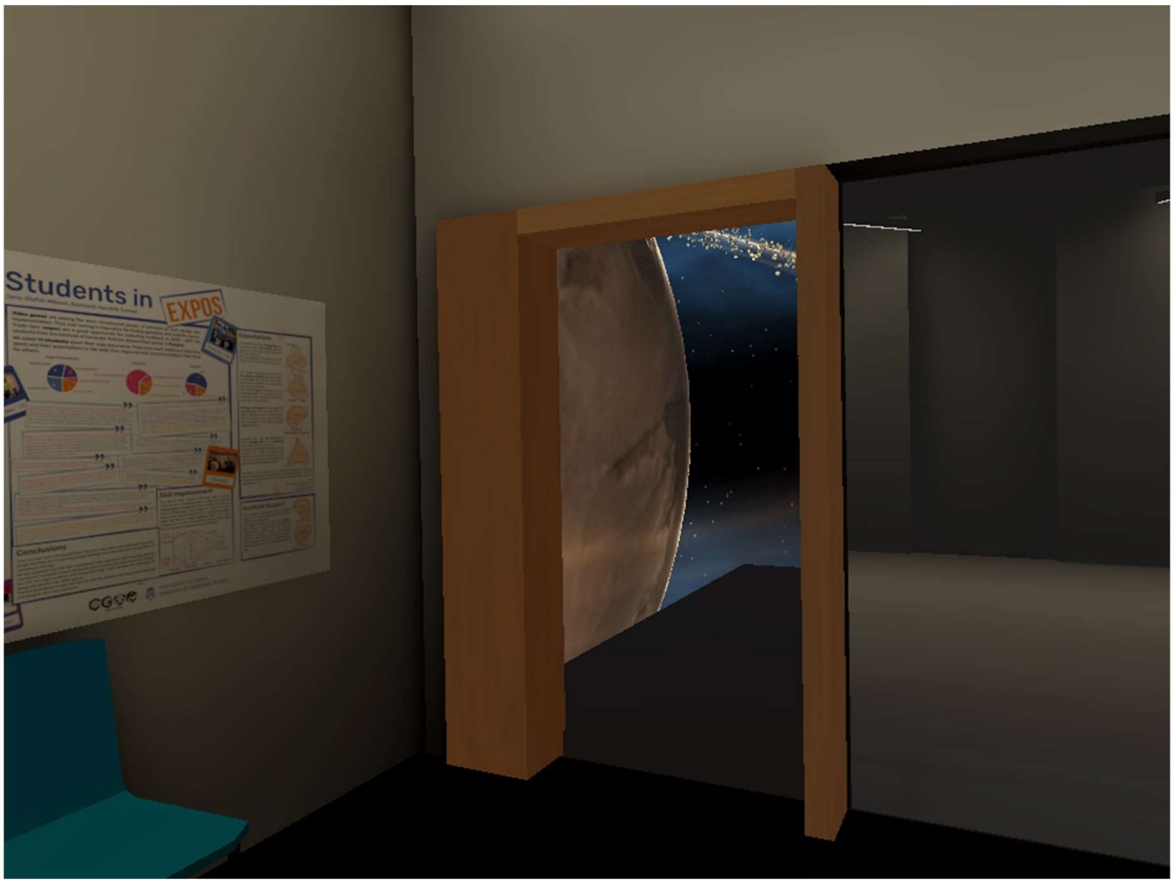


Figure 30. Duplicate of target scene visible through the portal shader.

Additionally, the player must be able to move through the portal using either of the two locomotion methods: teleporting and smooth locomotion. For smooth locomotion, a collider was added to the portal that teleports the player to the other side on collision. For teleporting, the teleport ray was allowed to hit the floor surfaces in the duplicate portal scene. If the player chose to teleport onto the duplicate portal scene floor, it would redirect the teleport to the linked portal destination.

This solution, however, has some drawbacks. First, the visible part of the target scene needs to fit behind the portal. Otherwise, the player can see the actual geometry behind the portal, breaking the illusion.

Secondly, creating a copy of the target scene is necessary, which could be time-consuming during development. The copy also needed to be updated whenever the portal's target scene was edited. To streamline this process, a script was created to automatically duplicate visible objects from the other side of the portal.

However, lighting information was lost when copying parts of the Delta Building. Re-baking lighting in the cloned scene was not feasible, as it lacked the complete geometry and would produce different results. To address this issue, an additional script called *`LightmapSync`* was created.

The *`LightmapSync`* component stores a reference to the original geometry and copies its lighting information during runtime. This ensures that the cloned scene always has up-to-date lighting information. The script works during runtime because Unity changes textures at runtime, and misalignment could occur if set earlier. Additionally, Unity does not provide a documented API to read or write scene lightmap data.

The script copied the original object's location for objects that used light probes instead of baked lighting. Then the light probes took the lighting information from that point during runtime. An example of synced lighting is visible in Figure 31 and Figure 32.

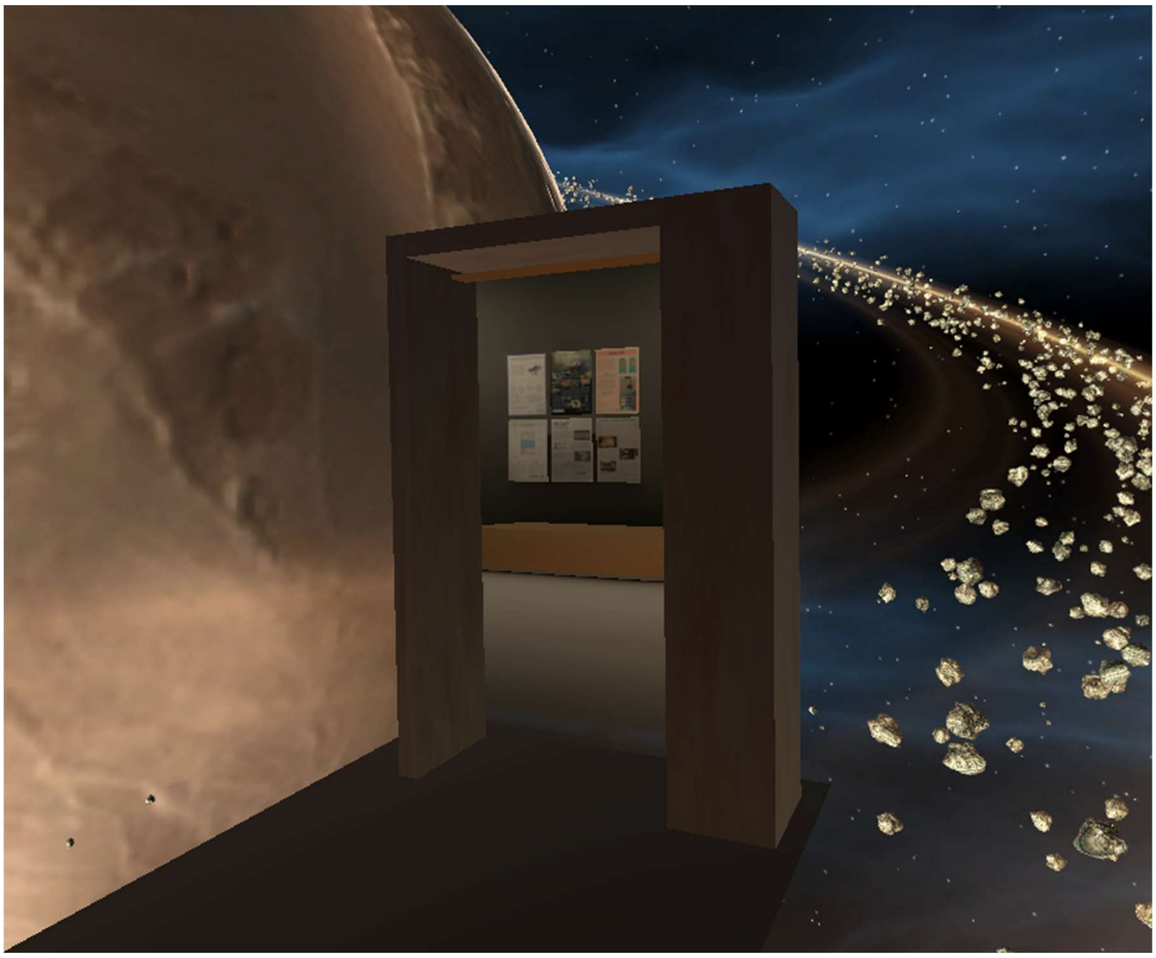


Figure 31. Delta Building hallway with synced lighting as seen through the shader.

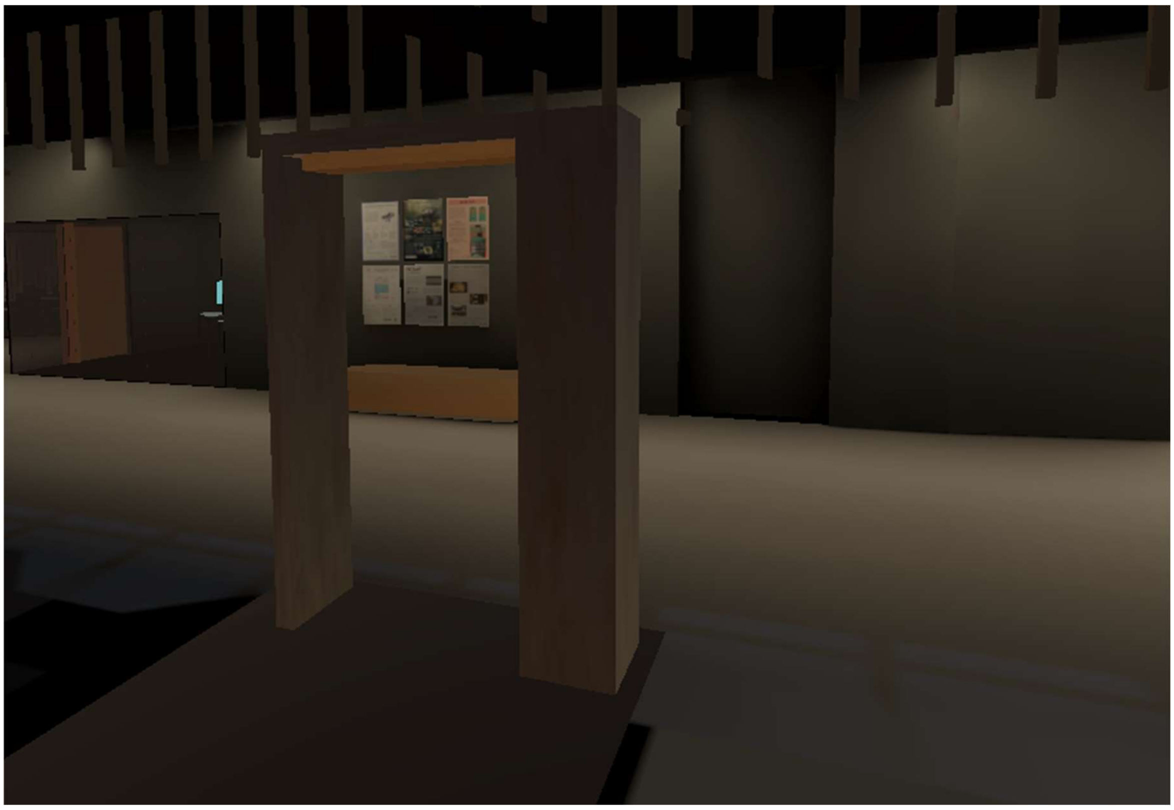


Figure 32. Portal Scene without stencilling.

Combining these approaches ensured lighting remained synchronised between the original and cloned scenes. The second variant can be found in the following folder: *Assets/_PROJECT/Components/Portals2*.

3.7 Multiplayer with Fish-Net

Chapter 2.1 explains that *DeltaVR Multiplayer 2.0* will utilize the Fish-Net library for networking. The first step in implementing multiplayer was synchronizing players. Each player has their own *XR Rig* prefab, containing all logic and game objects for a VR-based player. However, if every player used the same rig with the same scripts over the network, multiple instances would conflict.

To address this issue, a new prefab called *`XRNetworkPlayer`* was created, containing a script called *`NetworkPlayer`*. This prefab is instantiated whenever a player connects to the server across all clients and the server. The *`NetworkPlayer`* script checks for player ownership in Fish-Net, because ownership determines who controls an object's actions.

If the `NetworkPlayer` script detects ownership, it locally instantiates the XR Rig, meaning each client only creates their own XR Rig. This also means that players cannot see others now, as only their own XR Rig is created.

To remedy this, another prefab with a script called `XRPlayerMirror` was created. This prefab is instantiated regardless of client ownership, but only one client owns the component. This prefab is a simple mirror of the complete XR Rig, containing only the player's head and hand game objects without additional scripting. The `XRPlayerMirror` script checks for ownership and, if found, sets the mirror head and hands' positions to match the ones in the XR Rig.

To ensure all players see the mirror in the same position, the owner must send the game objects' positions and rotations to the server. Fish-Net's *NetworkTransform*[36] component is used for this purpose. Adding these components to the player's head and hands automatically synchronises their positions and rotations over the network.

Furthermore, since our hands have animations, the owner must synchronize them. To achieve this, we use Fish-Net's *NetworkAnimator*[37] component. This component ensures that all animations are synced across the network.

With these implementations, players and their movements became visible across the network. However, the rest of the scene, including experiences, remained unsynchronized. To address this issue, a new script called `XROwnershipRequester` was created (Figure 33).

When a player grabs an object in the scene, the `XROwnershipRequester` script checks if it has a NetworkTransform component. If it does, ownership of the GameObject is requested from the server through an RPC (Remote Procedure Call). An RPC is a communication method that enables a client to request the server to perform an action or process on its behalf.

[36] https://fish-networking.gitbook.io/docs/manual/components/network-transform

[37] https://fish-networking.gitbook.io/docs/manual/components/network-animator

Once the server grants ownership, the player can move the GameObject with their hands, and its position will be synchronized. When the player releases the object, the script sends another RPC to the server to relinquish ownership, allowing other players to grab it.

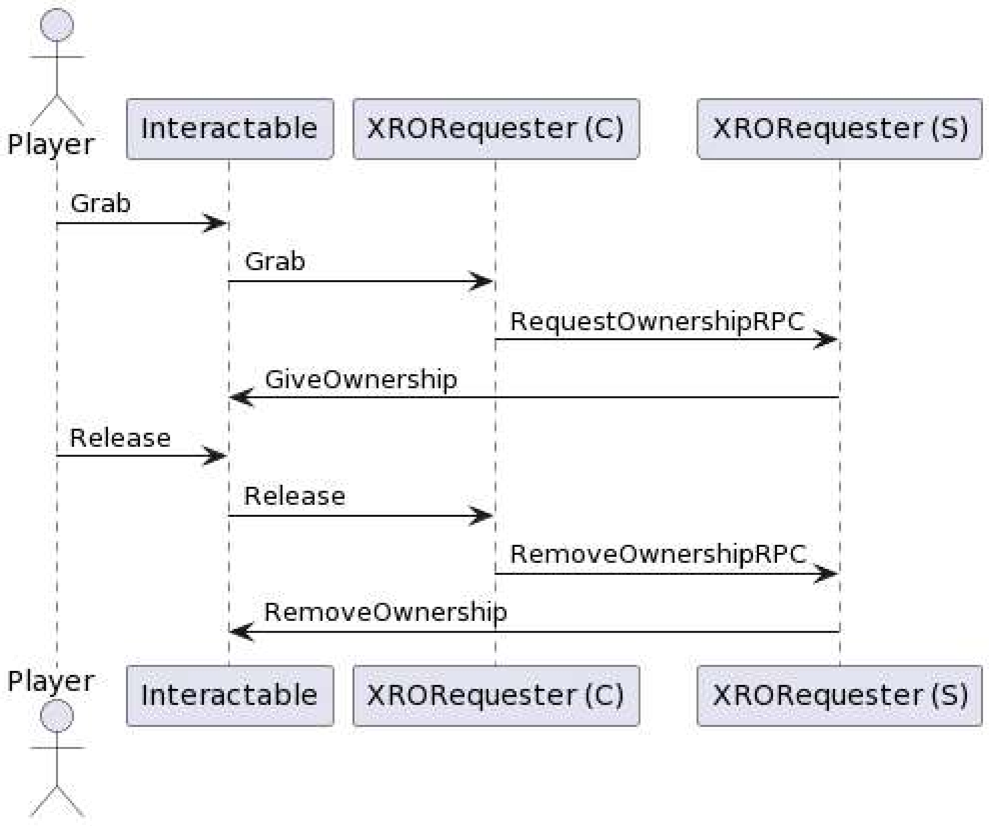


Figure 33. Sequence diagram for XROwnershipRequester.

After adding the NetworkTransform component to every interactable prefab, this script ensures that simpler interactables stay synchronized across the network. However, more complex objects may require additional logic.

3.8 Drawing Experience

As discussed in Chapter 2.7.3, a drawing experience was planned. Two prefabs were created for implementation: *DrawableSurface* and *SprayGun*. A sequence diagram of their networking is given in Figure 34.

DrawableSurface consists of two main scripts: `TextureDrawing` and `NetworkDrawingManager`. `TextureDrawing` updates the texture on the prefab's mesh, which is stored in a *Texture2D*. It has a method for drawing on the texture with parameters for coordinates, colour, and size, and it updates the mesh's texture accordingly.

`NetworkDrawingManager` syncs the Texture2D across the network. When a client connects, the server sends the current Texture2D, ensuring new players see the same drawing. It also contains RPCs for syncing player drawings. A separate script was made to serialise and deserialize Texture2D, converting it into a PNG byte array for sending over the network.

The SprayGun prefab includes a SprayGun mesh and spraying effects and can be grabbed by players. The prefab has its own script, checking if the player is holding it and pressing the trigger. If so, it shoots a raycast to detect the DrawableSurface prefab. If hit, it instructs the `NetworkDrawingManager` to send an RPC based on the player's distance and the SprayGun's colour. The RPC is then sent to all clients, whom all update their textures.

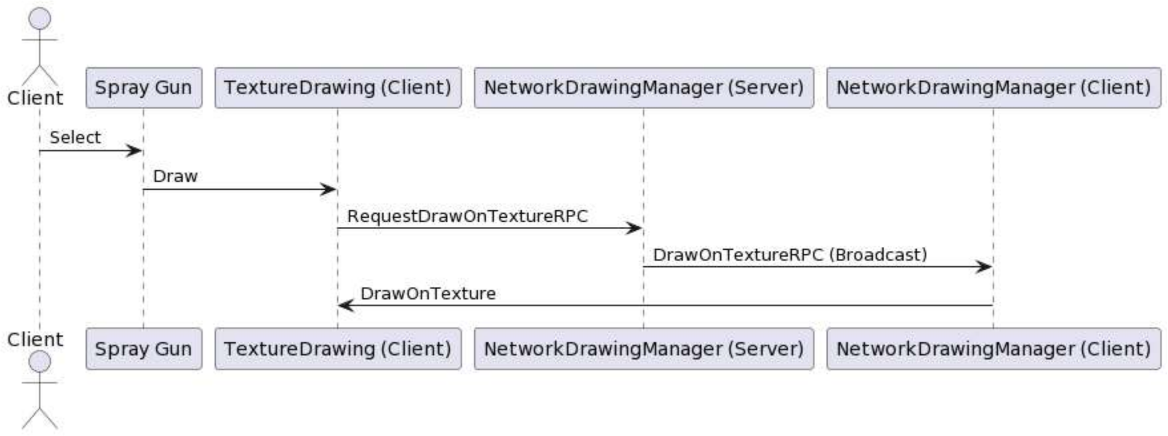

Client
Spray Gun
TextureDrawing (Client)
NetworkDrawingManager (Server)
NetworkDrawingManager (Client)
Select
Draw
RequestDrawOnTextureRPC
DrawOnTextureRPC (Broadcast)
DrawOnTexture

Figure 34. Sequence diagram for Drawing experience networking.

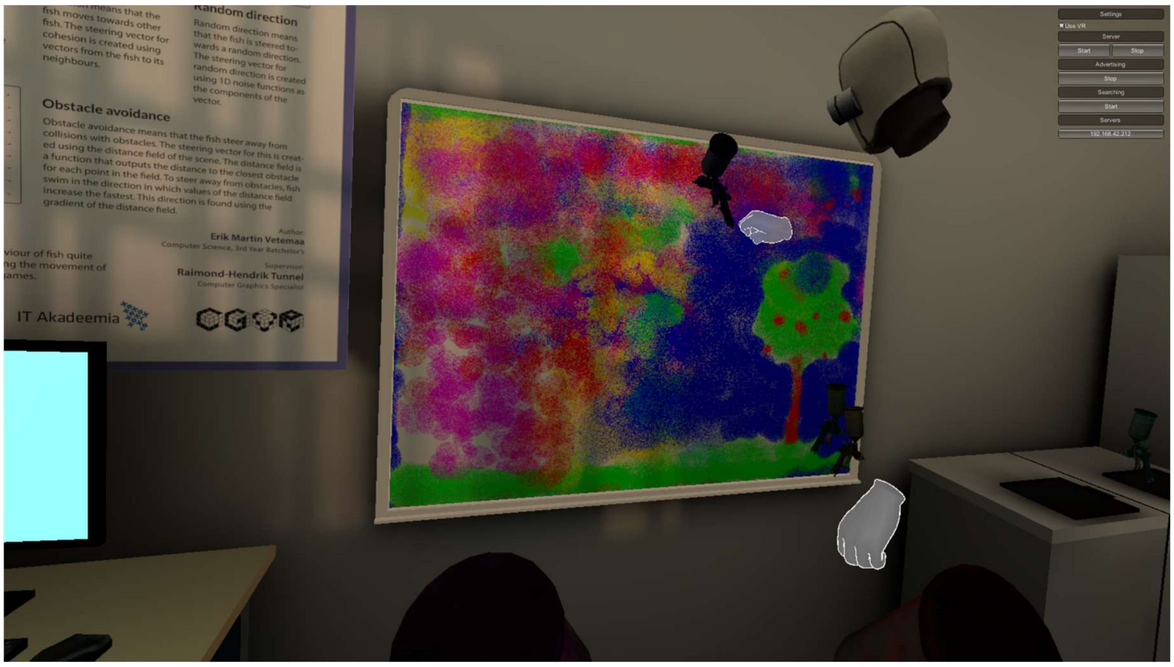


Figure 35. Playtesters drawing on the canvas.

These prefabs and components create a fully networked drawing experience (Figure 35). The code can be found in the following folder*: Assets/_PROJECT/Components/Drawing*.

3.9 Archery Range Experience

The archery range already had some networking code in the previous multiplayer thesis. Although this code used RPCs for network syncing and porting it to Fish-Net would have been relatively simple, the bow needed to be re-implemented from scratch for two reasons.

First, the existing code contained critical bugs that prevented it from functioning correctly over the network. Second, the XR Interaction Toolkit it was built on had been updated from version 0.9 pre-release to 2.3, resulting in numerous changes to method signatures and implementations, making the previous code incompatible.

The main bow logic was implemented similarly to the original single-player *DeltaVR* thesis, with updates to accommodate the new toolkit methods. The primary change in this implementation concerns arrow spawning. In the original version, players had to pick up arrows and place them in the bow. With the new implementation, the server spawns and positions the arrows in the bow for the players.

Although the universal `XROwnershipRequester` script synchronised the bow ownership over the network, the arrows within it were not. A new script was added to the bow, which checks for ownership changes. When a client recognizes that the ownership has shifted to them, they also send out an RPC requesting ownership of the arrow (Figure 36).

Furthermore, the arrow is connected to the bow via *XRSocketInteractor*[38]. The XRSocketInteractor is an Interaction Toolkit component that can grab objects like player hands. However, XRSocketInteractor does not have to be attached to a controller. If the client does not know that the arrow is socketed in the interactor, it will not follow the bow once the client gets ownership. Therefore, this state was also communicated to the client when giving it ownership (Figure 36).

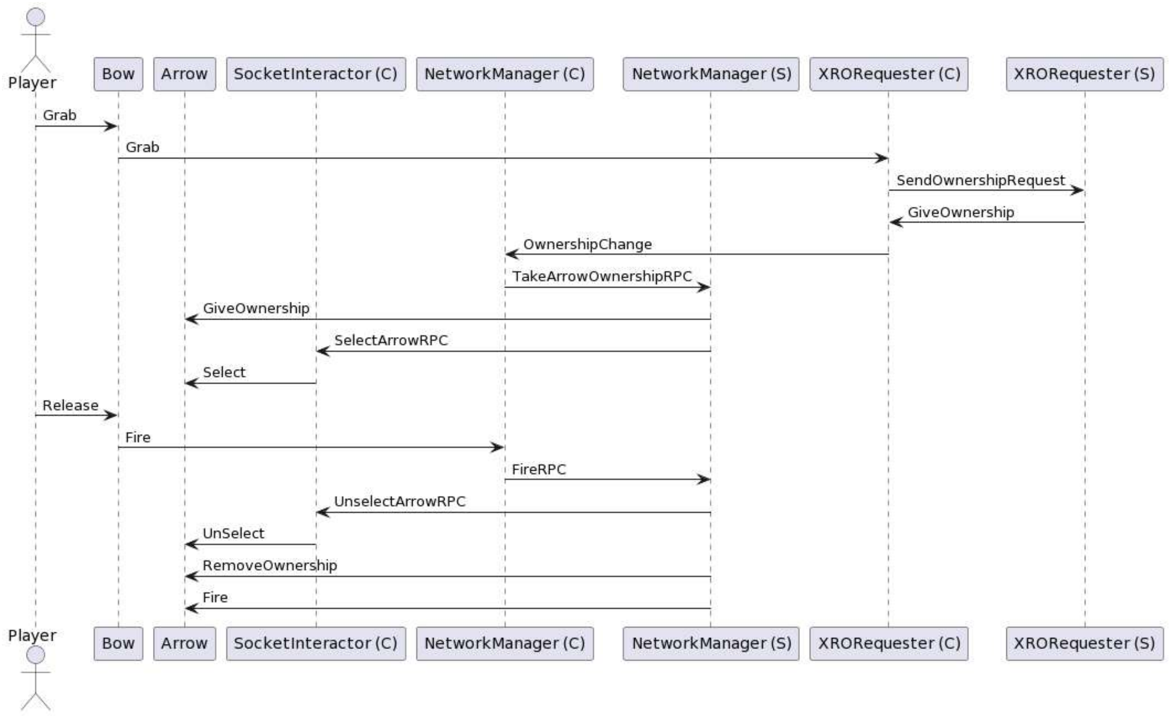


Figure 36. Sequence diagram for bow and arrow networking.

Additionally, modifications were necessary for the archery target scripts. The server, rather than clients, now spawns targets, and target movement logic is executed solely on the server. Additionally, separate RPCs were introduced to update the scoreboard, high score, and time left indicators across all clients.

[38] https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/manual/xr-socket-interactor.html

With these implementations in place, the archery experience was networked. This version of the archery range is in the *Assets/_PROJECT/Components/Bow* folder.

4 Testing

The goal of the thesis was to improve upon the previous thesis. To determine if this goal was achieved, it is essential to conduct testing. Subchapter 4.1 showcases an in-development version at an expo and describes some issues found and how they were fixed. Subchapter 4.2 describes how the application was performance tested and if it is better than the previous iteration. Subchapter 4.3 describes usability testing with actual users, their opinions on the application, and bugs found.

4.1 University of Tartu Alumni Day Expo

On March 18, 2023, the University of Tartu held an alumni day event[39] where a development version of *DeltaVR Multiplayer 2.0* was showcased (Figure 37). The demonstration occurred in the CGVR Lab (Room 2007) from 11:00 am to 6:00 pm, with approximately 15 people trying the application. Several issues with this version of the application were identified throughout the event.

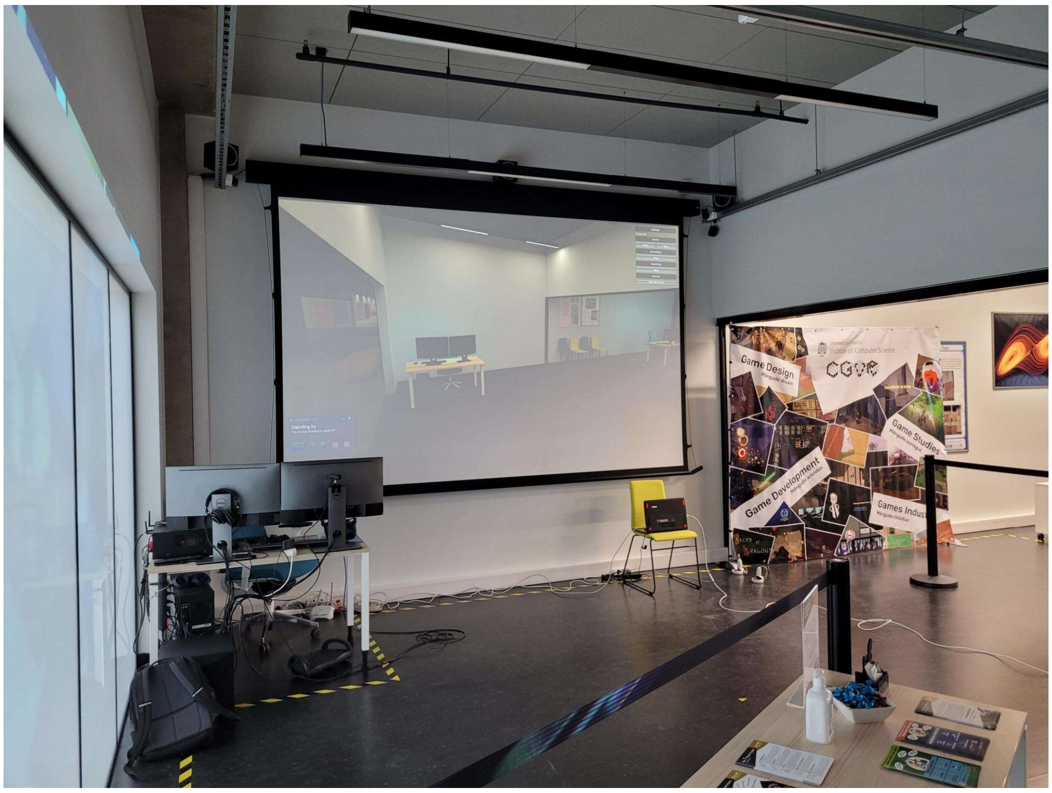


Figure 37. DeltaVR Multiplayer 2.0 setup at Alumni Day.

[39] https://cs.ut.ee/et/vilistlaspaev

4.1.1 Application Crashing

The most significant issue discovered was that the application would crash after just a few minutes, which had not been encountered during development. This problem was particularly troublesome when multiple users attempted to play consecutively, as the application had to be restarted pre-emptively to prevent crashes during gameplay.

Determining the cause of this error proved challenging, as the log files provided no valuable information. The log file cited "Attempt to access invalid address" as the reason for the crash. According to a Unity employee, this message indicates that Unity's crash handler could not identify the underlying cause.

To identify the cause of the crash, the Post-Mortem/Forensic Debugging guide[40] in Unity's documentation was followed. This process involved forcing Windows (the operating system) to generate a full memory dump when a crash occurred. The Windows Registry was modified according to Microsoft's guide[41] to achieve this. Then the full memory dump was opened using the Windows Debugger (WinDbg). After adding the symbol files mentioned in Unity's documentation to WinDbg, the debugger displayed the exact method names where the application crashed (Figure 33).

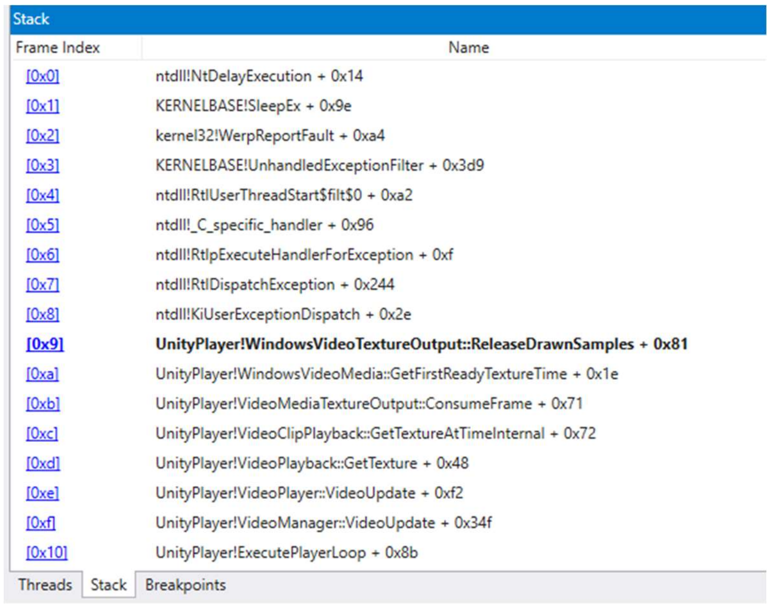

```
Stack
Frame Index   Name
[0x0]         ntdll!NtDelayExecution + 0x14
[0x1]         KERNELBASE!SleepEx + 0x9e
[0x2]         kernel32!WerpReportFault + 0xa4
[0x3]         KERNELBASE!UnhandledExceptionFilter + 0x3d9
[0x4]         ntdll!RtlUserThreadStart$filt$0 + 0xa2
[0x5]         ntdll!_C_specific_handler + 0x96
[0x6]         ntdll!RtlpExecuteHandlerForException + 0xf
[0x7]         ntdll!RtlDispatchException + 0x244
[0x8]         ntdll!KiUserExceptionDispatch + 0x2e
[0x9]         UnityPlayer!WindowsVideoTextureOutput::ReleaseDrawnSamples + 0x81
[0xa]         UnityPlayer!WindowsVideoMedia::GetFirstReadyTextureTime + 0x1e
[0xb]         UnityPlayer!VideoMediaTextureOutput::ConsumeFrame + 0x71
[0xc]         UnityPlayer!VideoClipPlayback::GetTextureAtTimeInternal + 0x72
[0xd]         UnityPlayer!VideoPlayback::GetTexture + 0x48
[0xe]         UnityPlayer!VideoPlayer::VideoUpdate + 0xf2
[0xf]         UnityPlayer!VideoManager::VideoUpdate + 0x34f
[0x10]        UnityPlayer!ExecutePlayerLoop + 0x8b
```

Figure 38. Stack trace from WinDbg tool.

[40] https://docs.unity3d.com/2019.4/Documentation/Manual/WindowsDebugging.html

[41] https://learn.microsoft.com/en-us/windows/win32/wer/collecting-user-mode-dumps

Investigating the specific stack trace and method names led to an issue[42] on the Unity Issue Tracker. This issue revealed that when a Video Player component in the application loops a video, it has a chance of causing a crash. As a temporary workaround, all video players in the scene were disabled. At the time of writing this thesis, this bug remains unresolved, and the video players are still turned off.

4.1.2 Tutorial Issues

In the application's newer version, the tutorial's first iteration was introduced. Observing people playing and gathering feedback revealed several usability issues with this iteration. Only a few individuals managed to complete the tutorial without external help.

The first issue was that players did not notice the tutorial. The tutorial text was attached to the controller and had a hard-to-notice dark blue colour. If players did notice the tutorial or had it pointed out to them, they struggled to understand the text labels. For example, when asked to move the joystick forward, they moved the entire controller forward instead. Although the joystick on the controller was outlined, players did not notice it or make the connection.

Two changes (Figure 39) were made after the expo to address these usability issues. The text colour was changed from dark blue to safety orange, often used to distinguish objects from their surroundings[43].

[42] https://issuetracker.unity3d.com/issues/crash-on-windowsvideomedia-getfirstreadytexturetime-when-focusing-gameobject-in-tutorial

[43] https://en.wikipedia.org/wiki/Safety_orange

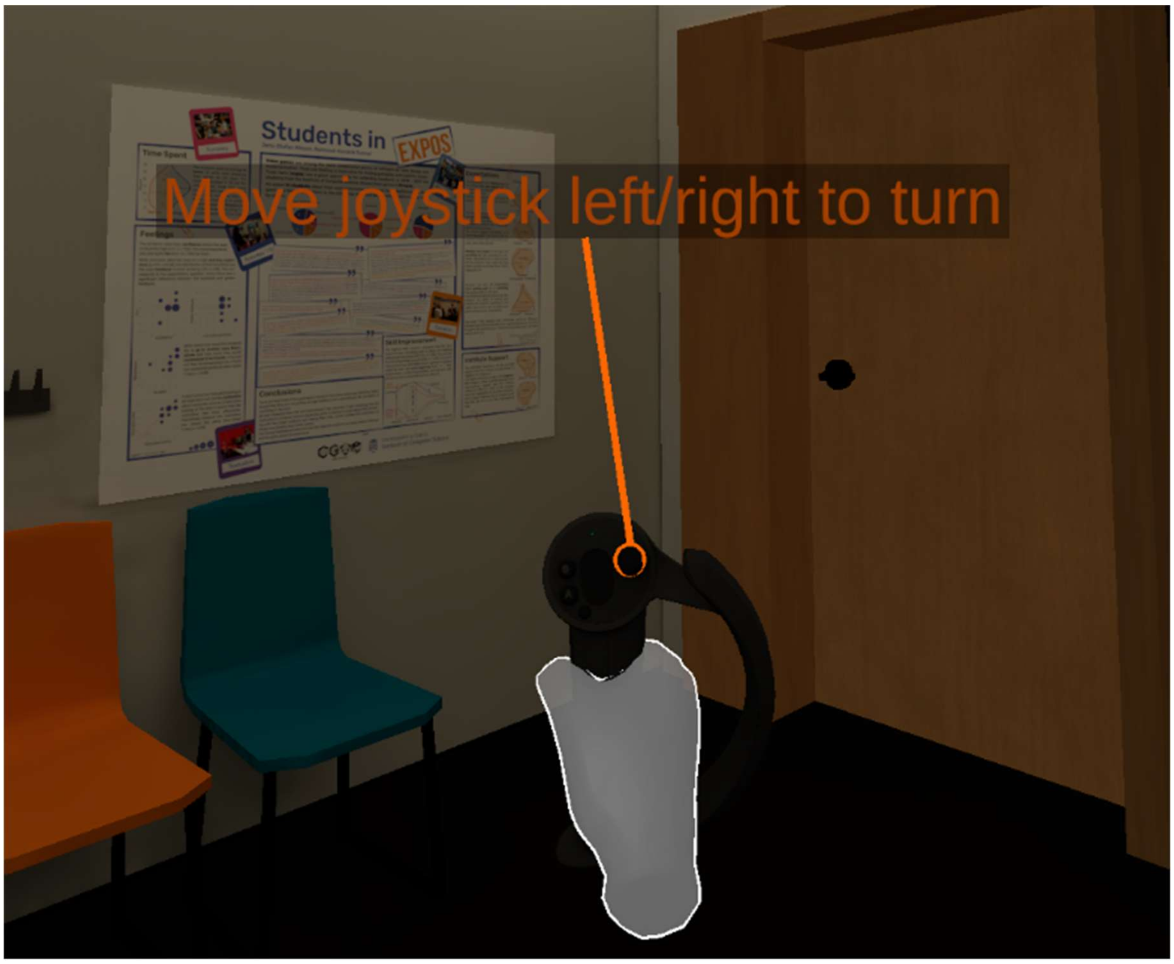


Figure 39. Improved tutorial, with a different colour and a line between text and control.

Additionally, a line was added between the text and the highlighted control. This line aims to help users understand which button the tutorial refers to. Even if the user is unfamiliar with the term "joystick," the combination of the line and highlight should help them make the connection.

4.1.3 Archery Range Issues

The updated application also featured a new version of the archery range mini-game. None of the network issues mentioned in Chapter 2.1.1 occurred during the expo. However, users still struggled to use the bow in some instances.

One problem was grabbing the bow. The bow has two small grabbable colliders on both sides, and players often miss these colliders or grab the wrong one. Additionally, aiming the bow could only be done by moving the hand that grabbed the front of the bow, which felt unnatural for some players.

To address these issues, the bow prefab was redesigned. After upgrading the XR Interaction Toolkit to version 2.3, support for multiple grabbing points on the same collider became available. Using this new feature, the two smaller colliders were replaced with one large collider that covers the entire bow (Figure 40). By setting the grabbing points to the original smaller collider positions, players would always grab the front of the bow before the back of the bow. The bow scripts were refactored to accommodate these changes.

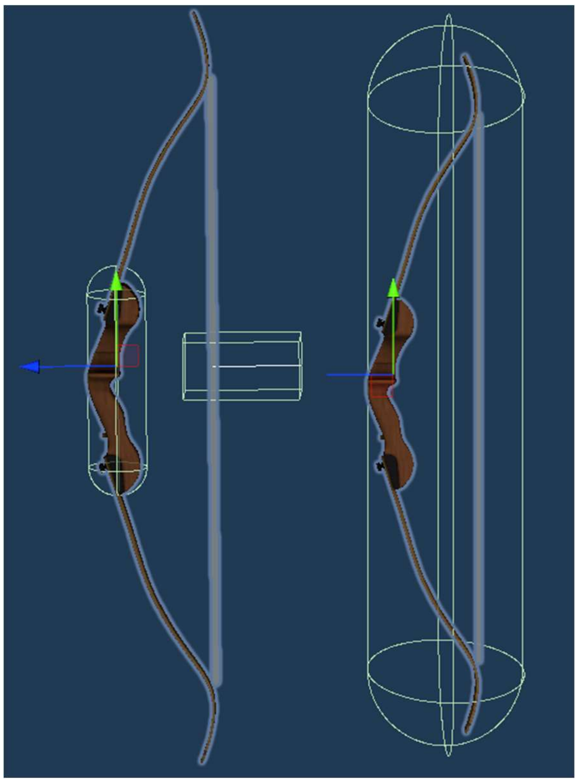


Figure 40. Old bow (left) with two smaller colliders and new bow (right) with one large collider. Colliders are shown in green.

This modification also allows players to aim the bow with both hands since they now share a collider, and moving either hand moves the bow accordingly. This version of the bow can be found in the following folder: *Assets/_PROJECT/Components/TwoHandedBow*.

4.2 Performance Testing

During this thesis, one of the primary additions was incorporating many missing details into the building. The new version now includes all floors, unlike the previous version, which only had two. Furthermore, the addition of multiplayer and other experiences increases performance overhead. As a result, it is crucial to ensure that performance has not dropped below the minimum threshold.

To assess the application's performance, we can measure frame time, the amount of time it takes for the computer to generate a frame[44]. The ideal maximum frame time depends on the headset and screen, typically between 11.1ms and 6.94ms.

The previous multiplayer thesis did not include performance testing on a PC, but the first *DeltaVR* thesis did. In that thesis, the performance was tested on the Vive Pro with a target of 11.1ms. The computer used was CGVR-Torrance[45] in the CGVR Lab. The hardware inside that computer remains the same as it was in the original thesis. Therefore, the performance results should be comparable by testing on the same computer and with the same headset.

The first thesis measured frame time during an entire game session, where a player navigated most of the building and played all experiences. In order to get a fair measurement, testing *DeltaVR Multiplayer 2.0* followed the same pattern.

To measure the frametime, SteamVR's Frame Timing tool was used. The result of the frametime recordings was saved into a CSV file by following the tutorial on Valve's Developer Community wiki[46].

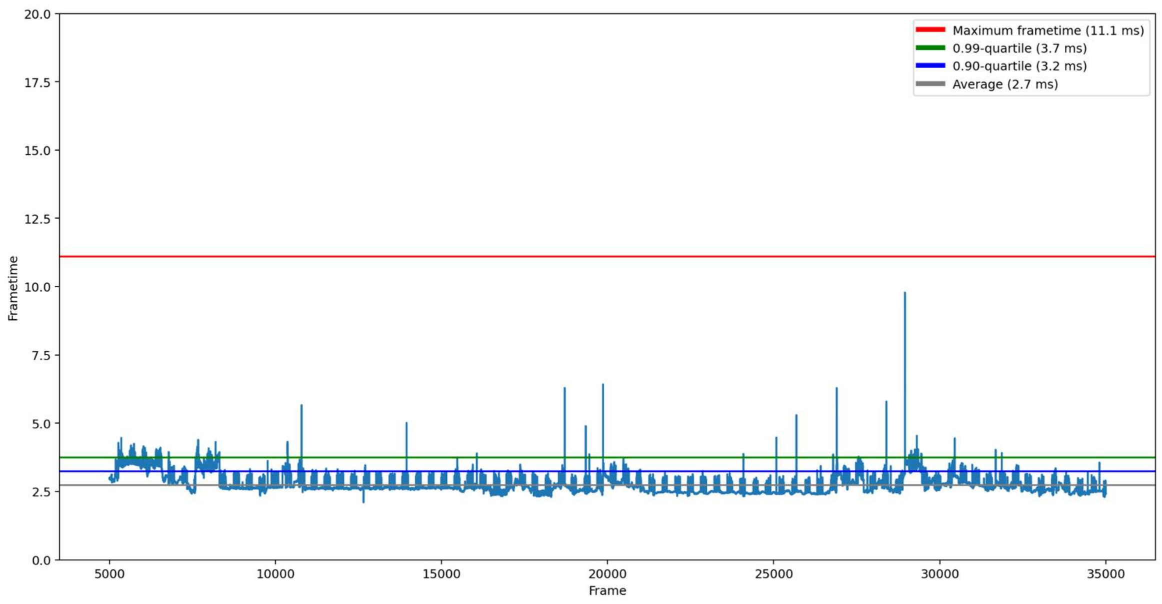


Figure 41. Frametime during a test session.

[44] https://cgvr.cs.ut.ee/frame-rate-vs-frame-time/

[45] https://cgvr.cs.ut.ee/inventory/

[46] https://developer.valvesoftware.com/wiki/SteamVR/Frame_Timing

The results of the recording are displayed in Figure 41. All the frames stayed below our maximum frametime; therefore, the performance remains acceptable.

Table 2. Comparison of frametimes of the original and new version.

	DeltaVR	DeltaVR Multiplayer 2.0	Difference
Average	6.54ms	2.7ms	2.42x
0.90-quartile	7.82ms	3.2ms	2.44x
0.99-quartile	8.94ms	3.7ms	2.41x

Table 2 compares the original *DeltaVR* frametimes against *DeltaVR Multiplayer 2.0* frametimes. The table shows that the newer iteration of *DeltaVR* runs around 2.4 times faster than its predecessor. This is a very positive result considering the additional detail added to the virtual Delta Building during this thesis.

Additionally, the application should perform better on lower-end hardware, such as the expo laptop mentioned in Chapter 1.4.1. On higher-end hardware, the available performance could be used to render the application at a higher resolution or increase anti-aliasing[47] to improve the look of the application.

4.3 Usability Testing

Usability testing involves testing the application with real users to evaluate its ease of use and identify any problems affecting the user's experience. This process helps to improve the product's overall usability. (Moran, 2019)

While feedback about this iteration of DeltaVR has already been gathered at expos, observing people's reactions to the application at expos may not provide as much specific feedback or identify issues as effectively.

According to Jerald, this is because people at expos may not have the time, patience, or privacy to provide detailed feedback or identify issues as they would in a controlled environment like a usability testing session. They may also be more likely to hold back

[47] https://learnopengl.com/Advanced-OpenGL/Anti-Aliasing

criticism or give positive feedback to avoid being perceived as negative or critical in a public setting. (2015, pp. 436–437)

Therefore, conducting usability testing with users in a controlled environment would likely provide more detailed and reliable feedback, which can help to improve DeltaVR's user experience further. To gather feedback from the testers, a questionnaire is required.

4.3.1 Feedback Questionnaire

The feedback questionnaire is based on the example questionnaire provided by Jerald. The questionnaire consists of four parts: a simulator sickness questionnaire, a Likert scale, a background and experience questionnaire, and some open-ended questions. (2015, pp. 489–493) The full questionnaire can be found in Appendix IV.

The first part is the Kennedy Simulator Sickness Questionnaire (SSQ). It contains 16 symptoms to rate from none to severe. (2015, pp. 195–196) These symptoms cover three categories: oculomotor, disorientation, and nausea. Analysing the results of these three categories will show if and how our application makes users sick. It is important to mention that neither previous theses about DeltaVR measured sickness, so there is no baseline to see if our locomotion improvements made a difference.

Additionally, the user is asked to only fill out the SSQ after testing the application. While having accurate information on how the testers felt before playing the application would be good for comparison, Young et al. have shown that taking a questionnaire before can make the player sicker (2007, p. 427).

The second part of the questionnaire is the Likert scale. According to Bhandari, The Likert scale measures the user's opinion of a statement or a question. The Likert scale should be used to measure the opinion on a greater scale than just “yes and no”. Additionally, they should be used to measure unobservable results. (2020)

Our scale asks the user to rate statements from “Strongly disagree” to “Strongly agree” with no neutral opinion. The neutral opinion is replaced with “Not relevant” if the player does not see or interact with whatever the statement says.

This questionnaire's Likert scale is split into six sections: the tutorial, spacewalk, archery, drawing, application in general, and the Delta Building. With the tutorial, spacewalk, archery, and drawing, the questionnaire measures four things: visibility, clarity, usability, and if it achieves its goal.

For example, with the tutorial, the questionnaire measures:

1) Visibility: Did the player notice the tutorial;
2) Clarity: Did the player understand what the tutorial is telling the player to do;
3) Usability: Did the player successfully do what the tutorial asked;
4) Goal: Did the player successfully learn the controls.

Additionally, there is a statement about the controls themselves being good, as the tutorial might be fine; however, the applications control scheme might be bad. The Likert scale should be used to measure these because they are not always observable. For example, the player might notice the tutorial but ignore it.

The questions are also in the order that the player interacts with that specific experience. This should show where the first point of failure was if the player had a negative opinion of the experience.

The final two sections of the Likert scale part are about the application in general and the Delta Building. The application part measures the player's opinion on the whole application. Additionally, since the building was improved in this thesis, the building part measures players' opinions on how realistic the Delta Building looks and if they like its appearance.

The third part of the questionnaire is about background and experience. This part contains questions about how often the player uses a computer and how experienced they are with video games and virtual reality. Those familiar with virtual reality and video games may find it easier to use the controls and experience less sickness. Therefore, this information is required to determine if the tutorial, locomotion, and other usability changes help newer players use the application and avoid sickness.

The fourth and last part of the questionnaire is open-ended questions. Open-ended questions can provide information from the users that were not expected in the Likert scale or give them a chance to explain their opinion. In this part, the questionnaire asks about the player's likes and dislikes about the application, experiences, and the building. Additionally, it lets the player fill out any other comments they might have in their mind that was not covered by the previous questions.

4.3.2 Usability Testing Results

Usability testing was conducted from the 14th of April to the 28th of April 2023. Users for usability testing were gathered by sending invitations (see Appendix III) to participate in testing in the University of Tartu Computer Science students mailing list. The mailing list had around 1300 users when sending the e-mail. Out of those, 23 students showed interest in testing. Out of the students who showed interest, 20 were tested.

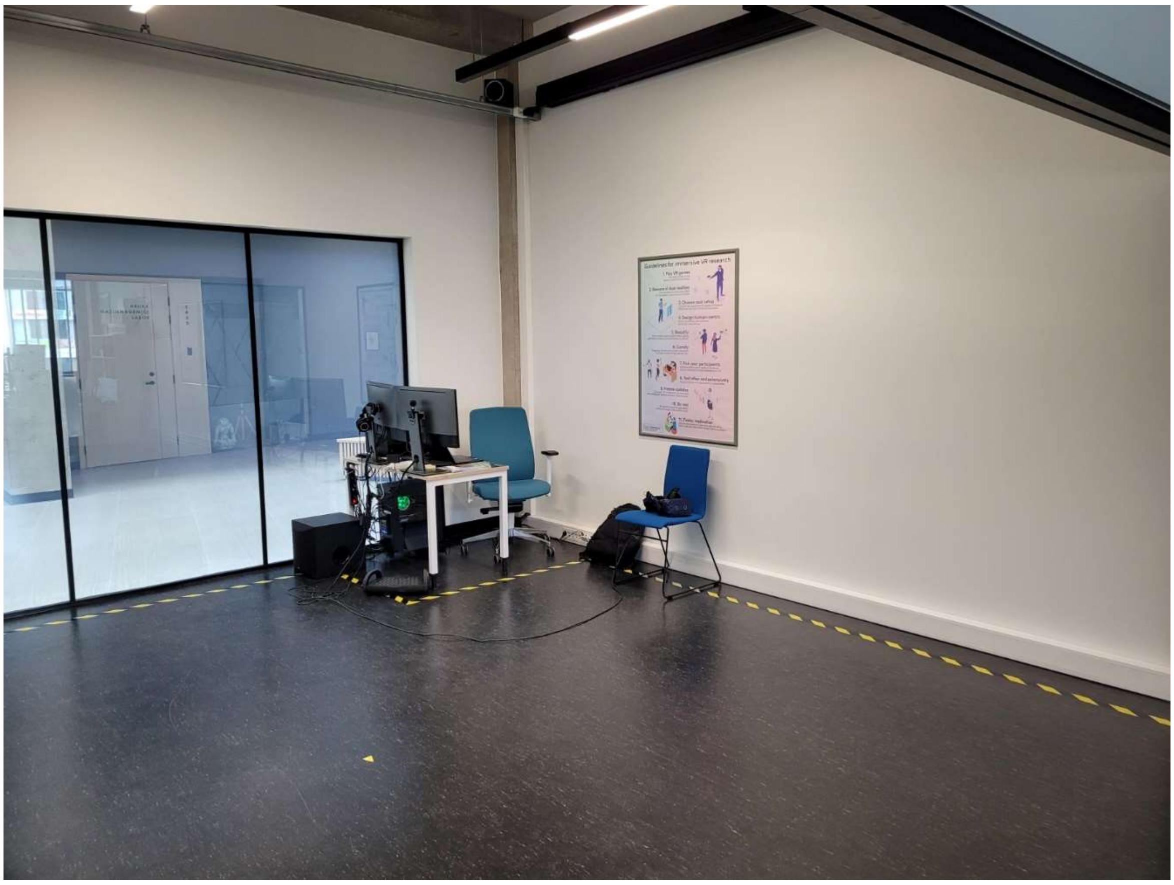


Figure 42. Picture of the testing area for one of the players.

The usability testing took place in room 2007 of the Delta Building (Figure 42). The participants were given a short introduction on what they were testing, were instructed how to put on the headsets, and were freely allowed to roam the virtual experience. They were, however, required to try out all the experiences. After trying out the experiences, they could optionally keep playing. After that, they were requested to fill out the questionnaire.

Subchapter 4.3.2.1 analyses the results of the sickness questionnaire. Subchapter 4.3.2.2 analyses the testers' opinions about the tutorial, the experiences, the Virtual Delta Building and the application in general.

4.3.2.1 Sickness Questionnaire Responses

The first part of the questionnaire was questions about sickness. Out of 20 participants, one reported that they had already felt bad before using the application. Their responses are not shown in any charts, as it is unclear if the application or headset caused harmful effects. Figure 43 shows how many participants reported feeling worse after using the application. Most participants did not think they felt worse after using the application.

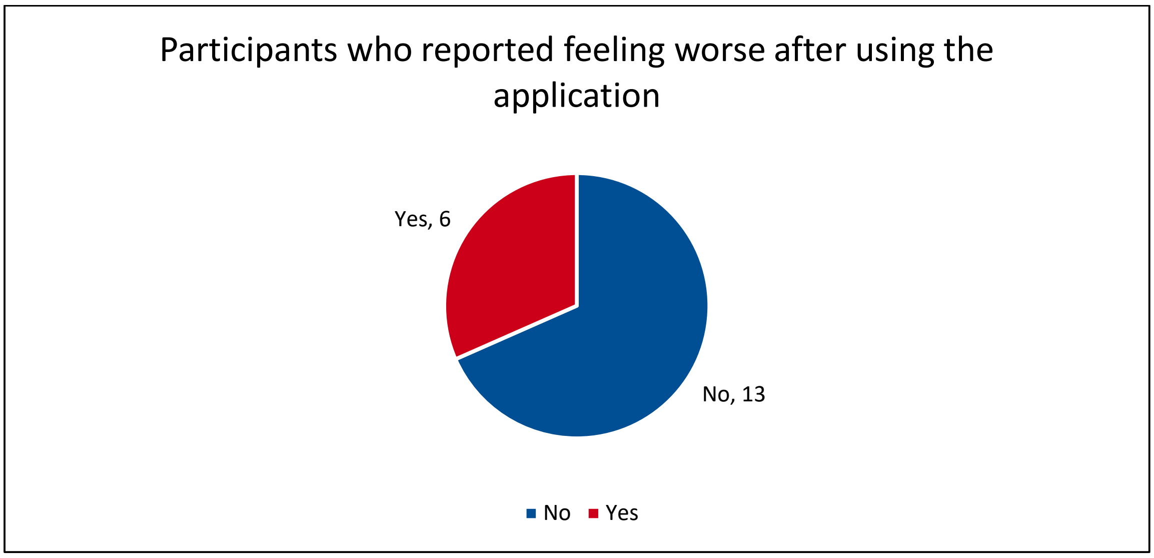


Figure 43. Participant answers to having side effects after playing the application.

Figure 44 shows participant answers related to the oculomotor section, which contains eye movement and vision issues. Most of the issues were only slight. Participants who reported feeling worse in this section said it was due to the headset not being properly adjusted for them. Most of them said issues started after taking the headset off, with one person saying it was due to the virtual scene being dark compared to the well-lit testing room.

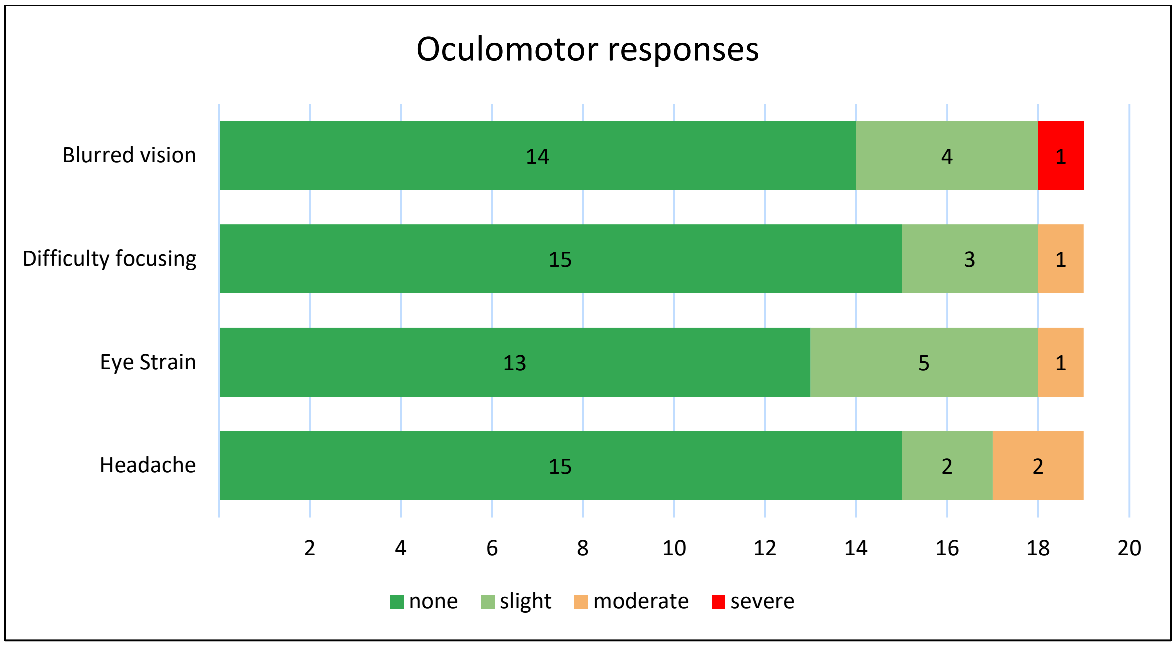


Figure 44. Eye movement and vision-related issues.

Figure 45 shows participant responses regarding nausea, which contains issues related to sickness and discomfort. Only a few participants reported issues in this category, with most of the issues only being slight. Participants commented that they had issues when using a lot of smooth locomotion, especially when walking on stairs or, in some rare cases, climbing classroom desks and chairs.

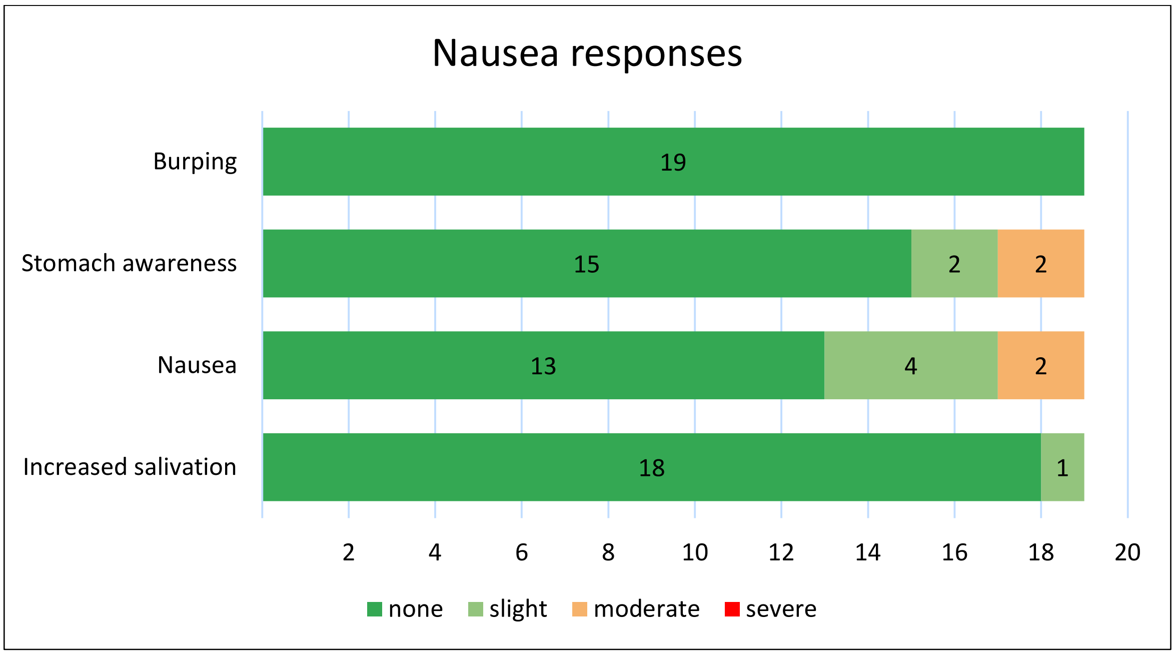


Figure 45. Sickness or discomfort-related issues.

Figure 46 shows participant responses regarding disorientation. Most participants only had slight issues in this category. Participants noted again that issues here were primarily due to smooth locomotion or fast movement in general.

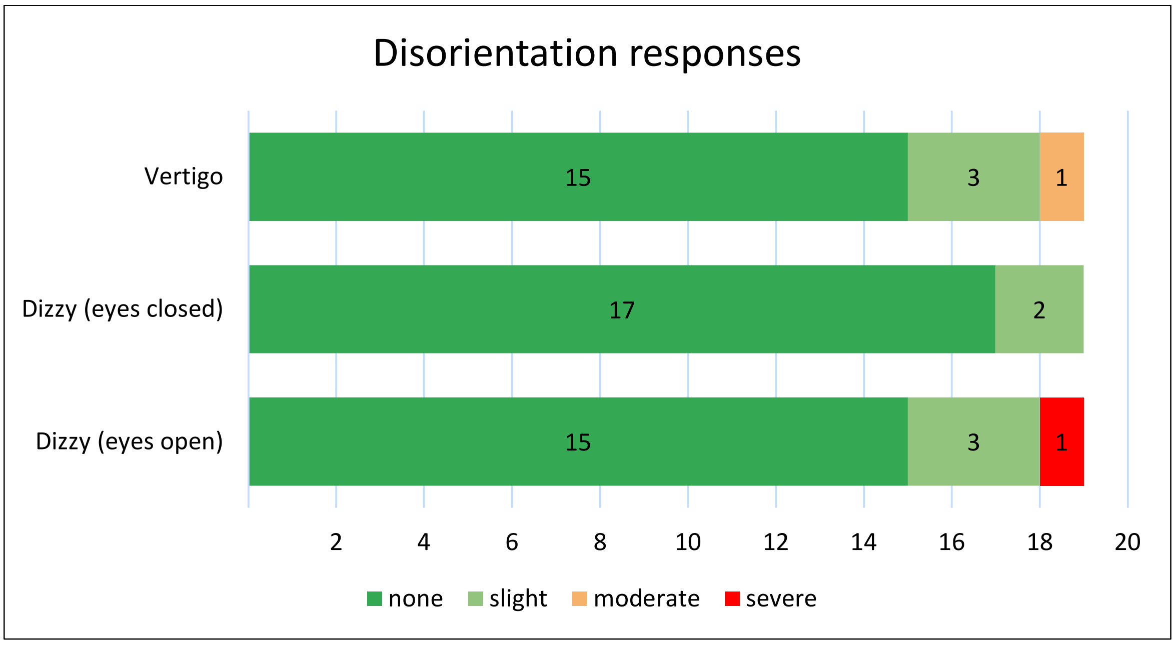


Figure 46. Disorientation or confusion-related issues.

Figure 47 shows uncategorized responses in the SSQ. In this section, more participants noted that they felt worse than before than in other sections; however, only slightly. Some participants noted that they started to feel tired over a more extended playing period. Additionally, some said the headset was on too tight, hurting their head, and the cable annoyed them. Some said that these symptoms were due to VR in general.

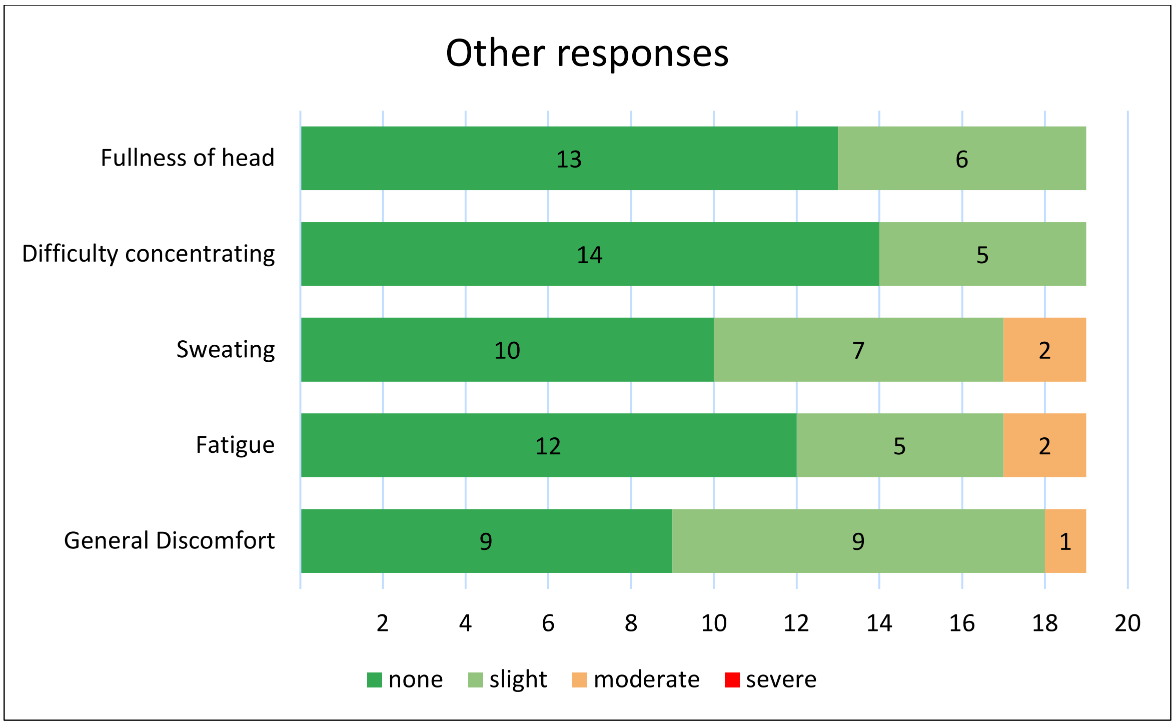


Figure 47. Uncategorized responses in the SSQ.

One of the reasons brought out for feeling worse was the use of smooth locomotion. While the application did offer an option to move by teleporting, many players chose not to use it. To reduce nausea caused by this, the vignette could be made stronger, or the option of smooth locomotion could be removed.

Another issue for feeling uncomfortable was due to one of the headsets used during testing, the Quest 2. Players often said using it was uncomfortable, as it had a bad fit. A bad fit can cause blurred vision and headaches. The lenses might not be appropriately aligned, and the headset presses against the head too strongly. One possible solution would be to use an alternative strap for the Quest 2 or better instruct the players on how the headset should fit.

4.3.2.2 Experience Related Responses

Figure 48 shows participant responses to questions about the tutorial. Most of the players had a positive opinion across all questions. One of the issues regarding the tutorial was still its noticeability. If players did not look down at their hands, they would not see the tutorial. This should be fixed by making the tutorial text always visible to the camera. However, once they noticed the tutorial, it was easy for most to follow its instructions.

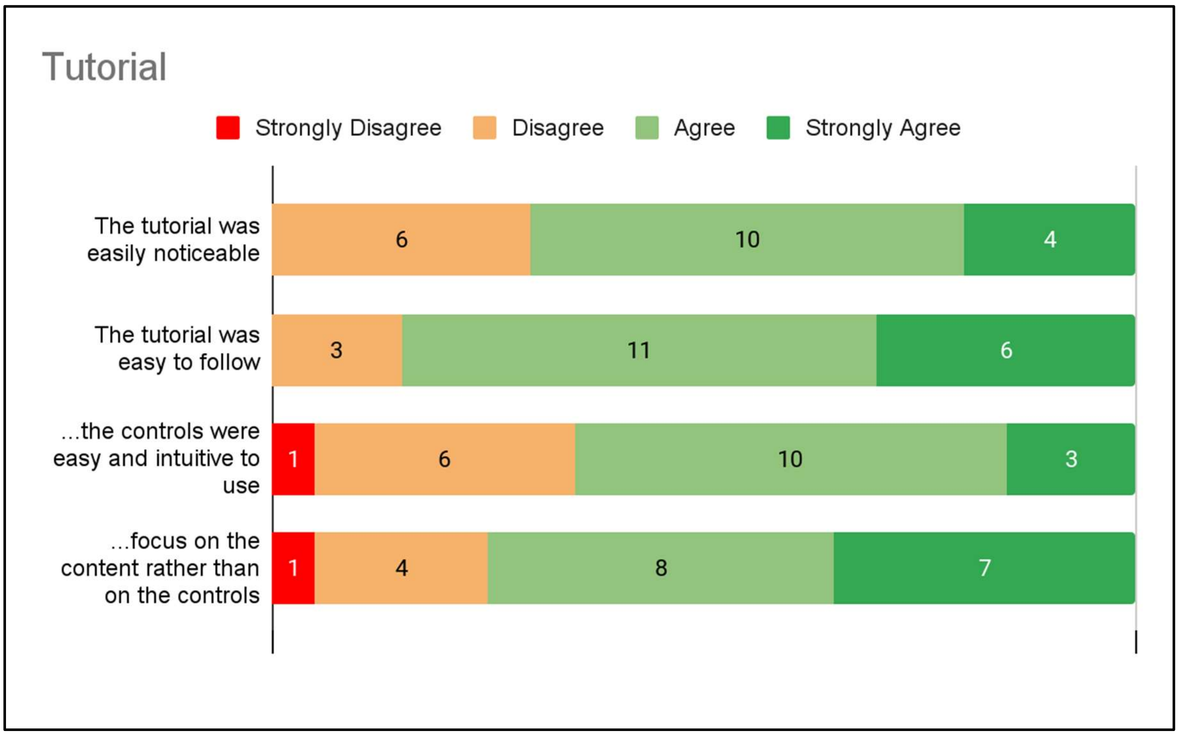


Figure 48. Playtester's opinion about the tutorial.

The control schema itself also brought out some issues. For locomotion, there are two options: smooth locomotion and teleportation. Players often choose to keep using one after completing the tutorial but forget how to use the other option. Since the application has some unused buttons, they could be made to show all the controls again as a reminder.

Also, players would often accidentally push the joystick to the sides when they wanted to move it forward, causing unwanted movements. To resolve this, the control schema could be simplified. Rather than having multiple controls on both joysticks, one joystick could teleport, and the other could only turn. This change would make the controls easier to remember, as there are fewer of them.

An issue remarked by some VR-experienced players was that they wished that the snap turning was instead smooth. However, as discussed in Chapter 2.3, such an option can create nausea for newer players, so it should stay a snap turn. What could be considered instead is giving experienced players the option to switch between these modes. However, as seen in the SSQ responses, players who have reported nausea often use the smooth locomotion option after being warned that it is more likely to create adverse effects, indicating that players overestimate their tolerance.

Figure 49 shows participant responses to the drawing experience. Playtesters' opinions about the experience were overall positive across all categories; however, some negative opinions were given.

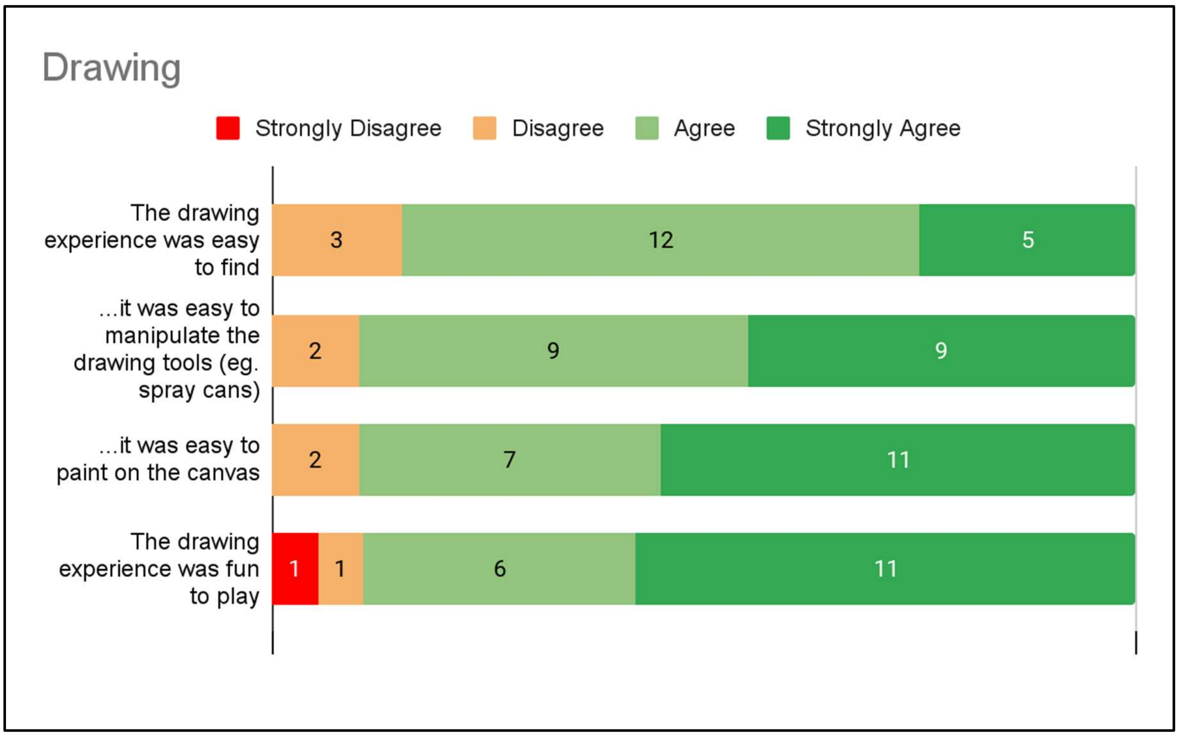


Figure 49. Playtesters' opinion about the drawing experience.

It was in the starting room, so players mostly did not have issues finding it. Players also had no issues with using the drawing tools. However, it was noted that it was difficult to draw on the blackboard/canvas.

The first issue was knowing they could only draw on the blackboard, as there was no indication. This could be fixed by adding a sign or having drawings on the b.

The second issue was that the spray cans only worked at a certain distance, and players often tried to use them outside their maximum range. This could be fixed by increasing the maximum range.

Overall, most players said they enjoyed the experience. The few that did not enjoy it said that drawing was boring. Enjoyment could be improved by adding more drawing tools and allowing players to mix colours.

Figure 50 shows participant responses to the spacewalk experience. While most of the responses were positive, some negative responses were given.

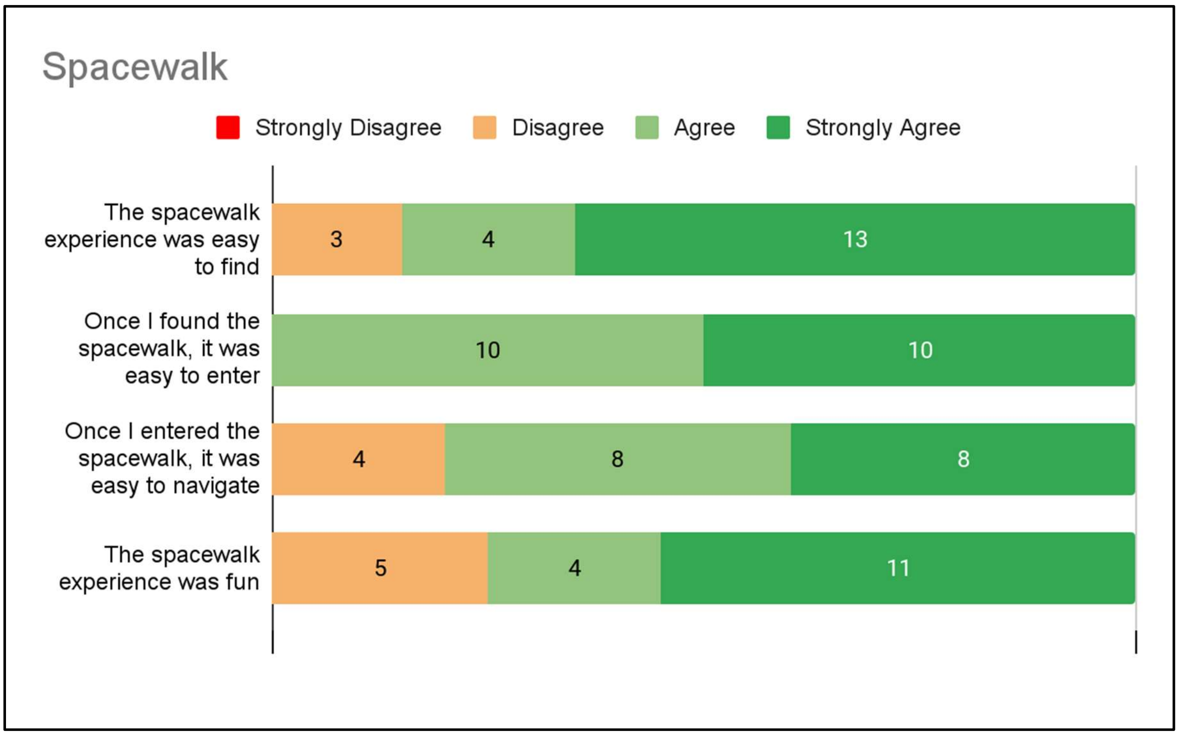


Figure 50. Playtesters' opinion about the spacewalk experience.

This experience was findable by just exiting the starting room, so players did not have difficulty finding it. Entering the experience was walking through a doorway, and players did not find this problematic. While the experience did require a mixed use of locomotion controls, most players did not struggle with this.

More VR-experienced players noted that it was not very interesting for them, as they had already tried alternatives. More content should be added to the spacewalk to improve this rating, as just walking across a plank was trivial for them. Additionally, some graphical glitches related to the Delta Building in the distance were brought out.

Figure 51 shows participant responses to the archery experience. Since the balcony was away from the starting area, many players could not find it without external assistance.

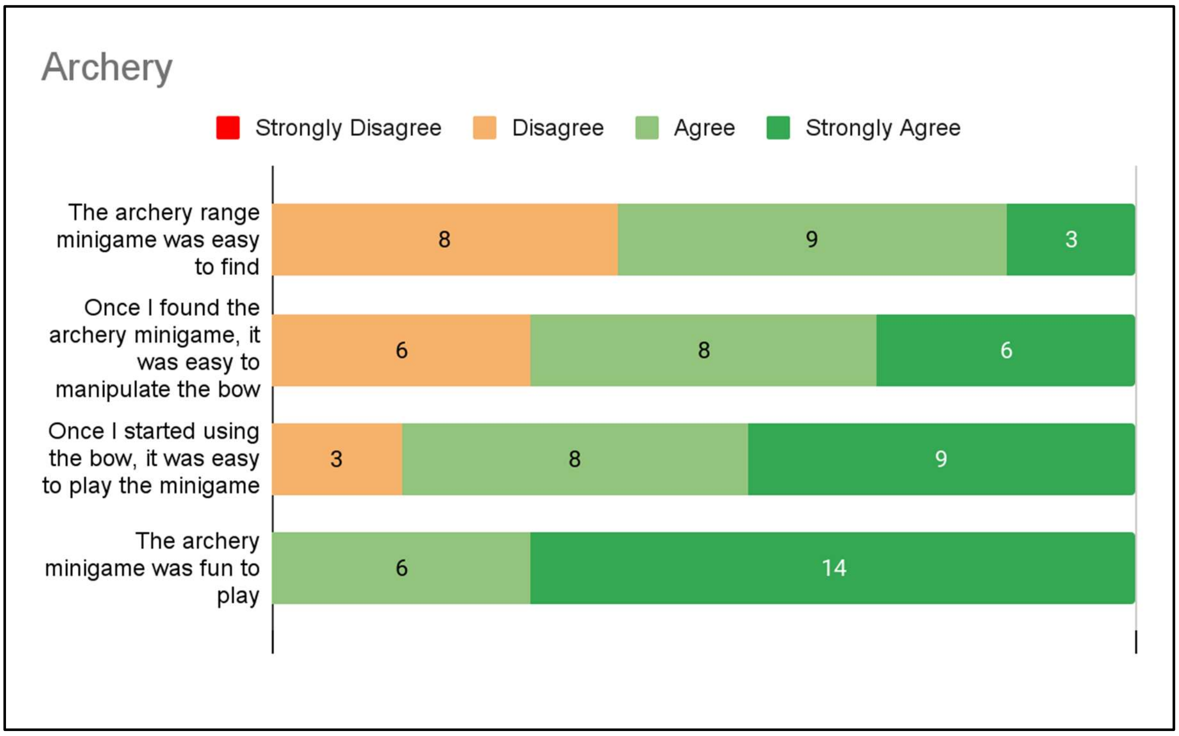


Figure 51. Playtesters' opinions on the archery range experience.

Some players still had issues manipulating the bow. They said that it was unintuitive to use. This was because it required interactions with both hands, but players were unaware they could use both hands. A separate tutorial could be implemented for the archery range to improve this.

Overall, the enjoyment of this experience was largely positive. This variant was playable by multiple people compared to the previous multiplayer thesis. It did not break down after one playthrough. However, people who played on the computer that did not host the server noted a slight delay in launching arrows. This delay made shooting the bow harder for them and should be fixed.

Some VR-experienced players noted that the experience gets boring quickly, as the enemy pattern is always the same. Some more variety to enemies could be added to make it more interesting for them.

Figure 52 shows participant responses to the virtual Delta Building. Most of the opinions were strongly positive; however, some issues were still brought out.

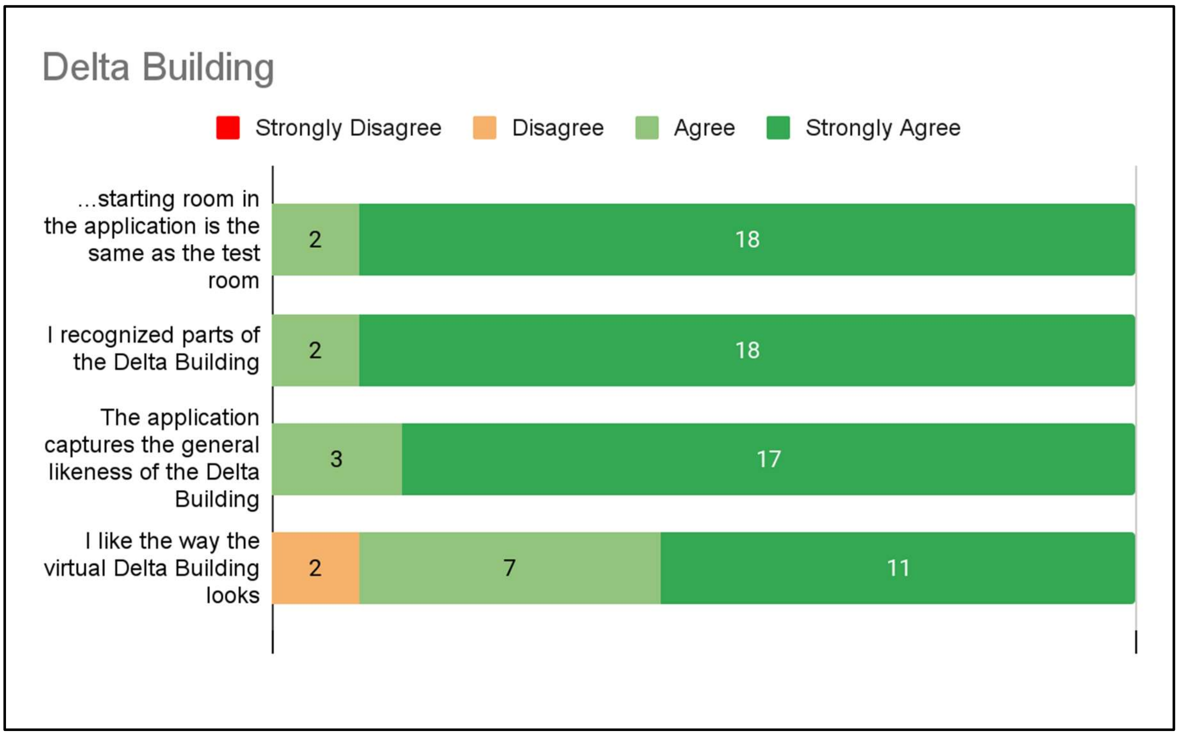


Figure 52. Playtesters' opinion on the virtual Delta Building.

All participants successfully recognized parts of the building, including the starting room, and overall think the application captures the likeness of the Delta Building well.

While most players liked how the building looked, they still noted some graphical issues. For example, some small places still have missing geometry; some stairwells have missing guardrails, and a tree was in the middle of the outside park walkway. Additionally, some graphical artefacts appeared from different angles when viewing the building outside.

Additionally, some players were disappointed that they could not go on floors besides the first and second floors. While the other floors do not have games, furniture or lighting, players could still be allowed onto those floors, for example, with flashlights. In that case, players should also be warned that nothing is there and they may get lost.

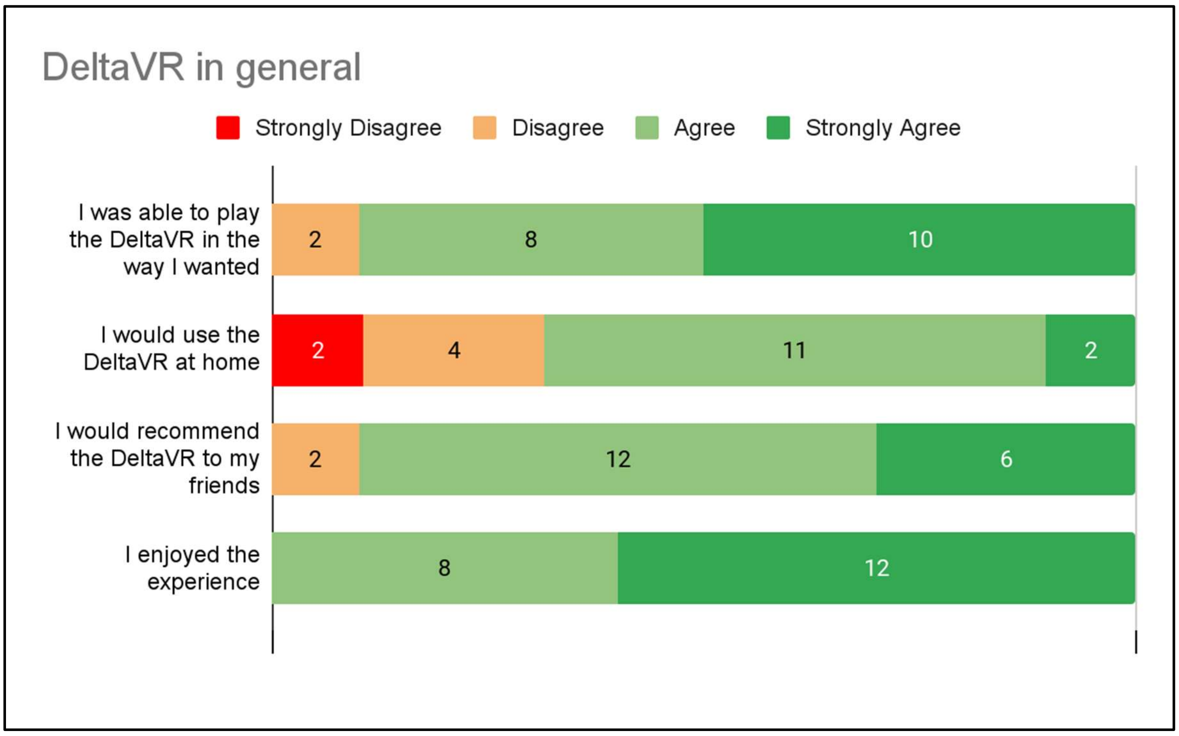


Figure 53. Playtesters' opinion on the application in general.

Chart X shows participant responses to some statements about *DeltaVR* in general. The first one was if they could play *DeltaVR* in the way they wanted. Participants mostly responded positively. Negative responses may be caused by participants being forced to play experiences they are not interested in or the application not meeting their expectations.

The second statement was if they would use the application at home. Participants mostly responded positively. The current iteration of *DeltaVR* is meant to be a short experience with little replayability; therefore, any negative ratings are understandable.

The third statement was whether they would recommend the application to their friends. Most players would recommend it to their friends, indicating that the application was enjoyable to them and that others may enjoy it too.

The same is shown by the last statement, where players say they enjoyed the experience. However, it was still brought out that there are not many things to do in the application, and some experiences should be more polished, such as adding more sounds, effects and feedback. Regarding multiplayer, some players felt that the minigames were not multiplayer-focused.

4.3.2.3 Issues Found

According to Nielsen, only five users are needed to find most issues in an application. To find all the usability issues, at least 15 users are needed. (Nielsen, 2000) Since the application was tested with 20 users, most issues should have been found. The issues found are listed in Table 3. The table consists of the ID of the issue, its description and if there was a matching issue in the predecessor *DeltaVR Multiplayer*.

Table 3. Issues found during Usability testing.

Issue ID	Description	Linked issue in predecessor
1	Arrows sometimes collide with invisible objects in the archery range	-
2	Drawing game spray looks choppy when moving hand too fast	-
3	Teleport ray is flipped sideways on some surfaces	-
4	Player hands do not release grip when walking too far away from doors, causing doors to teleport back and forward.	UI-8, FI-6
5	Players can walk through some doors	UI-8, FI-6
6	Players can not open some doors	UI-8, FI-6
7	You can fall off the map and get stuck falling	UI-7
8	You can physically walk through the red bounds in the spacewalk experience and then fall	UI-7
9	Index grip controls are too sensitive, causing players to hold onto them for too long	UI-8, FI-6
10	Non-host players have slight	-

	delay when shooting arrows	
11	Taking interactables from other people’s hands causes them to become desynced	UI-6
12	You can teleport up floors by aiming at the ceiling	UI-11
13	In some cases, you can get stuck in spacewalk after portals get destroyed	-
14	In some cases, newly connected players do not get the current version of the drawing canvas	-

While the application performed better than its predecessor, some issues were not completely fixed, or new ones were introduced with new experiences.

More significant usability issues remain with the doors. This issue is caused by the XR Interaction Toolkit changing the parent GameObject of the interactable the player is grabbing. This change was not synced over the network. A separate script to sync the parent would need to be developed to fix this.

New issues were created due to the addition of gravity. The map's borders should be walled off, or the player should be allowed to reset their position. Additionally, the player should be stopped from physically walking through some borders.

Networking code related to archery range needs client-side prediction to remove the slight delay when firing. The firing should be done on the client side before returning ownership to the server.

The Valve Index controller grip being too sensitive was a significant issue. The gripping value threshold should be adjusted, or players should be better instructed on how the controller works. The rest of the issues in the table are less critical but should be fixed.

Conclusion

During this thesis, a new version of *DeltaVR* was developed. The new version builds upon its predecessor, *DeltaVR Multiplayer*, by fixing game-breaking bugs, improving the virtual Delta Building, and adding and improving experiences. Additionally, the new version had both usability testing and performance testing.

Compared to the predecessor, usability testing showed that experiences were less likely to break down during gameplay due to various de-synchronization issues. Additionally, the new networking solution does not require an external internet connection, making it easier to show off during expos.

The first notable addition to the application was the inclusion of a tutorial. Usability testing shows that around ¾ of players should be able to learn the application's controls independently rather than requiring someone to explain them. However, some ideas to improve the application further were brought out for those who still needed help.

The second notable change to the application was improving the virtual Delta Building. Compared to the predecessors, this version of the building now includes all the building floors rather than being cut off from the second floor. Additionally, performance testing shows that even after adding much more detail to the experience, optimisations during the thesis made the application run faster than its predecessors. Usability testing showed that testers think the virtual building looks a lot like the actual building and most of the esters like how it looks.

The third notable addition and change were related to the experiences. The already included archery range experience was improved based on user feedback and observation, and usability testing results show that all the testers had fun playing it. However, some usability issues remained and were brought out during the testing chapter.

Two new experiences were added, a drawing experience and a virtual spacewalk experience. Most testers enjoyed both experiences; however, some usability issues and bugs remain here. The complete list of bugs remaining was brought out in Chapter 4.3.2.3.

Lastly, improvements were made to reduce the nausea caused by the application. During the usability testing, 6 out of 19 said they felt worse after playing the application. Most of these people rated their symptoms as slight on a scale of none to severe, and some of these issues were caused by a bad fit of the headset rather than the application alone. Since neither of the predecessors measured this, there was no baseline to compare this result.

In conclusion, this thesis has developed an enhanced version of *DeltaVR*, addressing numerous issues, expanding the virtual environment, and introducing new and improved user experiences. The usability and performance testing conducted throughout the development process has helped refine the application and provided valuable insights into areas requiring further improvement.

I want to thank Mark Muhhin for supervising this thesis. I am also grateful to Raimond-Hendrik Tunnel for conducting informative thesis writing seminars. Furthermore, I would like to extend my appreciation to Kaarel Rüüsak and Mathias Plans, fellow students from the University of Tartu, for their insightful feedback on the thesis. Lastly, a special thanks go to all the students from the University of Tartu Computer Science students mailing list who demonstrated interest in testing the application.

References

Bhandari, P. (2020, July 3). *What is a likert scale?* Scribbr. https://www.scribbr.com/methodology/likert-scale/

Glazer, J., & Madhav, S. (2015). *Multiplayer Game Programming: Architecting Networked Games*. Addison-Wesley Professional.

Jerald, J. (2015). *The VR Book: Human-Centered Design for Virtual Reality*. Morgan & Claypool.

Kao, D., Magana, A. J., & Mousas, C. (2021). Evaluating Tutorial-Based Instructions for Controllers in Virtual Reality Games. *Proceedings of the ACM on Human-Computer Interaction*, *5*(CHI PLAY), 1–28. https://doi.org/10.1145/3474661

Kennedy, R. S., Lane, N. E., Berbaum, K. S., & Lilienthal, M. G. (1993). Simulator Sickness Questionnaire: An Enhanced Method for Quantifying Simulator Sickness. *The International Journal of Aviation Psychology*, *3*(3), 203–220. https://doi.org/10.1207/s15327108ijap0303_3

Moran, K. (2019, December 1). *Usability testing 101*. Nielsen Norman Group. https://www.nngroup.com/articles/usability-testing-101/

Nielsen, J. (2000, March 18). *Why You Only Need to Test with 5 Users*. Nielsen Norman Group. https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/

Püks, J. (2022). *DeltaVR - Multiplayer* [University of Tartu]. https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=74390&language=en

Sellis, H. (2022). *Non-Euclidean Space in Virtual Reality*. https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=74350&year=2022&language=en

Tamm, T. (2021). *DeltaVR* [University of Tartu]. https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=71682&year=2021&language=en

Young, S. D., Adelstein, B. D., & Ellis, S. R. (2007). Demand Characteristics in Assessing Motion Sickness in a Virtual Environment: Or Does Taking a Motion Sickness Questionnaire Make You Sick? *IEEE Transactions on Visualization and Computer Graphics*, *13*(3), 422–428. https://doi.org/10.1109/tvcg.2007.1029

Zimmerman, E. (2003). Play as Research: The Iterative Design Process. In *Design Research: Methods and Perspectives*. The MIT Press.

Appendix

I. Dictionary

1. **CGVR Lab** - The Computer Graphics and Virtual Reality Study Lab in University of Tartu.[48]
2. **Collider** - Component for GameObjects in Unity. Allows two GameObjects to collide with each other.[49]
3. **De-sync** - situation where the game state is different between the clients and/or server.
4. **Depth buffer** - data structure in computer graphics used to determine the visibility and order of objects in a 3D scene based on their distance from the camera.[50]
5. **Deserialize** - converting serialised data back to its original format or structure.
6. **Disorientation** - loss of one's sense of direction or awareness of one's surroundings.
7. **FBX file** - file format for 3d models, animations, etc. Made for 3D applications.
8. **Frametime** - the amount of time it takes for a single frame to be rendered.
9. **IFC file** - file format used in architecture, engineering and construction industries to share data between different programs.
10. **Layer** (in Unity) - layer refers to a way of categorising game objects or components.
11. **Light probes** - light probes are objects used to sample and approximate the illumination of a 3D environment or scene by capturing and storing lighting information from different positions.
12. **Lightmapping** - technique used to precalculate and store the lighting information of a 3D scene into a texture or set of textures.
13. **Locomotion** - Locomotion refers to the ability to move oneself around.

[48] https://cgvr.cs.ut.ee/
[49] https://docs.unity3d.com/Manual/CollidersOverview.html
[50] https://en.wikipedia.org/wiki/Z-buffering

14. **Memory dump** - contents of a computer's memory written to a storage device.
15. **Mesh** - collection of information that defines the shape and geometry of a 3D object.
16. **Model** - see 15. Mesh
17. **Observer pattern** - object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, enabling the observers to update their state or behaviour accordingly.
18. **Occlusion** - blocking or obscuring of a portion of an object or scene by another object or surface.
19. **Occlusion culling** - selectively skipping the rendering of objects or portions of objects that are occluded or not visible to the camera.
20. **Oculomotor** - muscles and nerves involved in the movement and control of the eyes.
21. **Oculus Integration** - software development kit (SDK) provided by Oculus VR, designed to enable the development of virtual reality (VR) applications and games for the Oculus platform.
22. **Portal** - virtual doorway or passage between two different areas or environments within a game or virtual world
23. **Raycast** - simulating the path of a ray as it intersects with objects in a 3D environment.
24. **Render** - process of generating or creating a 2D image from a 3D model or scene.
25. **ScriptableObject** - objects that can store data.
26. **Serialize** - converting data or objects into a specific format that can be stored or transmitted.
27. **Shader** - computer program that is used to define the appearance and behaviour of objects or surfaces in a 3D scene.
28. **Smooth turning** - the player's view smoothly rotating or turning based on the movement of the player's head or controller.
29. **Snap turning** - the player's view instantly rotating or turning to a pre-defined orientation based on the movement of the player's head or controller.

30. **SteamVR** - virtual reality (VR) platform and software development kit (SDK) developed by Valve Corporation
31. **Stencil buffer** - graphics feature that stores per-pixel information, and it can be used to mask or selectively render parts of a scene.[51]
32. **Texture** - texture refers to an image or set of images that are applied to the surface of a 3D object or scene to simulate surface details.
33. **Texture2D** - class that represents a 2D texture or image.
34. **Universal Render Pipeline** (URP) - rendering system and set of graphics features provided by Unity game engine.
35. **Instantiate** - Instantiate refers to a method that creates a new instance of a prefab of game object at runtime.
36. **Modalities** - different forms or types of communication
37. **Nausea** - feeling of discomfort or queasiness in the stomach.
38. **Non-Euclidean geometry** - type of geometry that deviates from traditional rules, allowing for impossible geometries and spaces to be represented.
39. **Passthrough mode** - allows the user to temporarily view the real-world environment around them, typically using cameras mounted on the headset.
40. **Prefab** - a reusable template or blueprint for a game object, containing all the components, properties, and settings needed to create an instance of the object at runtime.
41. **Teleportation** - movement technique where the player's avatar or viewpoint is instantly transported from one location to another within the virtual environment.
42. **Vignette** - vignette refers to a gradual fade or darkening of the edges or corners of an image or video.
43. **Virtual reality** - computer-generated simulation or representation of a 3D environment or world, typically experienced through a head-mounted display or other specialised equipment.

[51] https://learn.microsoft.com/en-gb/windows/win32/direct3d9/stencil-buffer-techniques

II. Issues Discovered During Expos

Table 4. Performance issues.

ID	Name	Description
PI-1	Room join stutter	Game drops frames when joining a room, causing nausea for users
PI-2	Game stutter on laptop	Game started stuttering on a laptop, causes nausea
PI-3	Game doesn't run natively on Quest 2	Framerate issues on Quest 2

Table 5. Usability issues.

ID	Name	Description
UI-1	Can only customise with left controller	You cannot use the right controller to click buttons in the player customization screen
UI-2	No tutorial	Players currently need another player to explain controls
UI-3	No reset	When switching players, there is no option to reset the game
UI-4	Smooth turning only	Smooth turning causes nausea for some players
UI-5	Can teleport onto walls	Walls are teleportable surfaces, no check for surface angle

UI-6	Desync between players	Occasionally objects are out of sync between players
UI-7	No gravity	Players can float from the second floor when using smooth movement
UI-8	Doors hard to open	Players struggle to open doors
UI-9	Bow range hard to use	Players struggled to use the bow range, controls and movements are not understandable
UI-10	Player hands can clip through walls	
UI-11	Players can teleport through some surfaces	For example, robotics room glass on second floor
UI-12	Current multiplayer tied to central user account	Current multiplayer implementation requires authentication with a remote service, that is currently tied to the user who built the game.
UI-13	Running multiplayer in LAN mode requires proprietary software	Running the game in LAN only mode is not completely supported out of the box
UI-14	Player can not customize appearance after joining the room	Customizing appearance requires leaving and entering the game.
UI-15	Teleportation menu has no clicking cooldown	Teleportation menu on left hand keep clicking the button when your hand collides with it, repeatedly opening and closing the menu. Players could not open and close the teleportation menu properly.

UI-16	No indication of successful area teleport	Players didn’t notice that they had successfully teleported using the menu and tried to teleport to the same place again
UI-17	Players can teleport under the building	Players can exit the intended virtual space in several places

Table 6. Functional issues.

ID	Name	Description
FI-1	Bow range only one player	Only one player can play in the bow range at a time
FI-2	Bow range sometimes can’t pick up bows	Sometimes when picking up a bow, the bow teleports out of your hand and becomes unusable
FI-3	UFOs get stuck on the building	Occasionally UFOs in the bow range do not disappear when hitting the building and stay visible.
FI-4	Bow range arrows shoot through UFOs	Sometimes bow range arrows pass through the UFOs, and the player cannot score points.
FI-5	Bow range does not start	Bow range can’t be started in some cases
FI-6	Doors teleporting	Over longer play sessions doors start teleporting around the building
FI-7	Hand animations missing	
FI-8	Desync between players	In some situations, objects are not in sync between players

III. E-mail Sent to Testers

/English below/

Tere!

Otsin testijaid magistritööle DeltaVR Multiplayer 2.0. DeltaVR on virtuaalreaalsuse elamus Delta õppehoones, kus saab koos erinevaid minimänge mängida.
Testid toimuvad järgmise kolme nädala jooksul Delta õppehoone teise korruse arvutigraafika ja virtuaalreaalsuse õppelaboris.
Test võtab aega umbes 10-20 minutit ja lõpus tuleb täita tagasiside küsitlus.
Testimine on paarides, seega võib sõbra kaasa võtta või leiame ise kaastestija.

Kui on testimiseks sõber kaasa võtta, võite mõlemad kohe aja kirja panna siin:
https://calendly.com/toomastamm/deltavr-multiplayer-2-0-testing

Kui on soov testida, aga sõpra ei leidu, saab siin endale sobivad ajad valida ja leiame kaastestija:
https://doodle.com/meeting/participate/id/el2pyMgb

Pärast kummagi vormi täitmist saadan teile e-kirja.

Küsimuste korral võib kirjutada e-mailile:
toomas@toomastamm.ee

Parimate soovidega
Toomas Tamm
--
Hi

I am seeking testers for my master's thesis, DeltaVR Multiplayer 2.0. DeltaVR is a virtual reality experience where you can play various minigames together.
Tests are happening within the next three weeks on the second floor of the Delta Building in the Computer Graphics and Virtual Reality Lab.
The test lasts 10-20 minutes and ends with a feedback form.
Tests are done in pairs, so either bring a friend or we can find someone you can test with.

If you already have a friend to bring with you, both can book the same time here:
https://calendly.com/toomastamm/deltavr-multiplayer-2-0-testing

If you want to test but don't have a friend to bring with you, fill in your available times here, and we will find someone:
https://doodle.com/meeting/participate/id/el2pyMgb

After filling either of the forms, I will follow up with an e-mail.

In case of questions, send me an e-mail:
toomas@toomastamm.ee

Best regards
Toomas Tamm

IV. Feedback Questionnaire

Questionnaire Part 1 - Sickness

1. Did you feel that you are in the same state of good health as you started the experiment?

   ☐ Yes ☐ No

   If you answered no, please explain briefly in the space provided below.

For each of the following conditions, please check how you are feeling right now, on a scale of none to severe:

Condition	none	slight	moderate	severe
1. General discomfort				
2. Fatigue				
3. Headache				
4. Eye strain				
5. Nausea (stomach distress)				
6. Blurred vision				
7. Dizzy				
8. Vertigo (surroundings seem to swirl)				

If you expressed slight, moderate, or severe on any of the questions above, please state if you felt that way before using the application. And if so, explain how you felt worse after using the application. If possible, specify when it started/changed during the application use.

Questionnaire Part 2 - Likert scale

1. Mark a single box (strongly disagree, disagree, agree, or strongly agree, not relevant) for each statement below.

2.1 Tutorial

	Strongly disagree	Disagree	Agree	Strongly agree	Not relevant
The tutorial was easily noticeable					
The tutorial was easy to follow					
Once I followed the tutorial, the controls were easy and intuitive to use					
Once I finished the tutorial, I found that I could focus on the content rather than on the controls					

2.2 Spacewalk

The spacewalk experience was easy to find					
Once I found the spacewalk, it was easy to enter					
Once I entered the spacewalk, it was easy to navigate					
The spacewalk experience was fun					

2.3 Archery

	Strongly disagree	Disagree	Agree	Strongly agree	Not relevant
The archery range minigame was easy to find					
Once I found the archery minigame, it was easy to manipulate the bow					
Once I started using the bow, it was easy to play the minigame					
The archery minigame was fun to play					

2.4 Drawing

The drawing experience was easy to find					
Once I found the drawing experience, it was easy to manipulate the drawing tools (eg. spray cans)					
Once I started manipulating the drawing tools, it was easy to paint on the canvas					
The drawing experience was fun to play					

2.5 Application in general

I was able to play the application in the way I wanted					
I would use the application at home					

I would recommend the application to my friends					
I enjoyed the experience					

2.6 Delta Building

Have you been to the Delta building before this test session: ☐ Yes ☐ No

	Strongly disagree	Disagree	Agree	Strongly agree	Not relevant
I recognized that the starting room in the application is the same as the test room					
I recognized parts of the Delta Building					
The application captures the general likeness of the Delta Building					
I like the way the virtual Delta Building looks					

Questionnaire Part 3 - Background/Experience

Age ____

Gender ____

For each question, put a check next to your answer.

1. How often do you use a computer in your average week?
   Before today, I used computers less than:

   ☐ 1 hour ☐ 2 hours ☐ 5 hours ☐ 10 hours ☐ 20 hours ☐ 40 hours

   ☐ more than 40 hours

2. In the past 2 years, what is the most you have played video games in a single week? (both virtual reality and regular)?

   Before today, I played less than:

   ☐ 1 hour ☐ 2 hours ☐ 5 hours ☐ 10 hours ☐ 20 hours ☐ 40 hours

   ☐ more than 40 hours

3. Have you used virtual reality applications before this application (applications do not have to be games)?

   ☐ Yes ☐ No

4. In the past 2 years, what is the most you have used virtual reality applications in a single week (applications do not have to be games)?

   Before today, I used applications for less than:

   ☐ 1 hour ☐ 2 hours ☐ 5 hours ☐ 10 hours ☐ 20 hours ☐ 40 hours

   ☐ more than 40 hours

Questionnaire Part 4 - Open-ended questions

1. What did you most like about the application?

2. What did you think about the experiences/minigames?

3. What did you think about the virtual Delta Building?

4. What did you dislike about the application?

5. What suggestions or ideas do you have for improving the application?

6. Any other comments?

V. User Guide

The build of the application can be found in the attached materials, in the following folder: *DeltaVRMultiplayer2\DeltaVR Build.* Double click DeltaVR.exe to start. After launching, the game window should open (Figure 54).

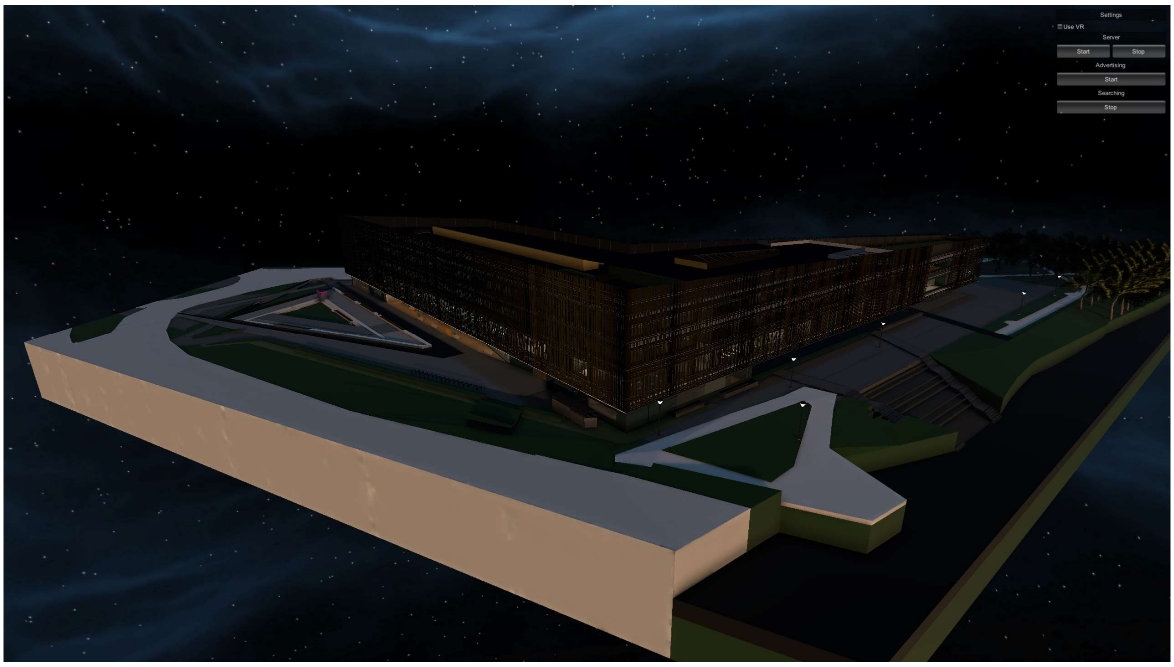


Figure 54. Game window after opening the game.

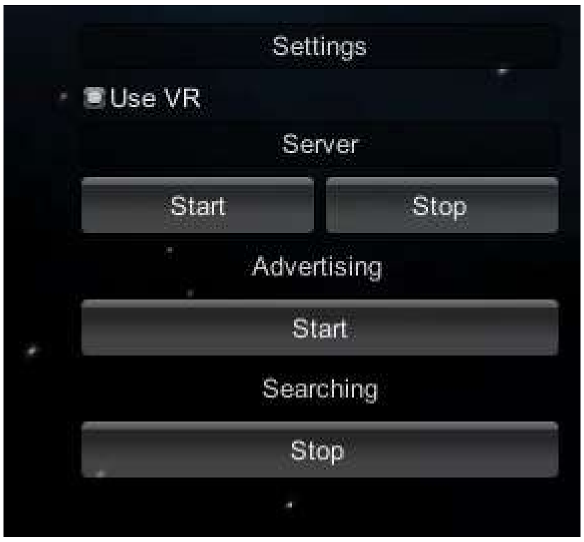


Figure 55. Zoomed-in view of networking controls and player settings on the top right of the game window.

In the top right corner of the game window, 4 section should be visible (Figure 55). The first section is settings. If "Use VR" is checked, the game will launch in VR mode when joining the server. If it is not checked, the game will launch in desktop mode. It should be noted that experiences are not playable in desktop mode.

The second section is server. This category has two buttons, one to “Start” the listen server, and another to “Stop” the listen server. When playing alone or this should be the computer that hosts the server, “Start” should be clicked here. If a firewall notification was received and allowed, it is recommended to restart the application for the settings to take effect.

The third section “Advertising”. This option is only relevant if this computer is hosting the server. If Advertising is started, the server will broadcast out to the local network that a server is running on the computer and computers will be able to find and connect to it.

The fourth section is “Searching”. In this category you can toggle if servers should be searched. Regardless if the server is being hosted on this computer or another, to get in the game, you need to connect to a server found here. If the server is not visible, for troubleshooting stopping and starting searching may find it.

After clicking on an IP below the “Searching” section, the game should start. In desktop mode, by moving the mouse the camera pans around, and by holding WASD buttons the character moves around. In VR mode, the controllers should show a tutorial, however for convience sake, the controls are included below.

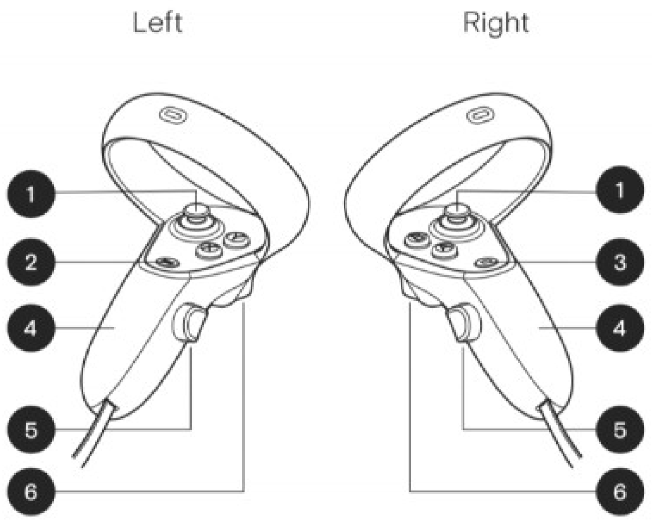


Figure 56. Quest 2 controls[52].

[52] Image from: https://www.vrteamspace.co.uk/setup/

In Figure 56, controls on the Quest 2 are numbered. The controls are as follows:

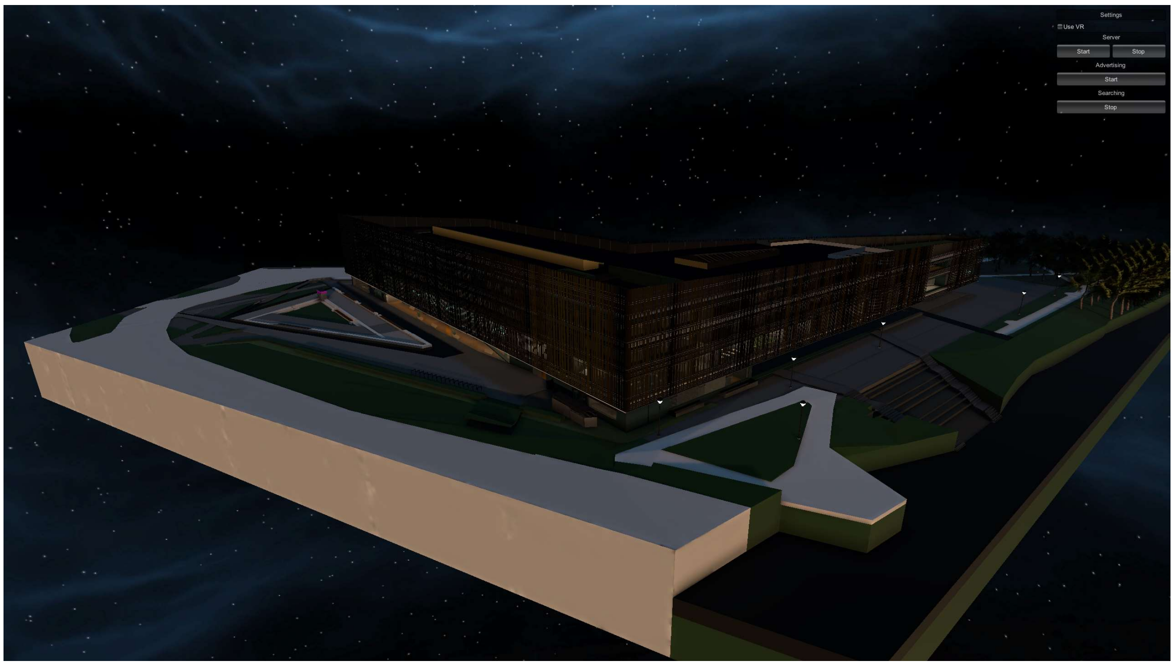
1) On the left controller, by moving joystick represented with number 1, the character moves around using smooth locomotion
2) On the left and right controller, the buttons represented by number 6 let you spray with the spray gun in the drawing experience.
3) On the left and right controller, the buttons represented by number 4 let you grip items in the game by holding them down. The virtual controller or hand should be inside the object to grip it.
4) On the right controller, by moving and holding the joystick represented by number 1 forward, you can aim the teleport ray. When the ray is white and ends with a blue circle, you can teleport there by centering the joystick.
5) On the right controller, by moving the joystick represented by number 1 left or right, you can turn your character left or right using snap turning. This is to avoid getting tangled on the cable.

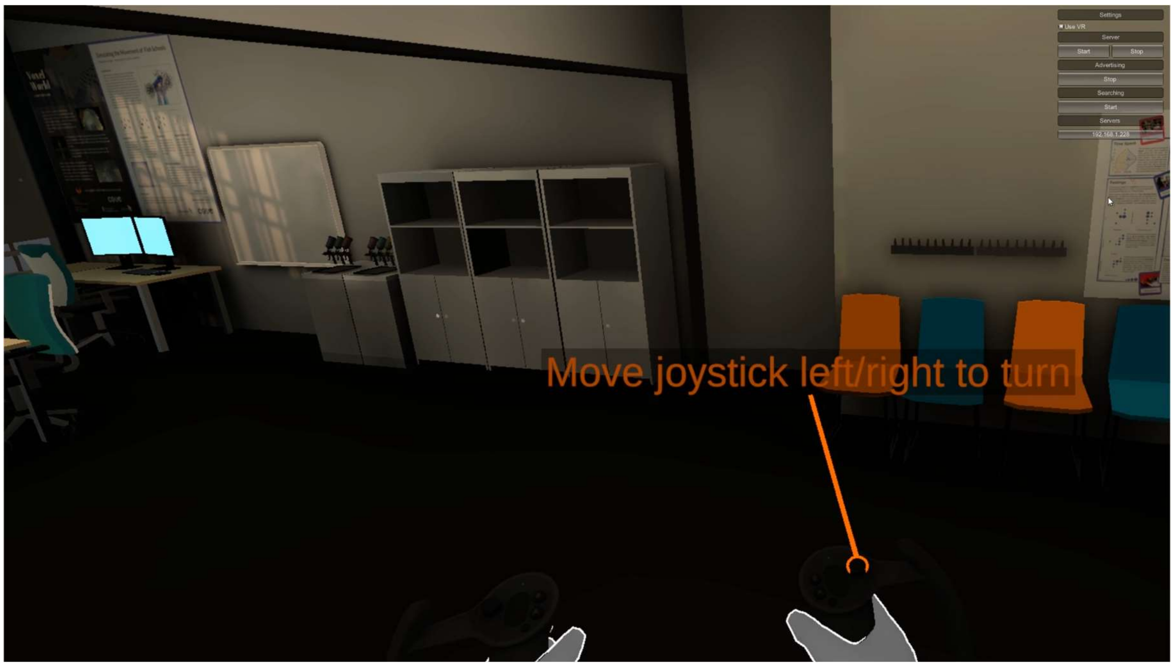
The controller scheme works similar on other controllers as well. Tested controllers are the Valve Index controllers, the Quest Touch controllers and the Vive Wands. If your controller is not on the list, it should still be supported by the application, but the tutorial will default to the Quest Touch controllers.

When moving long distances in the application, it is strongly recommended to use teleportation over smooth locomotion, as it causes less nausea to the user.

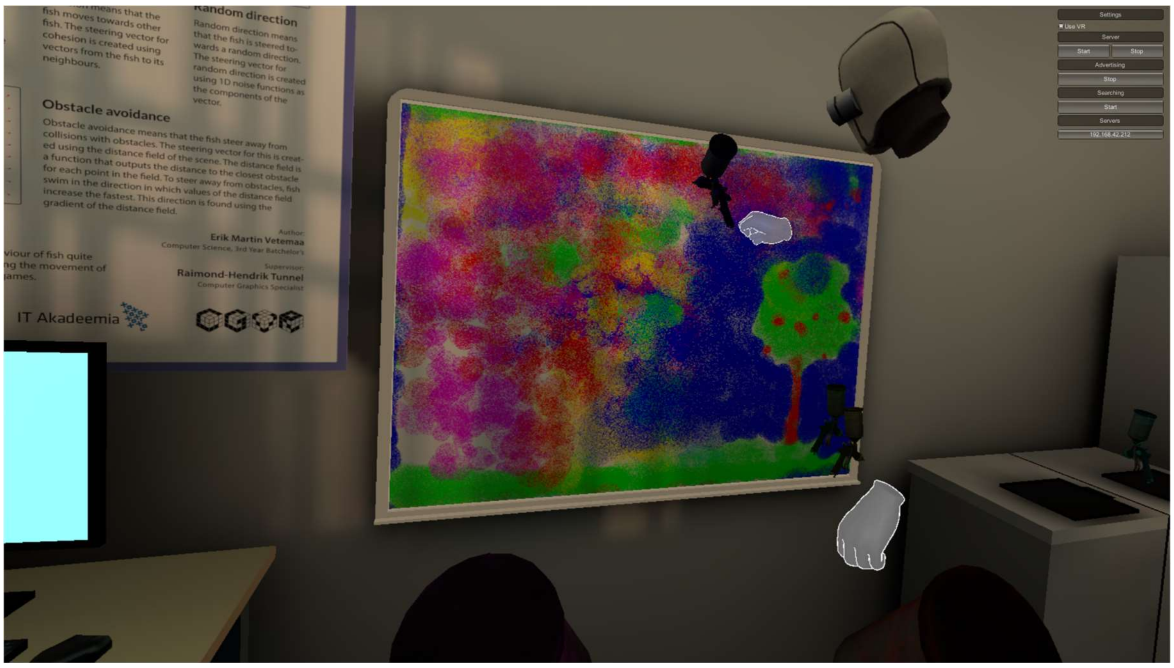
In case there are any issues during gameplay, a video is also available in the following folder: *DeltaVRMultiplayer2\Videos*.

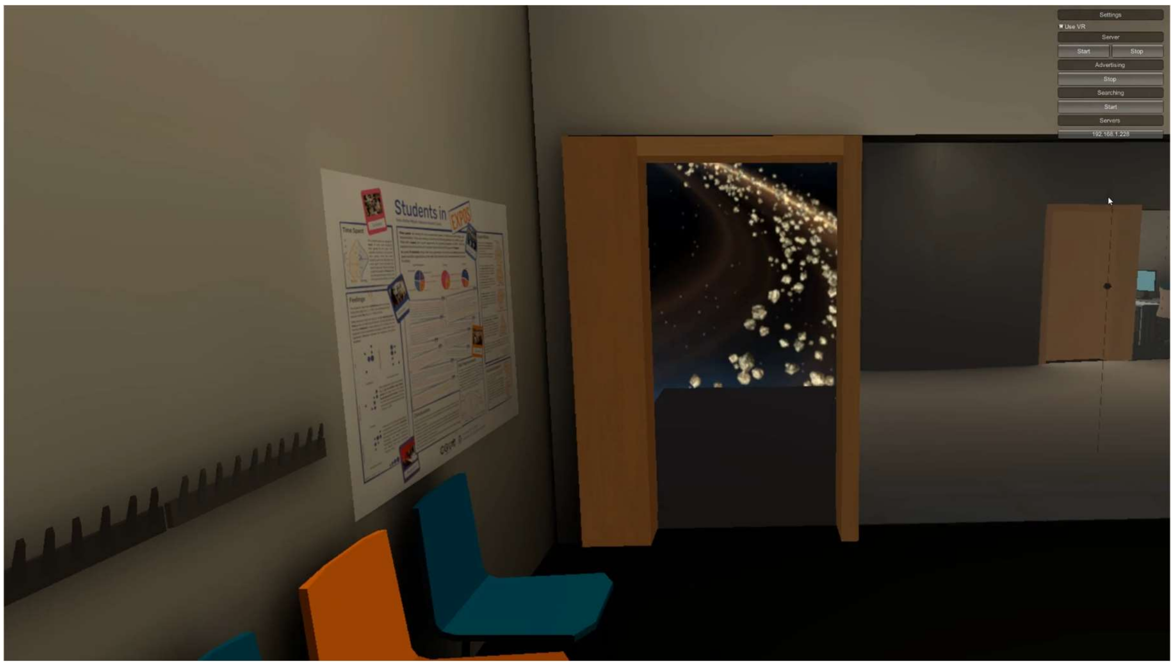
VI. Pictures of Application

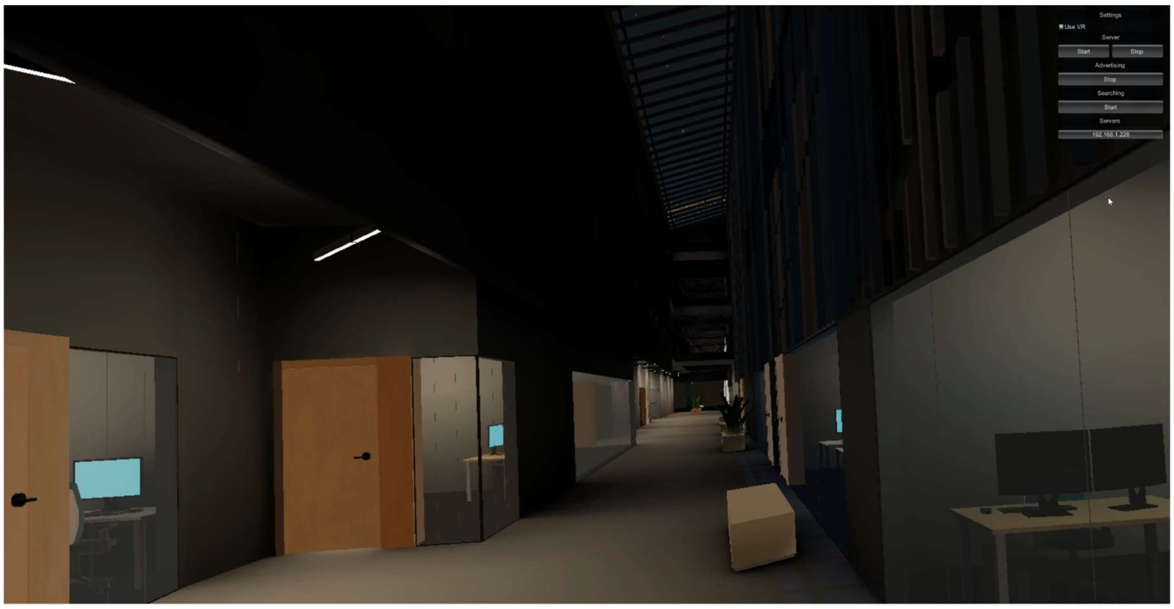


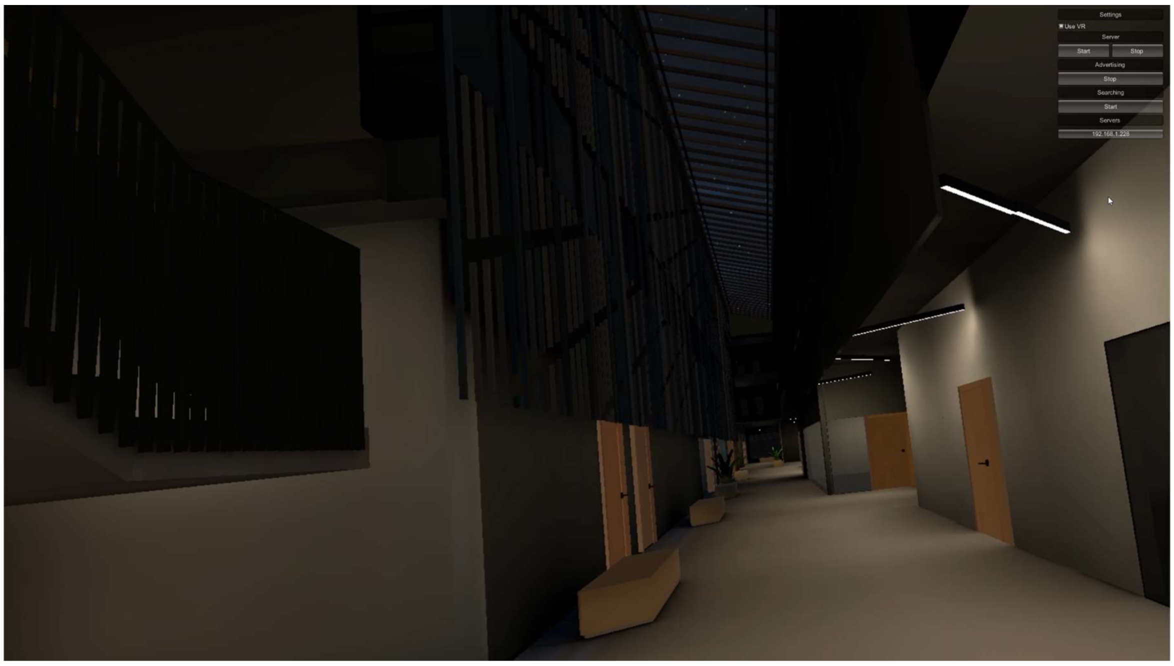


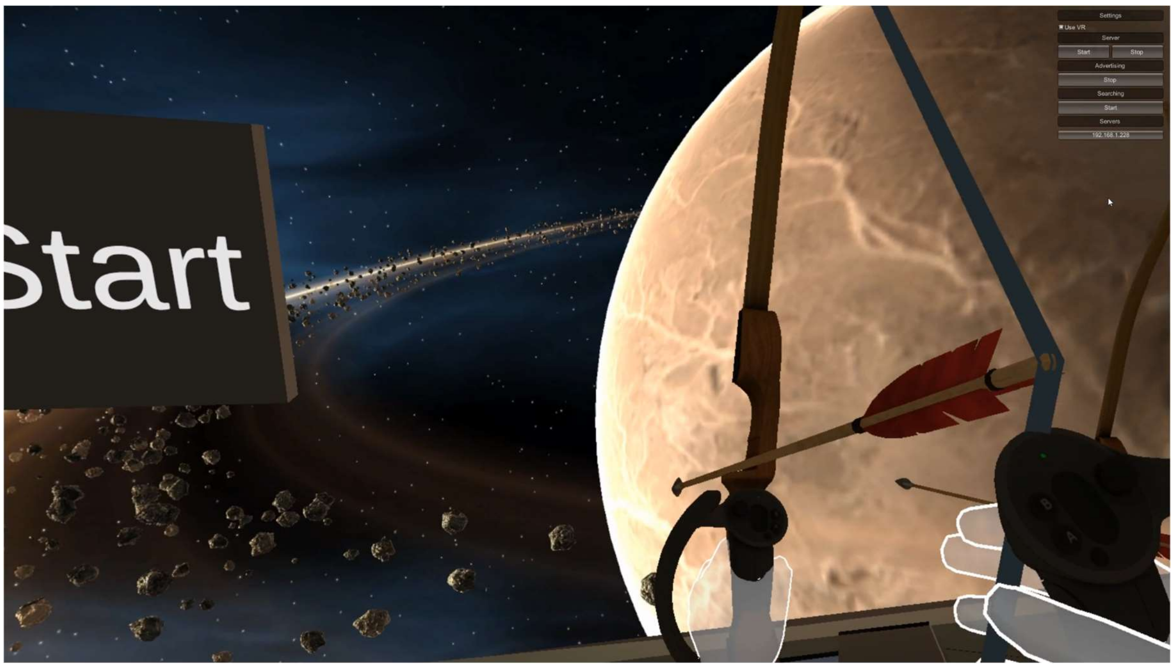
Move joystick left/right to turn

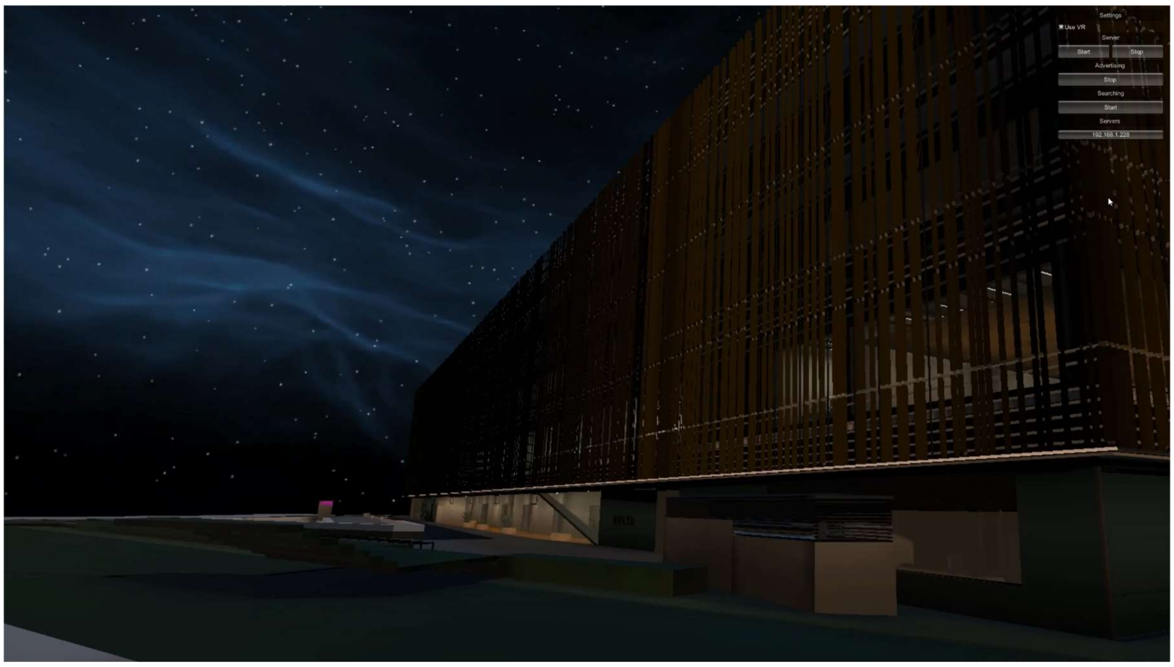

means that the fish moves towards other fish. The steering vector for cohesion is created using vectors from the fish to its neighbours.
Random direction
Random direction means that the fish is steered towards a random direction. The steering vector for random direction is created using 1D noise functions as the components of the vector.
Obstacle avoidance
Obstacle avoidance means that the fish steer away from collisions with obstacles. The steering vector for this is created using the distance field of the scene. The distance field is a function that outputs the distance to the closest obstacle for each point in the field. To steer away from obstacles, fish swim in the direction in which values of the distance field increase the fastest. This direction is found using the gradient of the distance field.
Author:
Erik Martin Vetemaa
Computer Science, 3rd Year Batchelor's
Supervisor:
Raimond-Hendrik Tunnel
Computer Graphics Specialist
viour of fish quite
ng the movement of
james.
IT Akadeemia
Settings
Use VR
Server
Start
Stop
Advertising
Stop
Searching
Start
Servers
192.168.42.212


Students in EXPO
Settings
Use VR
Server
Start
Stop
Advertising
Stop
Searching
Start
Servers
192.168.1.220


Settings
Use VR
Server
Start
Stop
Advertising
Stop
Searching
Start
Servers
192.168.1.228


Settings
Use VR
Server
Start
Stop
Advertising
Stop
Searching
Start
Servers
192.168.1.228


Start
Settings
Use VR
Server
Start
Stop
Advertising
Stop
Searching
Start
Servers
192.168.1.228


Settings
Use VR
Server
Start
Stop
Advertising
Stop
Searching
Start
Servers
192.168.1.228

VII. License