UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science

Joonas Püks

DeltaVR - Multiplayer

Bachelor's thesis (9 EAP)

Supervisor: Ulrich Norbisrath, PhD

Tartu 2022

DeltaVR - mitmikmäng

Lühikokkuvõte:

Lõputöö kirjeldab mitme mängija versiooni loomise protsessi virtuaalreaalsuse programmile DeltaVR. DeltaVR on programm, kus mängija saab uurida Delta keskust virtuaalreaalsuses ja mängida mänge. Töö käsitleb erinevaid viise, kuidas olemasolevat tarkvara Unity mängumootoris edasi arendada mitme mängija jaoks ja sellega kaasnevaid raskusi. Nende hulka kuuluvad erinevad Unity paketid, mitme mängija jaoks disainimine ja platvormide vaheline koostöö.

Võtmesõnad:

Virtuaalreaalsus, Unity, Mitmikmäng

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

DeltaVR - Multiplayer

Abstract:

This thesis will outline the development of the multiplayer extension to DeltaVR, a virtual reality application that allows the player to explore the Delta Center building of the University of Tartu and play minigames that are located in this virtual building. The text will go over different considerations and challenges of adding multiplayer capabilities to an existing single player virtual reality application. These also cover multiplayer networking solutions, multiplayer design and cross-platform compatibility.

Keywords:

Virtual reality, Unity, Multiplayer

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

Table of Contents

1. Introduction

Virtual reality has become more popular in the last five years. Additionally, with Meta and other companies talking about building the metaverse now, virtual reality and multiplayer are more relevant than ever. The topic of this thesis is adding a multiplayer extension to an existing virtual reality software called DeltaVR. This project was proposed by the CGVR lab and I chose it because of my interest in virtual reality. Multiplayer in virtual reality is somewhat new and such a type of thesis has not been done yet in the University of Tartu's Computer Graphics and Virtual Reality Lab. This project will likely have many challenges not only because developing multiplayer applications is difficult but since before actually starting development, I will need to check the current solution's code and understand it well enough to modify it efficiently. When successful, DeltaVR can be played on different platforms with multiple people. The work done will lead to a social application made specifically for an educational setting.

2. Previous Work

University of Tartu's CGVR lab members have made many different projects regarding virtual reality and its uses. A virtual reality headset study conducted by Peeter Paal in 2021 as a bachelor's thesis can support my work on the testing side [1].

2.1 DeltaVR

DeltaVR is a virtual reality application developed by Toomas Tamm as a bachelor's thesis in 2021 [2]. The model of the building was taken from the Delta Building Visualization project. In DeltaVR the user can explore the Delta Center by using teleport locomotion and interacting with tracked controllers. There are also two minigames in DeltaVR. One of them is an archery range where you use a bow and arrow to shoot at UFO-s. The second minigame is Breakout, where the player uses paddles to break boxes advancing towards them. Adding a multiplayer component to DeltaVR would allow multiple people to explore the Delta Center and potentially play the minigames together.

The result of DeltaVR left a very good groundwork for my thesis. Since the environment is already created, I can focus solely on the multiplayer extension itself. Multiplayer support is also a fitting addition for an application like this because of the existing environment.

Aside from adding multiplayer, I also plan to improve the application in other ways, like supporting more controllers and platforms. These changes, additions and improvements will be catalogued at the end of chapter 3.

2.2 Orbital Battleship

A similar solution to my intended solution is Orbital Battleship, an existing chemistry themed pen and paper game published in 2016 made into a virtual reality game in 2020 [3]. The paper goes through the process of creating the environment for virtual reality and evaluating the effect on student learning compared to the pen and paper version. While the sample size was very large (18 people), the results seem to have a positive effect on student learning.

While Orbital Battleship has similarities with DeltaVR broadly, there still are a few key differences. Firstly, Orbital Battleship was made for the Oculus Go headset. This means that the application was made for three degrees of freedom and the player cannot move laterally on their own. Orbital Battleship was also made in Unreal Engine. The reasons are convincing and I have been told about the ease of multiplayer implementation in Unreal Engine but since the original DeltaVR is made in Unity, porting the entire application to Unreal Engine would likely be enough work for another bachelor's thesis.

Even considering these differences, this paper still gives insight on developing a multiplayer application in virtual reality. Since DeltaVR comes from an educational background and has potential for learning use, Orbital Battleship is a good example of a multiplayer virtual reality application.

2.3 Unity Multiplayer Solutions

There are different ways to introduce multiplayer networking into a Unity project. I am going to evaluate three of them here and will go over the main advantages and disadvantages for each of them. I will choose one of them to go forward with based on these features. Links to all of the solutions considered are in the appendix.

Unity Multiplayer Networking

Unity has its own official networking solution introduced in 2020. This was the first solution I considered for DeltaVR because of its first party nature but the main disadvantage is how new this solution was. This means that the networking solutions was in active development and would not be considered very stable. Also, there were not many resources or tutorials for using it and troubleshooting common issues.

Mirror

An example of a third-party networking toolkit is Mirror that was released in 2019. Its main advantages are that it is open-source and was made by an experienced team that made a MMORPG toolkit for Unity. However, it is relatively new and needs a dedicated server to host.

Photon Unity Networking

One of the most popular and longstanding solutions for Unity networking is Photon Unity networking by Photon. Its initial version dating back to 2011, it has been a standard for integrating multiplayer into a Unity project before Unity's own networking solution. This also means that there are a lot of tutorials, documentation and discussions about using this toolkit for multiplayer development. In addition, Photon offers free cloud servers that can host up to 20 players at a time and official addons for voice chat. But as a downside, Photon Unity Networking does not allow dedicated servers without a fee, unlike Mirror and Unity Multiplayer Networking. Considering these advantages, I have decided to use Photon Unity Networking because of its approachability, many features and wide acceptance. I will be using Photon Unity Networking version 2, released in 2018.

3. Design

In this chapter I will analyze some virtual reality design points and explain my design decisions in making a multiplayer version of DeltaVR. The concepts that are outlined here will guide the multiplayer implementation.

3.1 Room-scale vs Remote

A fundamental decision that has to be made before any practical work for DeltaVR. That is if the users of DeltaVR should be in the same space when playing or should they play remotely. Born conducted a study to find if co-located users have more or less difficulty communicating and performing tasks [4]. This was evaluated by letting the participants play a virtual reality game that requires teamwork and communication and then surveying the participants. They found out that remote interactions work better for teamwork and communication. I have decided to design the multiplayer around users interacting with each other remotely. Therefore, this is likely the right call as the Delta Building space is too large to be explored in one room. When users can join the session remotely, they could move in the building independently of one another.

3.2 User Avatars

When multiple users are in the same space, there needs to be a visual indicator of each user. Existing social applications in virtual reality use different approaches to solve this. Usually, virtual reality systems have three tracking points: the headset and two controllers. This means that only the user's head and hands can be accurately tracked. This is why a lot of virtual reality social applications that use user avatars have avatars without arms or legs. To track arms and legs there needs to be an inverse kinematics system in place and even with that the results can look unnatural. Considering these points, DeltaVR multiplayer will initially only show the user's head and hands. There is also an idea to introduce avatar customization options, like changing the player's color.

3.3 Multiplayer Interactions

For multiplayer interaction I do not plan to add much more than DeltaVR already has. My goal is to have everything working in multiplayer that worked in the original application. Aside from seeing other users, every user needs to see others use bows and paddles in DeltaVR's experiences. In addition, the users would need a way to talk to each other, so a voice chat solution must be implemented.

3.4 Photon Unity Networking

Using Photon Unity Networking by Photon makes it easier to implement networking because Photon Unity Networking comes with pre-made components that you can add to existing game objects.

Game objects in Unity are used as a container to house components that communicate with other objects or scripts. Components are attached to game objects and can serve different purposes. In the example object (Figure 1), `Transform` is a mandatory component for any game object that tracks the object's position in world space. `RigidBody` and `BoxCollider` components are used to simulate physics with Unity's physics engine and `MeshRenderer` is used to render a mesh and make it visible in world space. Photon Unity Networking offers its own custom components for game objects.

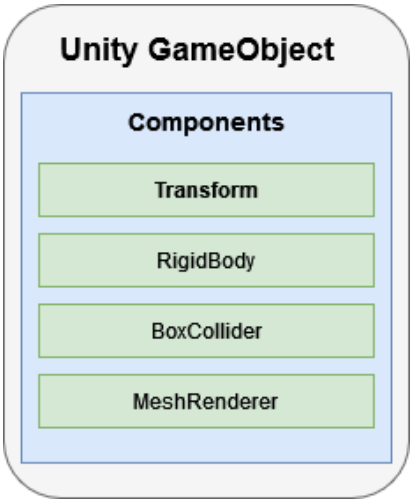

Figure 1. Structure of an example GameObject

These components cover typical multiplayer aspects that need to be tracked like position and animation. Photon also offers cloud servers, which means that testing multiple users does not need any additional servers from the university. The free version of Photon Unity Networking offers cloud servers with up to 20 concurrent players. There is also a dashboard where you can check server history, link servers to Unity projects and manage Photon subscriptions (Figure 2)

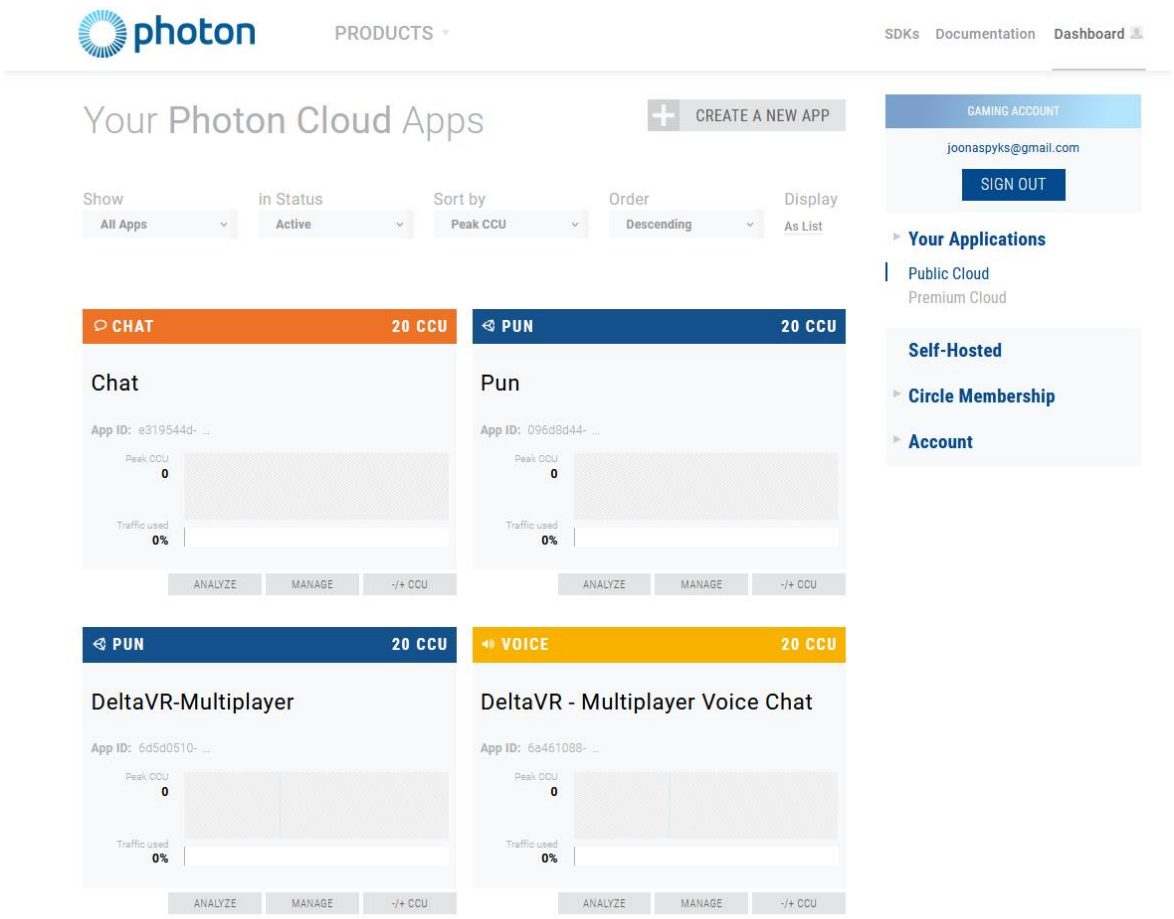

Figure 2. Photon dashboard

The main tools in Photon Unity Networking are the premade Photon components. There are different components to track certain characteristics of a Unity game object. Photon View is a general component that lets Photon Cloud receive info from the game object. Photon Transform View is a component that tracks the game object's info in the world space and updates it for all clients (Figure 3). There are different components to synchronize other information like animation and physics between clients. For any specific actions that need to be synchronized and are not covered by any Photon components, there needs to be a separate script to call remote procedure calls.

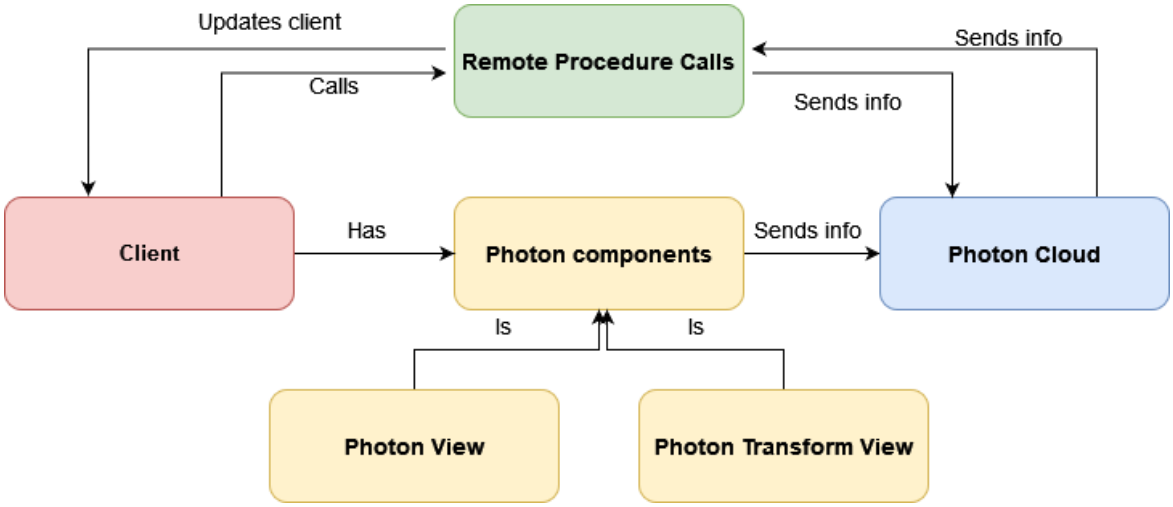

Figure 3. Diagram of Photon Unity Networking communication with cloud server

An important tool in the Photon Unity Networking toolkit is its remote procedure call (RPC) concept. Remote procedure calls are used to call methods for users other than the client in a network and are handled by Photon's servers [5].

3.5 Porting to Other Platforms

The original DeltaVR is made for virtual reality running on a Windows PC. The last few years have shown that Meta Quest 2 is overtaking the consumer virtual reality market [6]. At the time of writing, the Computer Graphics and Virtual Reality Lab has more Quest 2 headsets than any other headsets combined. The biggest advantage of the Quest 2 is that it is entirely wireless. Considering those points, we have decided that a Quest 2 port of DeltaVR is required. This means that DeltaVR must be ported from Windows to Android. Thankfully Unity makes it quite straight forward to port software from one platform to another. In the process of porting DeltaVR to the Quest 2, I will have to check if the headset and controllers work automatically or should there be a custom solution for them.

Unity has support for OpenXR, which is a framework to standardize most virtual reality input systems (Figure 4). Before OpenXR developers needed to put more effort into testing their applications on different frameworks (eg. SteamVR vs Oculus). This functionality will likely make porting to other virtual reality platforms significantly easier [7]. In addition to the Quest 2 port, this work will also result in a version of DeltaVR that does not need virtual reality. To do that, there needs to be a separate custom solution built.

Ideally, users on all the platforms would be able to join the same instance of DeltaVR and interact with each other. With Photon Unity Networking and OpenXR support, this will be possible.

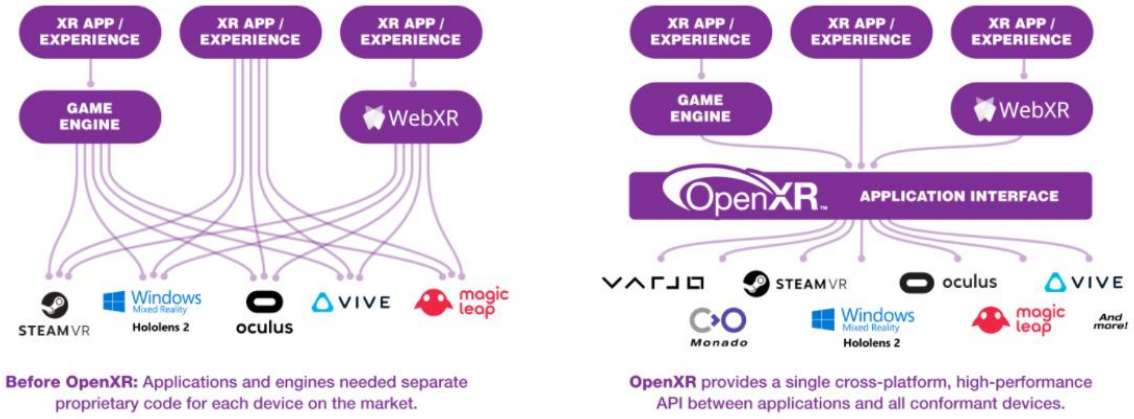

Figure 4. Diagram of OpenXR interaction with applications and platforms

3.6 Requirements for the Finished Software

To track and evaluate my progress on the multiplayer extension of DeltaVR, I'm going to catalogue some requirements. These will be referred back to in the conclusion.

RQ1	Multiplayer component to DeltaVR
RQ2	Non-VR version of DeltaVR
RQ3	Meta Quest 2 port of DeltaVR
RQ4	Port of DeltaVR minigames to multiplayer
RQ5	Functional multiplayer between platforms
RQ6	Support for more controllers
RQ7	Voice chat functionality

4. Methods

In this chapter I will go over the technology and methods used in developing the multiplayer component to DeltaVR. These can range from specific scripts to entire third-party frameworks intended to make creating a multiplayer experience more accessible and less time consuming.

4.1 Multiplayer Solutions

In my implementation, I used Photon's RPC-s for changing the player color. When a player wants to change their color, then all other players get the same update thanks to a remote procedure call.

Photon's networking toolkit also allows to set rooms to connect to and their settings. For example, you can set the maximum number of players in a room, whether it is visible or open and how long a player can be inactive before disconnecting them from the room automatically. Most of these settings will however be beyond the scope of this thesis but needed to be set up anyway.

A feature that I considered important to the multiplayer experience is voice chat. Photon also provides a toolkit to implement voice chat that is compatible with their networking toolkit. It uses a similar component-based system that communicates with Photon Cloud as a separate application.

4.2 Non-VR Version

One of the features that was in the proposed project was a non-VR version of DeltaVR. To do that, I needed to create a separate build of the game so that VR and non-VR functionalities would not conflict. To do that, I used Unity's input system to create a new input scheme specifically for non-VR users. This solution needed custom scripts for movement and other interactions, since non-VR mode would need keyboard controls and interaction in general is fundamentally different in virtual reality [8].

4.3 XR Interaction Toolkit

To make virtual reality specific interactions and controls from scratch would be very time consuming, so I am using a toolkit made by Unity called XR Interaction Toolkit. The tools in this package make the initial setup of virtual reality simple. Although the toolkit and interactions are already set up in the project, I still need to refer to these tools when I convert interactions to multiplayer.

4.4 Quest 2 Port

Handling the Meta Quest 2 port was not that complex thanks to Unity and the OpenXR framework. Unity has built-in settings to convert a project to an Android version.When converting a Unity project to an Android framework, it is suggested to pass the used textures through a texture compression algorithm to improve performance. This can be done from Unity's build menu (Figure 5). This menu is also accessed to convert a project to another platform. Further information about performance on Quest 2 and optimization methods used are outlined in chapter 5.2.

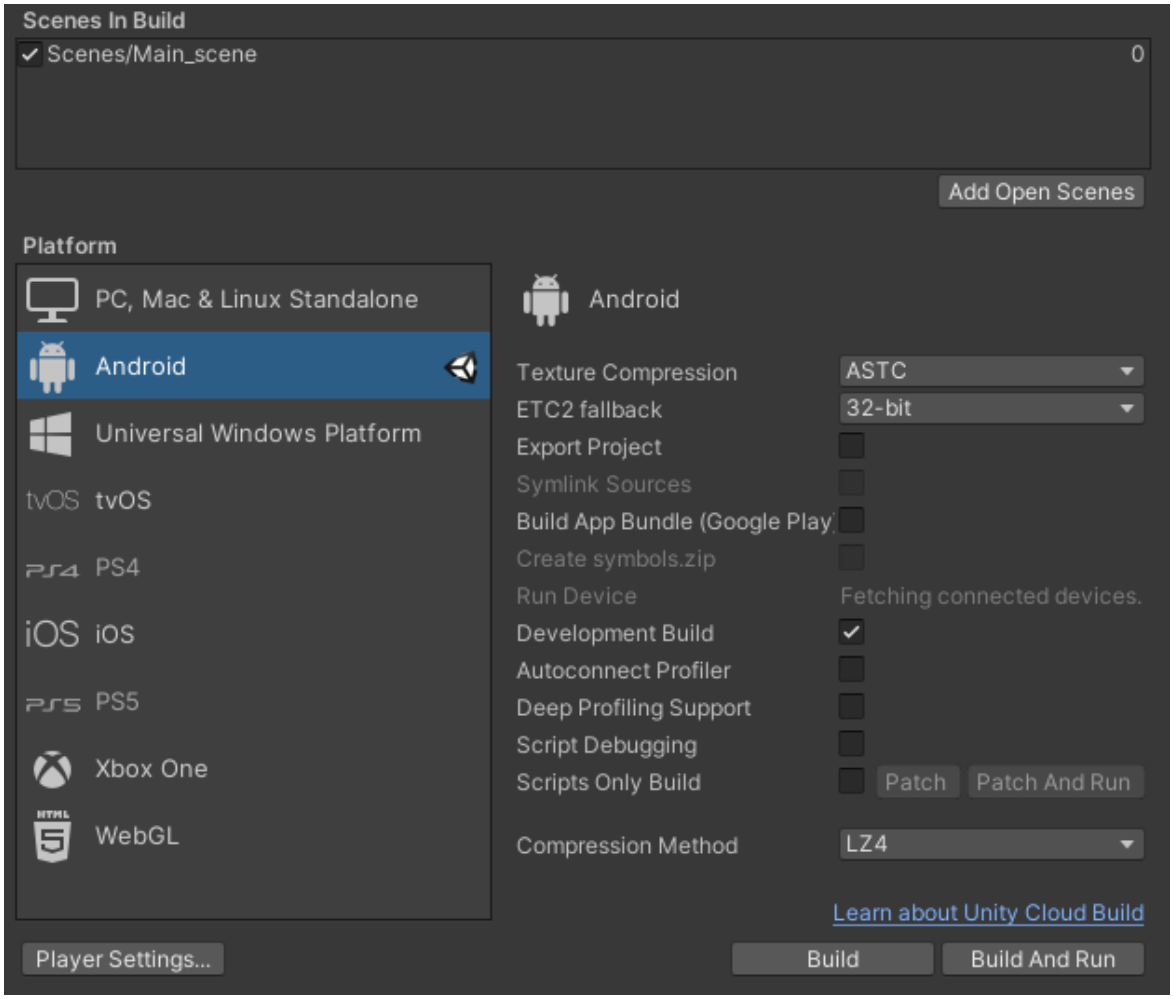

Figure 5. Unity's build menu

4.5 Minigames

One aspect of developing multiplayer functionality for DeltaVR is deciding what to do with the included minigames. At first, I tried to port both of them to multiplayer but found it difficult to use the scripts in multiplayer without rewriting the majority of code. In addition, the two minigames used different event systems. Archery range uses standard method calling with scripts and BreakoutVR uses Unity's event system. The problem is that these systems have very different approaches to be ported to multiplayer with Photon's toolkit. So, I decided to focus on the archery range, since method calls can be easily converted into remote procedure calls.

Converting regular methods to remote procedure calls means adding another method to call the procedure on the server. The logic in the initial version of the program will be put into a method with a special `PunRPC` tag that marks it as a remote procedure call. An important aspect to consider when converting these methods is which users the procedure will be called on. If the UFO-s are converted into objects that are tracked by the server, calling the method that instantiates these objects on all of the players in the server causes the range to instantiate the normal amount times the player number of UFO-s. When these objects are already tracked by the server, they only need to be instantiated once.

To have extra multiplayer features for DeltaVR, I made an activity where players can search for objects throughout the Delta building. I made the objects correlate with materials showcased in room 2007 of the Delta building so the objects would have some purpose aside from being arbitrary collectibles (Figure 7). Each object found restores a row of materials in room 2007 (Figure 6). To collect an object, the VR player has to touch it with their hand and non-VR player has to use the left mouse button. The collecting is shared between all the players so it would be a collaborative effort.

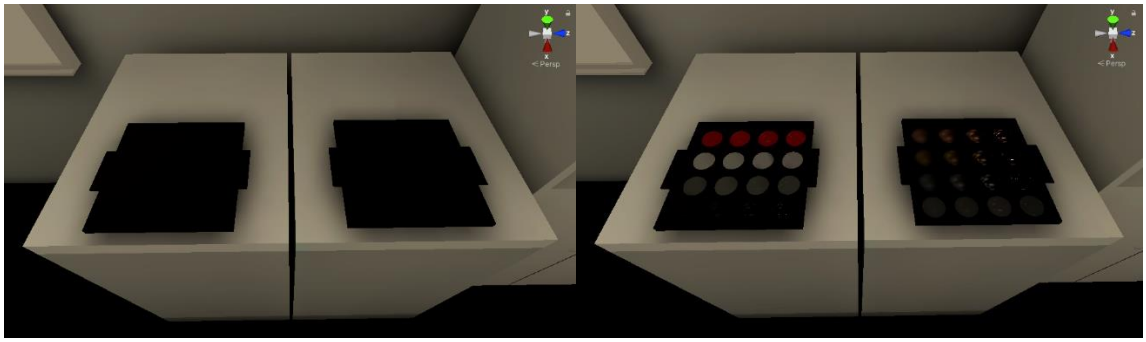

Figure 6. Materials before all collectibles (left) and after (right)

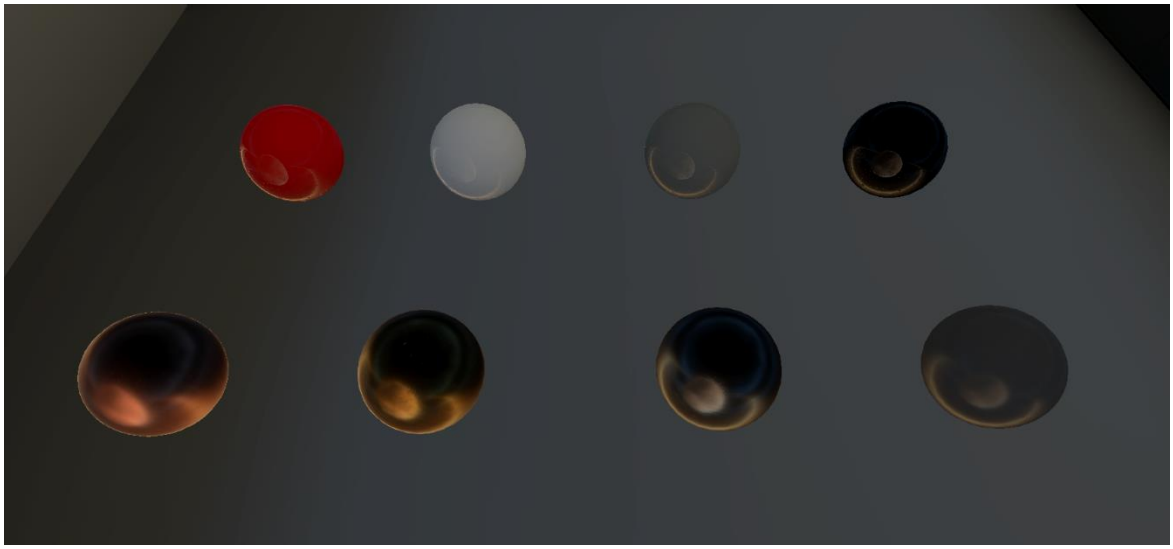

Figure 7. The collectibles

4.6 Character Customization

When multiple people are in the same space, it helps to be able to distinguish between people. To do this, I implemented a name tag and color changing system.

Each player has a wrist attached menu. It can be navigated by touching the visible buttons with their hand (Figure 8). The non-VR player equivalent of this is a regular screen overlay menu

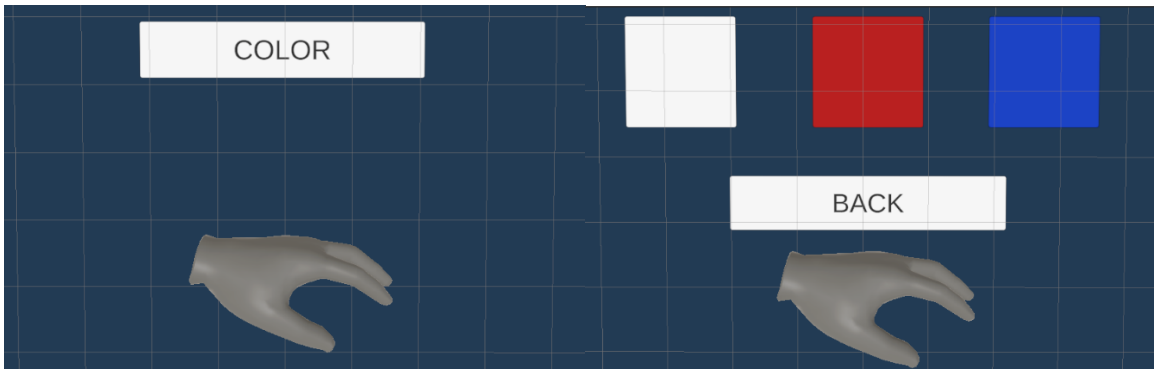

Figure 8. Wrist main menu (left), wrist color menu (right)

Another distinguishing feature between players is a name tag. It is a text hovering above the player which always faces the client player. In this version of DeltaVR, the name is set automatically to the player's `PhotonView` component id.

4.7 Difficulties

Since the initial application was not intended for multiplayer, many difficulties came from trying to port the experience to multiplayer.

Most of the problems came from the player scripts interacting with the multiplayer system. A simpler example would be if two non-VR players would join the session, there would be confusion which camera to use. That is because the way single player experiences are designed. There is only one player camera in that case. To circumvent this, there needs to be a separate script to detect the player instance that the client uses. Photon Unity Networking helps with this. With that toolkit, each player instance is assigned an id when they have a `PhotonView` component attached (1001, 2001 etc.). The toolkit also has separate methods to use for coding. The most important one to use for the previously mentioned problem is the `isMine` value, which is a boolean that indicates if the checked `PhotonView` component is owned by the client or not. So to fix the confusion between player instances, there will be an `isMine` check in a script. When it fails, the cameras of the non-client players are disabled. This method can generally be used to fix conflicts between player controls too. But some problems are not that trivial to fix:

The way that player hands were instantiated created some problems with multiplayer. The hands are created by a script on runtime. This means that attaching a networking component to the player's hands is not as simple as with other objects. Other objects could be set up to be tracked in the Unity editor. But objects instantiated on runtime need a separate script to be tracked. My fix was to create new hands for the player that are set up in the editor and not instantiated on runtime. This way the networking solutions would be consistent between objects. The main drawback to this solution is that hand animations need to be sacrificed to make this work.

Another problem that needed to be fixed is the teleport system. In the initial DeltaVR application, the only locomotion option was teleporting with a controller. I wanted to add a smooth locomotion option due to my own preferences and user choice. But adding multiplayer functionality to DeltaVR caused the teleport system to not work anymore.

After retracing the steps of making a teleport system, I eventually found what caused the issue. To make a teleport system you need to define a teleport enabled area. That is going to be the area the player can teleport to. The script that handles the teleporting logic is XR Interaction Toolkit's `Teleport Provider` script and is attached to the `Teleport Area` script on runtime start. This script also needs the `Locomotion System` script that handles all of player locomotion. Initially that script was attached to the player object's right hand along with line renderers that display the trajectory of the desired teleport movement. Since the player spawn is delayed due to the network manager, the `Teleport Provider` cannot be attached to the `Teleport Area` on runtime start. To fix this, I created a static `Teleport Provider` in the scene and attached the client player's Locomotion System to that script on client player spawn. Now the `Teleport Provider` is set up correctly and there will not be any conflicts due to multiple VR users.

While the teleport system itself works, I still could not find a way to get teleport controls working on other types of controllers aside from HTC Vive controllers. This means that it is still optimal to play DeltaVR in PCVR mode with HTC Vive controllers.

4.8 Timeline

This chapter will be dedicated to a timeline of implementations and other thesis related work. In the table below are shown specific tasks and the month or months they were worked on. In addition, there are estimated hours

The end of 2021 was primarily focused on planning out the multiplayer implementation, getting the existing project working on my personal computer and initial testing of the chosen multiplayer solution and other features needed for this project. This period had many setbacks with setting up the project. Some examples are needing to convert into another Unity input system, Blender files not working properly and virtual reality systems not recognizing DeltaVR. But with help from Toomas Tamm, the creator of DeltaVR, these problems were solved.

From the start of 2022 to May 2022, the focus was to add planned features and get the experience running as stable as possible. This consisted of adding a player color change menu, creating objects for players to collect and constant testing and bug fixing.

July 2021	Thesis topic choice

August 2021	Initial talks with supervisor (2h) Setting up initial DeltaVR build on personal machine (18h)
September 2021	Creating a non-VR player for DeltaVR (8h)
October 2021	Migrating to a newer DeltaVR version and changing the Unity's input system (12h) Researching different multiplayer solutions for Unity (8h)
November 2021	Implementing initial multiplayer solution with Photon Unity Networking (12h)
December 2021	Creating a Quest 2 version of DeltaVR (8h) Initial testing of current solution on different types of hardware (PCVR, PC non-VR, Quest 2) (12h)
January 2022	Initial testing of current solution on different types of hardware (PCVR, PC non-VR, Quest 2) (12h) Player menu with color changing buttons (8h) Testing multiplayer between PCVR and non-VR player (24h)
February 2022	Testing DeltaVR minigames and multiplayer (12h)
March 2022	Testing between multiple VR players (12h) Polishing and bug fixing (24h)
April 2022	Treasure hunt (6h) Polishing and bug fixing (24h)
May 2022	Polishing and bug fixing (8h)

5. Testing

After most of the new features were implemented in DeltaVR, I decided to test the performance of the Quest 2 build and gather opinions on implemented features. I will also go through the process of debugging the app.

5.1 Debugging Process

When developing software for virtual reality, the debugging process can be different than debugging regular software. And when you add multiplayer on top of that, the testing process is unique to this type of software. The most obvious difference is the hardware required. I own the needed hardware to test the PCVR version and the University of Tartu loaned me a Quest 2 for thesis work. Thanks to that I could test the application without needing to use the Delta Center CGVR lab.

The cross-platform multiplayer component of DeltaVR means that testing needed to account for all the combinations of builds. This means cross testing PCVR, PC non-VR and Quest 2 builds at the same time. To do that, I sometimes needed external help in testing.

5.2 Performance Testing

While porting DeltaVR to Quest 2 it was obvious that the performance was not optimal for virtual reality. Using Oculus Developer Hub to track performance, the framerates of DeltaVR on the quest averaged 25 frames per second. It was about 5 frames higher or lower depending on the area. This is far below the 72 frames per second recommended by Oculus [9].

DeltaVR uses Unity's Universal Render Pipeline. To change quality settings you need to find the render pipeline asset and change the settings there. Lowering these settings (disabling shadows, lowering anti-aliasing quality etc.) raised the average frames per second to about 35, still well under recommended levels. Even disabling transparent materials did not give a significant boost to FPS. An extreme measure to boost FPS would be to decrease the render scale of the pipeline asset. But that will drastically decrease the visual quality. So at the time of writing, DeltaVR for the Quest 2 remains at sub-optimal levels of FPS.

Figure 7. DeltaVR on PCVR (left), Quest 2 version with performance stats (right)

5.3 User Testing

While I did not conduct a specific testing session to get data about the multiplayer implementation, I still noted feedback when DeltaVR was played by my supervisor or other people. This helped me find more bugs and refine the overall experience of multiplayer.

Examples of issues found while user testing are player controls being disabled when one player leaves the session, duplicate hands spawning on the joined player's position that are only seen by the client player and Unity's physics system influencing the player's movement in certain situations.

As for general feedback, I got different perspectives on what people might want out of a multiplayer environment. Some people are satisfied with a static recreation of a building but others would like there to be more small interactive elements in the virtual space. This can be considered for any future work with DeltaVR or other virtual reality applications.

6. Conclusion

The results of this thesis work are three builds of DeltaVR – One for virtual reality on PC-s, one for PC use without virtual reality and one on the Meta Quest 2. These builds have cross compatibility with multiplayer so each version user can inhabit the same space.

When going through the requirements in chapter 3.6, I managed to implement most of them in some capacity. RQ1 and RQ5 were more realized than others. That is acceptable, since these requirements were the main focus of the thesis. As for other requirements, they had varying degrees of success. RQ2 was technically implemented but the non-VR version does not have the same functionalities as the VR version. RQ3 is also functional but its performance is not optimal. RQ4 proved to be more difficult than I thought due to different event systems used in minigames. The archery range is visually tracked but still only one person can play at a time. RQ6 is covered mostly by OpenXR but I could not find a solution to make the teleport system work with other types of controllers. RQ7 was not difficult to implement, since Photon provides a voice chat framework in addition to their multiplayer toolkit.

The result of this thesis still leaves a lot to be improved for DeltaVR and its multiplayer solution. There were multiple sacrifices made in porting DeltaVR to multiplayer. Few examples of this would be disabling hand animations and teleport system not working on other controller types. These features can be reimplemented and issues fixed when given enough time and effort.

It would also be beneficial to conduct a larger test of DeltaVR multiplayer to evaluate, if Photon's cloud servers can handle up to 20 people with the current implementation and if any other problems will arise from a large amount of people.

There were also considerations for an Unreal Engine port of DeltaVR. This was mostly due to Unity's licensing practices. But Unreal Engine also has more comprehensive and approachable first-party multiplayer solutions. This idea was set aside mostly because porting the entire application to another game engine and also creating multiplayer would likely be too time-consuming for a bachelor's thesis. An alternative to Unreal Engine would be Godot. The advantage to Godot would be its open-source nature.

Overall, the work done for this thesis can be a good foundation to more multiplayer specific additions and interactions to DeltaVR. Since basic player movements are already tracked, further additions would likely not have to worry about that and focus on more specific goals.

References

1. Peeter Paal, Virtuaalreaalsuse seadmete uuring, University of Tartu Computer Science, 2021 Year BSc thesis https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=71654&year=2021 (09.05.2022)
2. Toomas Tamm, DeltaVR, University of Tartu Computer Science institute 2021. Year BSc thesis https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=71682&year=2021&language=en (10.12.2021)
3. Rychkova A, Korotkikh A, Mironov A, Smolin A, Maksimenko N, Kurushkin M. Orbital Battleship: A Multiplayer Guessing Game in Immersive Virtual Reality. J Chem Educ. 2020 Nov 10;97(11):4184–8.
4. Born F, Sykownik P, Masuch M. Co-Located vs. Remote Gameplay: The Role of Physical Co-Presence in Multiplayer Room-Scale VR. In: 2019 IEEE Conference on Games (CoG) [Internet]. London, United Kingdom: IEEE; 2019 [cited 2022 Feb 9]. p. 1–8. Available from: https://ieeexplore.ieee.org/document/8848001/
5. Photon RPC and RaiseEvent documentation, https://doc.photonengine.com/en-us/pun/current/gameplay/rpcsandraiseevent (09.05.2022)
6. Steam's hardware survey featuring VR hardware statistics, https://store.steampowered.com/hwsurvey/Steam-Hardware-Software-Survey-Welcome-to-Steam (09.05.2022)
7. OpenXR 1.0 overview, https://www.khronos.org/openxr/ (30.04.2022)
8. J. Jerald, The VR Book: Human-Centered Design for Virtual Reality, First edition, ACM Books, 2016
9. Oculus performance and optimization guide, https://developer.oculus.com/documentation/unity/unity-perf/ (04.05.2022)

Appendix

Links to considered multiplayer solutions for Unity

Unity Multiplayer Networking: https://docs-multiplayer.unity3d.com/

Mirror: https://mirror-networking.gitbook.io/docs/

Photon Unity Networking: https://doc-api.photonengine.com/en/pun/v2/index.html

Head model used: https://sketchfab.com/3d-models/low-poly-head-6b6a762bd8b34b6d9d46d44129b55037

DeltaVR – Multiplayer on Gitlab: https://gitlab.com/Joonasp1/deltavr-multiplayer

Builds of DeltaVR – Multiplayer: https://drive.google.com/drive/folders/1Mad5BAZq-PvKO7UseP6raxFJ3jhKEj9m?usp=sharing

License

Non-exclusive licence to reproduce the thesis and make the thesis public

I, Joonas Püks,

1. grant the University of Tartu a free permit (non-exclusive licence) to

reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, my thesis DeltaVR - Multiplayer, supervised by Ulrich Norbisrath.

2. I grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in points 1 and 2.

4. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Joonas Püks

10/05/2022