Graduation Report

NETWORKING AND VR

Ana-Maria Ilea
Saxion University of Applied Sciences
VR Amsterdam
Graduation 2022/2023
474817@student.saxion.nl

Table of Contents

Abstract

This report documents the research and development process used to create a multiplayer system and framework. In collaboration with VR Amsterdam, the researcher, along with two other graduation interns, created a multiplayer game played in mixed reality using Unity. The project uses Oculus Meta Quest Pro headsets due to the improved passthrough feature. The final deliverable refers to a proof of concept or a demo of the game, having most of the features implemented and working, though perhaps not completely polished. The author researched different networking solutions and documented the development of the framework and the gameplay.

Introduction

Overview

The Graduation Module is a requirement for fourth-year students at Saxion University of Applied Sciences who are enrolled in the Creative Media and Game Technologies (CMGT) programme. Each student must apply to a company and do a graduation internship. A project will be given to the student during the graduation internship. The student must then select a research topic which is related to the project given. The student will research the topic chosen and write a report detailing the design process.

The researcher chose VR Amsterdam for the graduation internship. The student worked in a team of three students: two programmers and one designer/artist. The group assignment for the team is to research, design and develop an arcade-style multiplier game which uses the passthrough technology of the new Oculus Meta Quest Pro.

Terminology

Virtual Reality (VR) allows the player to enter and interact with a computer-generated environment using a VR Headset. One of the problems VR faces is the lack of realism of the applications. Augmented Reality (AR) aims to solve this problem by adding computer-generated objects to the real world. The separation between the real and virtual worlds decreases user immersion during AR sessions. Mixed Reality (MR) tackled this challenge with the creation of a Mixed Reality environment. This environment uses both the real and virtual worlds to create immersive gameplay (Rokhsaritalemi et al., 2020).

The Oculus Meta Quest Pro makes use of colour cameras to create a true MR experience. The Headset uses a stereoscopic colour passthrough solution with custom hardware, computer vision and machine learning to create an environment that naturally simulates the real world. The camera and display resolutions improve image quality and make the environment natural for the user's eyes (*Introducing Meta Reality: A Look at the Technologies Necessary to Convincingly Blend the Virtual and Physical Worlds | Meta Store*, 2022).

Team

The students researched the market on the intended audience and create a shipment-ready game that uses extended reality (XR) to create immersion. The students also chose different topics within the assignment and researched potential users for it. The topic chosen by the research is networking and the creation of a multiplayer framework which can sustain mixed reality.

The team is composed of three graduation interns: Ana-Maria Ilea – network and gameplay programmer (author of the paper), Kristyna Pavlatova – AR and gameplay programmer and Marc Garcia Pastor – designer and artist. The students worked on their research topic and on the group project at the same time. The goal of the students is to conduct their research and create an interactive product.

Empathize

Company Outline

The client is VR Amsterdam. The company is a VR Entertainment centre located in Amsterdam. Their mission is to amaze visitors by showing them VR experiences and to make their visit memorable. The company was formed in 2017 and currently contains two employees. VR Amsterdam works with freelancers (E.g., former Google or Magic Leap employees) and engages in business-to-business activities. VR Amsterdam works with companies such as Vertigo Games, Black Light or Phenomena to acquire the games in the arcade. Within the arcade, there are multiple games within a variety of genres. The games at VR Amsterdam include but are not limited to VR escape-room games, racing games and free-roaming shooter games. The equipment used for playing the games includes VR Treadmills, VR Gloves and VR Headsets (Oculus Quest 1 and 2, HTC Vive Focus 3) (*Onze Games - VR Amsterdam*, n.d.).

Company Objective

VR Amsterdam noticed in recent years that people are becoming bored with VR and want to experience something new. For this reason, VR Amsterdam wants to challenge the students to create a Mixed Reality multiplayer game using the passthrough feature of the Oculus Meta Quest Pro. The game must be free-roaming and fast-paced.

Product The Client Needs

The initial assignment was a fully polished, shipping-ready game. As development progressed, however, the became clear that the task was not feasible. As a result, the final product was a proof of concept. The game must be played on the Oculus Meta Quest Pro in a 10X10m play space. The game must be a multiplayer game which uses passthrough technology. Thus, the user must be able to interact with both the real world and the virtual world.

Market

The VR Amsterdam Entertainment Centre is located in Amsterdam. Because of this, the market can be divided into two categories. The first one is the market within the Entertainment Centre, referring to the games that can be played there currently. The second one is the market within Amsterdam, referring to the games that can be played in other VR arcades located in the city. In addition to the VR games within the arcades, SideQuest offers the market for most pass-through games.

VR Amsterdam Arcade:

VR Amsterdam has four areas where VR games can be played. Two of the areas are free-roaming and the other two are reserved for a racing simulator and a VR Omni Treadmill. In the free-roaming areas, four games can be played: VR Zombie, Eclipse, Ghost Patrol and Free-Roaming Laser Game. VR Zombie and Free-Roaming Laser Game are multiplayer shooter games in VR. Eclipse is a VR Escape Room-like multiplayer game. Ghost Patrol is a VR family-friendly adventure multiplayer game where the player shoots and colour-matches ghosts. The VR Omni Treadmill includes the game Robot Arena. The game is a multiplayer shooter game that is played on a treadmill. This way the player can move in a virtual space without the need for the real world.

The arcade has games three shooter games, based on both cooperation and competition, an escape-room-like game, a family-friendly adventure shooter game and a racing game. The games are played in VR and can be free-roaming or stationary.

Other Arcades:

In Amsterdam, there are five VR arcades, excluding VR Amsterdam. These are VRGH Arena, LightningVR, My Escape Club, A'Dam VR Game Park and VR Experience Amsterdam. The majority of games provided by these arenas are multiplayer shooter games. The most common shooter game is a VR Zombie Arena. In this game, the players team up to shoot as many zombies as possible. Other shooter experiences involve a battle royal-like game (E.g., VR Tournament at VRGH Arena), player vs player (E.g., Cops & Robbers at VR Experience Amsterdam) and kid-friendly shooter (E.g., Toon Strike at LightningVR). VR Escape rooms are also present in a few arcades (E.g., VR Escape Room at A'Dam VR Game Park). VRGH Arena additionally has the game VR Polyfun. The game is a multiplayer adventure in which the players compete against each other for the best score. The game is a non-violent experience where the player overcomes obstacles to gain score points.

SideQuest:

SideQuest is an application that allows developers to enable different functionality on an Oculus VR Set. Through SideQuest, one can download and install new content on the VR Set that might not be available through the Oculus official store (Baker, 2021). Most games that use the passthrough feature of the Oculus are available through SideQuest. One example is the game Spatial Ops from Resolution Games – a multiplayer shooter game in mixed reality. The players can team up or fight against each other in a first-person shooter game (*Spatial Ops — Resolution Games*, n.d.-a). Other games that use mixed reality don't interact with the real environment. The mixed reality aspect is dome to increase immersion and not necessarily to add gameplay elements.

Industry standards

VR games in arcades are mostly used with a computer. The computer is in a backpack and a VR Headset is connected (*Exclusive Experiences |ENG| VRGH Arena|Virtual Reality Arena in Amsterdam Westpoort*, n.d.). The games in the arcades are played entirely in VR, with no AR features.

AR applications are mostly used with the help of a smartphone. This is because smartphones have better cameras with different features (E.g., depth perception). These features in combination with AI and Computer Vision can create gameplay using the real environment.

Both AR and VR games are mostly made with Unity. Unity has more functionality which is either available or already integrated and is overall more optimized for these platforms (*Unity vs Unreal for VR/AR! Which Engine Should You Choose?*, n.d.).

The VR Entertainment Centres have mostly multiplayer games. These games can be both fast-paced (E.g., a shooter game) and slow-paced (E.g., an escape room). The games use computer servers to host and run the games on Local-Area Network (LAN) to give the players the multiplayer experience.

Trends

The most popular genre of games in VR arcades is shooter games. Multiplayer games based on either competition or cooperation create the most immersive experience. Therefore, it is reasonable to believe that most arcades would like to add more fast-paced games based on competition or cooperation.

Shooter games are also the latest trend for mixed reality. Such a game will involve the use of the real world and virtual objects to create a playable environment (*Spatial Ops — Resolution Games*, n.d.-b).

End-user

Game End User:

The final game will be played in the Entertainment Centre. Therefore, the end user is part of the target audience of VR Amsterdam. The audience includes families and groups of friends. The game will add to the experience offered by VR Amsterdam. The target audience can also include arcade lovers and young professionals between the ages of 20 to 43 years old since all of the games belong to the genre.

Research Topic End User:

The end user for the research topic would most likely be developers and arcade managers. The networking system could be used in other projects made by other people. People who are interested in making a multiplayer game which uses mixed reality could make use of the final system made for the game. Arcade managers can also be included in the target audience. While the system can be used by developers to create new applications, arcade managers will have to set up the game at the end of each round. Arcade managers, therefore, need to be taken into consideration when testing the user experience and interface of the game and system.

Theory

Mixed reality is a relatively new concept and technology. It was introduced with the Oculus Meta Quest 2, where the user can see a black-and-white version of the real environment. The Oculus Meta Quest Pro gives a more accurate representation of the real world. Because the technology is new, sources and use cases are currently limited.

The newest feature regarding mixed reality is the possibility of colocation and map-sharing. With one headset, the real world can be recorded and mapped with the help of spatial anchors. The map can then be uploaded to a Meta cloud and downloaded by other players (Heaney, 2023). This allows the users to have correct placement on the same map.

Visit at VR Amsterdam

Networking Setup

Initially, the team believed that the arcade had a dedicated server which runs the server applications of the games. This would have made the games fully online multiplayer. However, this is not the setup used at VR Amsterdam. The setup consists of two cases.

The first case includes the use of Oculus headsets. In this case, a Windows application for the server and an Android application for the Oculus headset are required. The server runs on a computer, enabling a LAN multiplayer system. The client builds run on the VR headsets and connect to the server computer. The computer application has a lobby system which allows the arcade manager to set some aspects of the game (E.g., number of players, game mode, map, map size) as well as see the connected headsets with some information about them (E.g., the headset ID, the headset IP Address). The client lobby system involved an empty room in which the players wait for the setup of the game to be completed. When the setup on the server is complete, start points become available for the clients. When the clients collide with the points, the game starts. While the game is played, the server has a bird's eye view so that spectators can watch the session.

The second use case involves the use of HTC Vive headsets. In this case, two Windows applications are needed, one for the server and one for the clients. Both the server and the client builds run on a computer. Because of this, a computer is needed for each client (or VR headset) who would take part in the game. The server runs on a different computer, enabling LAN multiplayer, similar to the first case. The server has a lobby system as well for the management of the clients and the game setup. The lobby for the clients is an empty room with a start button which must be pressed for the game to begin.

Game Room

The team was shown the room in which the final game would be played. The room has an unusual shape, with different corners. Within the room, there are also pillars and a hose box (Figure 1). The layout gave the team hope that the game could be successful and the free-roaming aspect would be fun to play.

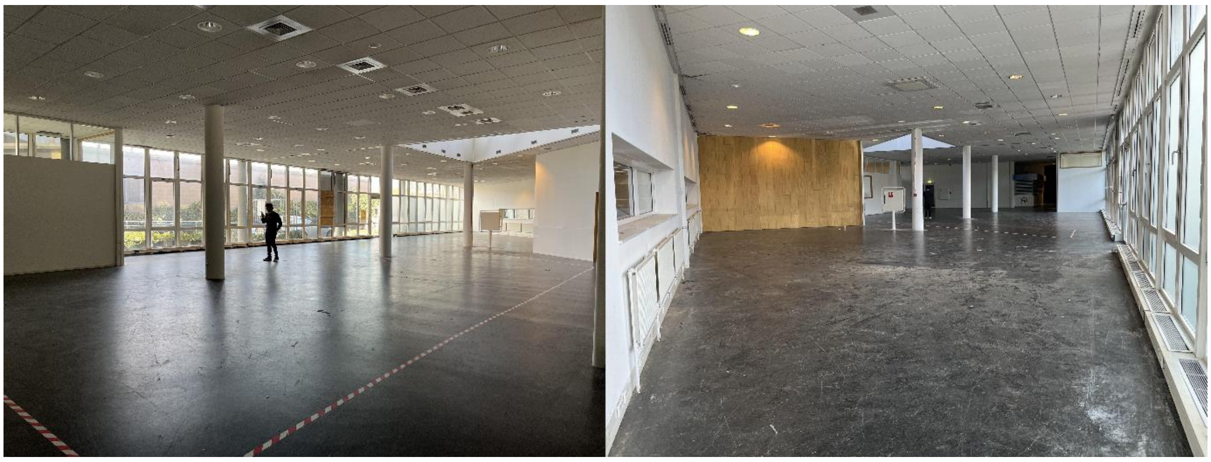


Figure 1 - Room at VR Amsterdam

Mixed Reality Game

After playing the game at VR Amsterdam, the team along with the client visited Purmerend to play Hado. Hado is a multiplayer game inspired by Dodgeball, played in mixed reality (*Arena Amsterdam (H2o Campus) - HADO Benelux*, n.d.).

The server runs on a computer and, similar to the setup at VR Amsterdam, has a lobby system for setting up the game. In the setup, the arcade master can change the number of players, team setup and enable or disable AI characters. The game can be played with either people, AI or a combination of the two.

The game is played by making gestures with the players' right arms. Depending on the movement, the players can shoot a ball or create a shield. The players avoid the balls either by hiding behind the shield, or by moving out of the way of the balls. The arena is divided into two areas, one for each team. When a player enters the other team's area, they will get a message telling them to go back to the correct area.

Conclusions

After the meeting, the following conclusions were made:

1. The project includes the use of Oculus headsets. Thus, the setup of the multiplayer would be similar to the first case noticed at VR Amsterdam, with one Windows application for the server computer and one Android application for the Oculus headsets.
2. Within VR Amsterdam there are no online multiplayer games. Thus, the multiplayer system must rely on LAN either through a peer-to-peer system or a dedicated server.
3. Having a lobby system for the arcade managers is important. This allows the games to be scaled, potentially including more content. A lobby would allow the managers to select different maps or game modes, which gives control over how the game is played.

Define

Problem statement

With the new advancements in mixed reality, VR Amsterdam is looking for a fast-paced, mixed-reality multiplayer game played using the Oculus Meta Quest Pro to add to its entertainment centre. The game must stand out among multiple VR arcade games, such as shooter games and escape-room games to create value for the company. A multiplayer game is difficult to set up, both during development and after release. Consequently, an optimized multiplayer framework, which is intuitive to the user, is needed to develop a performant game and set up a playable session after the release.

Research goals

Research which networking solution is suited for the project based on documentation availability, implementation potential and scalability options.

Research whether humanoid collisions will provide more accurate information about the player's position than the data from the VR headset.

Research if client-authoritative player movement is more suited for a VR/mixed reality game than server-authoritative movement.

Research the potential lag and latency of the chosen networking system and possible solutions to improve performance.

Main research question

How can a networking framework be optimized for best performance to include modular gameplay, different game modes and player states, while having a good user experience for arcade managers during the setup of the final game?

Scope

Deliverables:

The final deliverable will be a networking framework which can be used to play a session of a multiplayer game in mixed reality. The system must be modular to allow for different gameplay elements to be added. The networking framework is intended to be used by developers during the production of a multiplayer game. The user must be able to set up sessions and reset the game quickly and easily.

Inclusions:

The system must support up to six players and offer LAN multiplayer. The final game will be a shooter game. The system must support shooter mechanics. Consequently, humanoid collision should be implemented at least for the upper body to make the experience realistic. The game must run in mixed reality. Therefore, the system must support networked player movement using VR, while showing the real environment.

Exclusions:

Local multiplayer will not be implemented since the game will be played through LAN multiplayer. Therefore, cloud online multiplayer will not be included, as VR Amsterdam is using computers to play sessions, not an uploaded server. Humanoid collision for the lower body will not be implemented as the upper-body humanoid collision will provide enough interaction between players.

Assumptions:

The arcade manager will know how to start the game. The playable area will include a space in which the game gets reset. The system is expected to be intuitive for the arcade managers during the game start and for developers during production.

Constraints:

The Oculus set requires playable boundaries. When these are not set, the headset opens a window asking the user to set boundaries. This may be a problem if the playable area is bigger in size than the maximum size of the Oculus boundaries.

Concept

The concept is inspired by Pokémon Go and it is divided into two parts. The first part encourages the player to interact with the environment. Through this interaction, the player can grab different objects to befriend different creatures. The creatures have different types and require different items to be befriended (Figure 2).

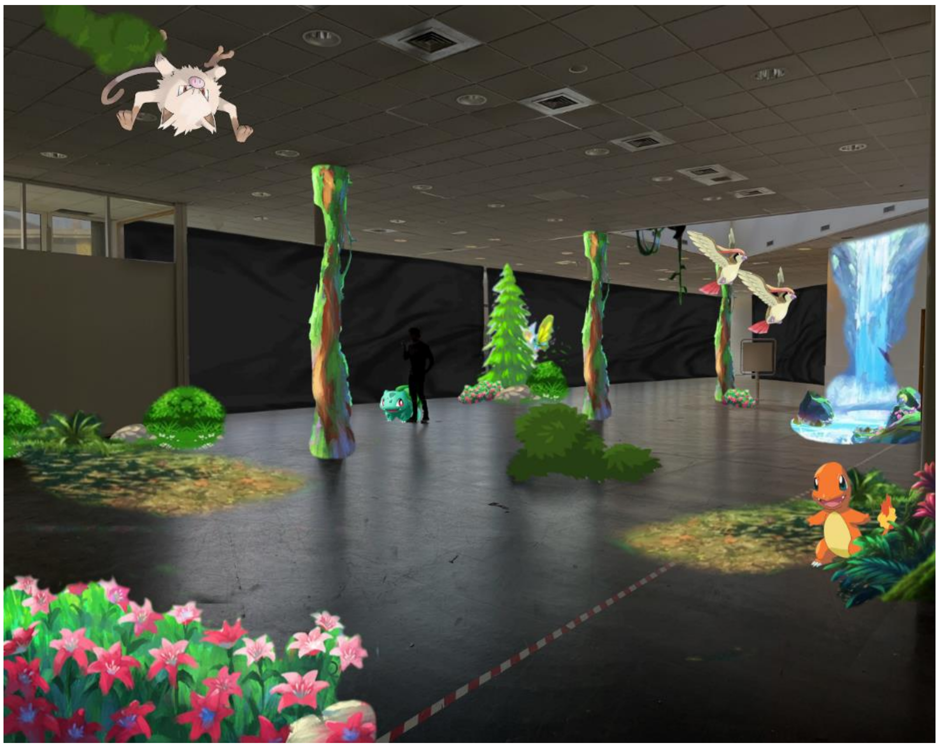


Figure 2 - Part 1 Concept - Made By Kristyna Pavlatova

The game switches to the next part only after every player collects every type of creature. When the second part starts, the players cannot grab items anymore and instead are able to shoot either projectiles or a stream of particles. The player learns about the shooting mechanics and the life system. The second part can either be a boss fight or a player vs player (Figure 3). The game can have parts in between the first and second parts, such as the team assignment in the Player vs Player mode.

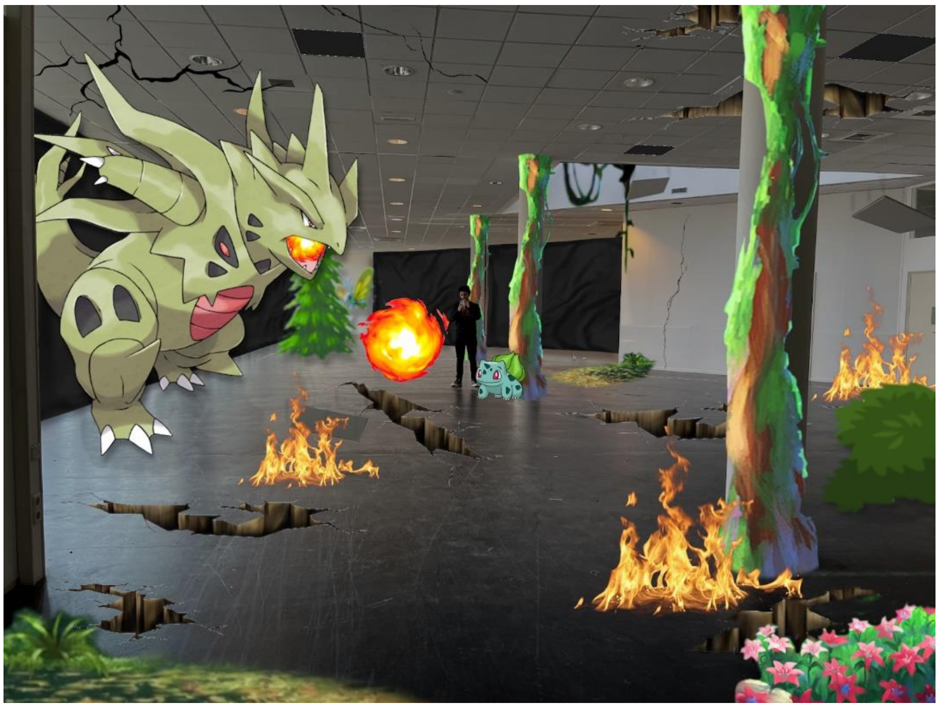


Figure 3 - Part 3 Concept - Made By Kristyna Pavlatova

Version Definition

Planning

The planning included three versions: a first version, a minimum viable product and the final deliverable. The team worked in sprints lasting one week, totalling 18 sprints. The team and the researcher used Jira to keep track of the tasks and plan accordingly.

The **first sprint** or first two weeks was dedicated to introductions and general research. The team met with the client and divided tasks and roles. Every member of the team researched different engines and what functionality is available for the creation of a mixed-reality game. The team researched the Oculus Meta Quest Pro as well along with what features are available to work within each engine. The researcher gathered preliminary research on networking solutions. This resulted in a list of potential directions the networking system can go into for both the Unity Engine and the Unreal Engine.

During the **second sprint**, the team defined the project. This included defining the problem statement, the research goals, the scope and the research questions.

The **third sprint** was reserved for brainstorming and concepting solutions for the final deliverable. During this print, the game design document, the technical design document and the art asset list were created.

After the third sprint, the team divided the project into three versions: a first version, which acts as a minimum viable product, a second version used to test additional gameplay and functionality and a third version, which represents the final deliverable.

After the third sprint, the planning followed the research questions. The research questions are divided based on the overall project planning. The minimum viable product must include most if not all the game mechanics and features. The mechanics must be networked properly and users must be able to play the game from start to finish.

The **fourth sprint** focused on the first and second research questions. The researcher compiled a conclusive list of networking solutions for the Unity Engine. After the list was complete, the researcher chose the two best solutions (FishNet and NetCode), compared them and consequently choose the solution used to make the final project. The final solution was also tested with the passthrough feature of the Oculus, to make sure that a multiplayer mixed reality game is possible.

During the fourth sprint, the team visited VR Amsterdam to take a look at the space in which the final deliverable will be played. The team also took the opportunity to play some of the games at the arcade and the mixed reality game Hado to get an idea of how it feels to play a VR and MR game.

The **fifth and sixth sprints** were dedicated to gameplay and feature implementation. At the beginning of the fifth sprint, the final concept was written down and the final game mechanics were decided on. The planning called for the gameplay to be first implemented in single-player and in VR (Figure 5). The features implemented include the grabbing core mechanic, the shooting core mechanic, friendly creature behaviour, item interaction and different game modes (Appendix A). The single-player gameplay was planned to be implemented by the end of March.

The **seventh and eighth sprints** were planned for networking the gameplay implemented in single-player. During these sprints, the researcher focused on the third and fourth research questions. The gameplay was networked by the first half of April.

During the **ninth sprint**, the researcher along with Kristyna Pavlatova, tested the networking of spatial anchors. Consequently, tests relating to player movement and position were conducted. Finally, the networked gameplay was tested with two players.

The **tenth sprint** tested the feasibility of humanoid collision. With the completion of the prototype, the fifth research sub-question was answered. During this sprint, the boss fight was implemented and networked.

As the first playtest was scheduled on the 26th of May, the **11th, 12th, 13th and 14th sprints** were scheduled for testing and fixing bugs. During these sprints, the team made a test map in which two players were able to play the game from start to finish. The team planned for two days at VR Amsterdam. On the first day, the team performed internal testing. This meant that the team would test the map and the models were scaled and placed correctly in the scene so that they align with the real world. On the second day, the team tested the game with the target audience and gathered feedback.

During the **15th and 16th sprints,** the team implemented the feedback gathered. The player vs player mechanic and the server lobby were implemented as well.

The **last sprints** were dedicated to finishing the project and writing proper documentation.

Game versions and iterations

The end product includes three versions determined mostly by the dates of the scheduled playtests. The first version of the game was finished by the end of April. The version included all the game mechanics, except for the boss fight and the player vs player mechanic.

The second version included the boss fight for the second part of the game. The version also included a client lobby area in which the players can calibrate the map. The first playtest was performed on the 26th of May at VR Amsterdam. Therefore, the second version had to be played from start to finish without game-breaking bugs.

The third version included the player vs player mode and the server lobby implementation.

Research sub-questions

Networking solutions theory

Question:

1. What networking solutions are available for Unity?

Source and type:

Secondary, qualitative and quantitative, the research was found online (E.g., documentation and YouTube tutorials)

Initial Approach:

Desk research and comparing systems. Desk research was made to gather information about the different networking engines which work with Unity. The researcher read the documentation of the engines and watched different tutorials which use different solutions.

Expected Results:

The result was a list of networking engines which work with Unity. The two best solutions were used for the second sub-question.

Conclusions:

Following the development of the research question, the researcher formulated the following list of networking solutions:

- Photon:

Photon or PUN is a networking solution for Unity. The solution has different licences for different products. The free licence can sustain 100 players connected at the same time. However, the free licence cannot be used for commercial products (*Licenses | Photon Engine*, n.d.). Because of this, the team decided not to use this networking solution.

Photon provides different services such as matchmaking or cloud hosting. Photon includes custom MonoBehavior scripts named MonoBehaviourPunCallbacks. These scripts implement different functionality such as custom callback functions (E.g., Awake, Start) (*Introduction | Photon Engine*, n.d.). PUN also has custom remote procedure calls. A remote procedure call is the way to communicate between the server and the clients. Photon has one RPC, named PunRPC with a parameter which specifies who the receiver is. To call the RPC a component named PhotonView must be used (*RPCs and RaiseEvent | Photon Engine*, n.d.). The syntax provided the team with another reason for looking into other networking solutions.

- Mirror:

Mirror is one of the most used networking solutions for Unity. Mirror consists of a high-level networking library, originally based on the UNET framework, made for building multiplayer games in Unity. The system is free to download on the Unity Asset Store and received regular updates as the Unity Engine changes (*Mirror | Network | Unity Asset Store*, n.d.).

Mirror includes a network manager and is generally server authoritative. The system can build server, client and host builds. The server can be a dedicated server or a host server and provides several transport channels for the information. Mirror offers three remote procedure calls (RPC): a clientRPC, a serverRPC and a targetRPC (*General - Mirror*, n.d.). A serverRPC is sent from the client so that the server performs an action. A clientRPC is sent from the server to the client so that all the clients perform an action. A targetRPC is similar to a clientRPC. It is sent from the server to a specific client (a target) so that only that particular client performs an action. Because of the use of the separate targetRPC, the code can be more challenging to structure as the methods need to be separated into client and target RPCs. Additionally, a custom network manager must be made for different stages of the game (E.g., a lobby network manager and a game network manager). This is done to have access to spawning information.

- Fishnet:

Fishnet is a free networking solution built on the Mirror framework. Due to this, the functionality is very similar. Fishnet provides a network manager which can handle the connection and spawning of players. The system includes the same RPCs as Mirror. The system is server authoritative and allows the use of dedicated servers (*Introduction - Fish-Net: Networking Evolved*, n.d.). The setup seems to be easier than Mirror's so Fishnet was chosen for the second research question.

Fishnet does require the Fish Networking Discovery add-on to implement LAN as the system is optimized for dedicated servers. There are other add-ons compatible with Fishnet, both free and paid, which can give developers additional functionality such as syntonised particles or audio clips (*Add-Ons - Fish-Net: Networking Evolved*, n.d.).

- Netcode:

UNET was the networking framework provided by Unity. This solution has been deprecated and it is no longer used by developers. Instead of this framework, Unity Encourages developers to use Netcode for GameObjects, their new networking and multiplayer solution(*Unity - Manual: Multiplayer and Networking*, n.d.).

Similar to Photon, Netcode offers different services which can be used for further development (E.g., matchmaking, cloud hosting, authentication). The setup is fairly simple. A developer would make a player and add the custom networking manager. The system includes two RPCs: a clientRPC and a serverRPC. Each of the RPCs can include parameters which contain information about the sender and the receiver. The parameters remove the need for a targetRPC, as a function can be called to run on specific players. Because the networking solution is new, tutorials and examples can be scarce.

Based on the information available and the needs and preferences of the team, **Fishnet and Netcode** were the two networking solutions chosen for the next research question.

Chosen networking solution theory

Question:

2. Which networking solution will enable the best workflow for developing the networking framework?

Source and type:

Primary and qualitative, research was conducted by the researcher.

Initial Approach:

A prototype was made by the researcher for the two best solutions found in the first research question. The prototypes include creating a session and connecting two clients, simple interaction through triggers and collision boxes, and two variables which need to be synced. The prototypes were made with the Peer-to-Peer model and included the use of a dedicated server. The researcher compared the ease of use, intuitivity and availability of the resources for both solutions.

Expected Results:

The research resulted in the best networking solution for Unity. The solution was used for the rest of the project.

Test plan:

Three tests were designed to check the capabilities of each system.

- Both systems must demonstrate how clients are being connected, disconnected and handled throughout the game.
- Both systems must demonstrate how player movement is handled.
- Both systems must demonstrate how a simple interaction, such as a button press within a trigger, can have an effect synchronized over the network. This could be done through RPCs or synchronized variables.

Development:

A Unity project was made for each solution. This was done because both Netcode and Fishnet use similar namespaces. Having both systems active in one project will cause the engine to send errors regarding namespace declaration.

The researcher started with the Netcode prototype. A network manager was added to the scene and a player prefab was created. A scene with a network manager was created. A player with a stationary camera was added to the scene. The researcher implemented basic movement using keyboard buttons input. After the movement was implemented and functional, a simple trigger-based interaction was added. The player would collide with a trigger object, press a keyboard button and update a UI element on the canvas.

The Fishnet prototype was made by exporting the Netcode scene and importing it into a different Unity project. All the code was edited to include Fishnet-related networking code and the scene was edited to include the Fishnet network manager.

After both prototypes were created and tested, a SWOT analysis was created for each solution (Table 1 and Table 2).

Strengths	**Weaknesses**
• Easy implementation of the network manager and easy setup of spawnable prefabs • Reference type variables can't be networked in the script, which makes the code safer. This is done in order to prevent developers from working with null variables.	• Network variables are not updated instantaneously, a tick is needed for the variable to be updated. The delay was noticeable when trying to update the UI element. The text on the screen would not update properly because the code did not take into account the small delay. • Object management is very limited. GameObject enabling and disabling is not fully supported and any nested network objects must be individually spawned from scripts.
Opportunities	**Threats**
• Unity has different products which can be used alongside Netcode such as	• Documentation can be scarce for VR integration

could hosting services, authentication and easy lobby creation	• Object spawning is available only on the server, which can be a problem for spawning spatial anchors.

Table 1- SWOT Analysis for Netcode

Strengths	**Weaknesses**
• System was made using Mirror's API, meaning that tutorials for Mirror can be applied to Fishnet • Documentation is well written	• Mirror API can be confusing with the use of TargetRPCs, making this system more confusing • Movement is not synchronized smoothly on the server side. This aspect would be an issue if a spectator view is needed.
Opportunities	**Threats**
• More client authoritative functionality is available which makes the integration of the spatial anchors easier	• More systems must be done manually, rather than using templates or additional made systems (E.g., lobby system, setting up cloud hosting)

Table 2 - SWOT Analysis for Fishnet

Conclusions:

Based on the two SWOT analyses and the preference of the team, Netcode was the chosen networking tool. Netcode is the networking solution provided by Unity and offers a lot of additional tools and services which can be useful for further development of the project.

Network object and behaviour handling

Question:

3. How do objects get enabled, disabled, spawned and despawned over the network during and after the player connection?

Source and type:

Primary, secondary and qualitative, research was conducted by the researcher

Initial Approach:

The researcher implemented several features in single-player and experimented with synchronization over the network. While networking the existing gameplay, the researcher looked through the Unity manual and experimented with different locations for scripts and GameObjects.

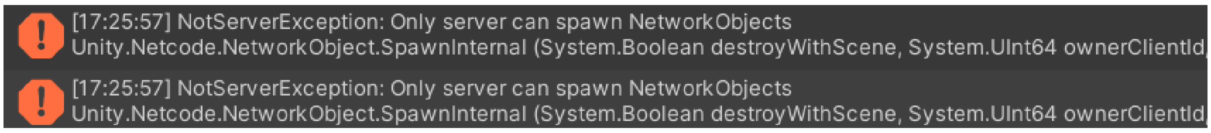
Expected Results:

The research concluded with information about the different ways Netcode handles GameObjects and authority. Additionally, the best way to spawn, despawn and synchronize over the network is a result of the question.

Theory and Research:

Netcode handles game objects over the network with the use of the Network Object component. This script must be attached to any game object that will be spawned, despawned or modified in any way over the network. The Network Object script must also be present if a game object includes a network behaviour. A network behaviour is a custom MoneBehaviour. It includes important functions like The OnNetworkSpawn and OnNetworkDespawn events (*NetworkObject | Unity Multiplayer Networking*, n.d.).

Netcode can handle two types of networked objects with attached Network Behaviours: the ones placed in the scene and the ones that are spawned at runtime. Objects can only be spawned at runtime through the server. If a client tries to spawn an object Unity would throw an error (Figure 4).



```
[17:25:57] NotServerException: Only server can spawn NetworkObjects
Unity.Netcode.NetworkObject.SpawnInternal (System.Boolean destroyWithScene, System.UInt64 ownerClientId,
[17:25:57] NotServerException: Only server can spawn NetworkObjects
Unity.Netcode.NetworkObject.SpawnInternal (System.Boolean destroyWithScene, System.UInt64 ownerClientId,
```

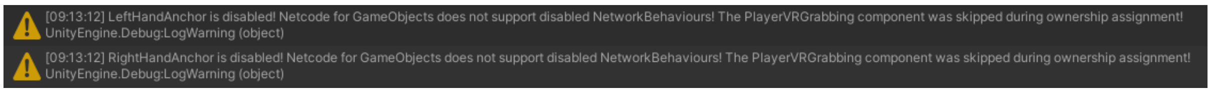
Figure 4 - Client Spawning Over the Network

Player Camera and the Oculus Headset:

During the creation of the player, the team discovered that the OVR Camera Rig – the camera used when working with an Oculus Headset – must be turned off by default. If the rig is not turned off when two clients spawn, there would be two active OVR Managers in the scene. Through testing, the team found out this would cause both cameras to freeze. This behaviour would also prevent the headset from being tracked at all.

Problems Encountered:

During the networking of single-player mechanics and gameplay, the researcher encountered several problems. The first problem relates to the Object Grabbing mechanic. Initially, in the single-player mode, the grabbing script was attached to each of the two controllers. This setup proved to be a problem due to the setup of the VR camera. Because the rig must be disabled by default, the grabbing scripts would be disabled as well. Netcode for GameObjects does not seem to support disabled NetworkBehaviour scripts (Figure 5).



```
[09:13:12] LeftHandAnchor is disabled! Netcode for GameObjects does not support disabled NetworkBehaviours! The PlayerVRGrabbing component was skipped during ownership assignment!
UnityEngine.Debug:LogWarning (object)
[09:13:12] RightHandAnchor is disabled! Netcode for GameObjects does not support disabled NetworkBehaviours! The PlayerVRGrabbing component was skipped during ownership assignment!
UnityEngine.Debug:LogWarning (object)
```

Figure 5 - Spawning Disabled NetworkBehaviours

Testing Potential Solutions:

The first solution was to spawn or attach a Network Behaviour after the GameObject is spawned. This approach did not work because in order to spawn anything over the network, a Network Object component must be present. Additionally, when the object would be spawned, parenting is an unreliable process. A developer should use the TrySetParent instead of the SetParent method, as this process requires certain rules (*NetworkObject Parenting | Unity Multiplayer Networking*, n.d.).

Final Solution:

The final included creating separate game objects for the grabbing mechanic. These objects will always be enabled within the player prefab. A VR collision script was added to each controller. This script will call functions in the grabbing scripts so that collision will be properly detected.

Conclusions:

Based on the documentation available and the tests performed, Network behaviours must always be attached to an active GameObject. The network behaviour can be disabled after the object has spawned. Additionally, parenting is an unreliable process, which can be replaced if needed with a combination of custom code and ownership change.

Player VR Functionality

Question:

4. How will player VR functionality such as movement and grabbing be networked?

Source:

Primary, secondary and qualitative

Initial Approach:

Desk research on VR movement for multiplayer games was conducted. The researcher implemented prototypes which implement both server and client authoritative functionality and tested potential latency for either case.

Expected Results:

The result will be networked player functionality in mixed reality. In addition, the question will result in determining which features must be client authoritative and what functionality must be server authoritative.

Development:

Initially, the player movement was based on the headset position and rotation. The player prefab contains a Client Network Transform in order to give the client authority to change the position. Netcode does not include this script in the package, as the networking engine is made to be server authoritative by default. The choice to make the player movement client authoritative was made due to the fact that the game uses a VR headset. Using server authoritative movement would have created a significant delay between the player's physical movement and the position update of the virtual object.

The delay was noticed mostly with the use of the grabbing mechanic. In single-player mode, the grabbing mechanic parented and unparented the object grabbed. This method, however, was not reliable in a multiplayer setup as described in the third research question.

The next solution consisted of moving the grabbed object while the trigger of the controller was pressed. While the object would move as long as the trigger button was pressed, the object would move back to the original position as soon as the player stopped pressing. The grabbable object had the Network Transform component attached, meaning that any movement was updated by the server. However, even if the Network Transform Component is replaced by the Client Network

Transform, the problem persists. The cause of this issue is the object authority. Unless mentioned when an object spawns over the network, the server will have authority and owns the object. This means that only the server can change the position of the object. It is important to mention that this problem does not occur for the player GameObject, as when a player spawns it will be given authority over itself.

The next solution was the usage of a combination of client and server RPCs. When the player grabs an object, the position of the controllers would be sent through a serverRPC to the server. The server would then change the position of the grabbed object. This method would cause a similar amount of lag for all clients when grabbing objects.

Another solution involved changing the authority of the object right after it is grabbed. When the player presses the trigger button to grab an object a serverRPC would be sent to change the authority of the object. The client will then be able to move the object without any issues. When the player releases the button, a serverRPC is called to change the authority back to the server. This solution causes lag only for the server and the non-local players. This means that the player who grabbed the object will see the position update instantaneously. Based on testing with the team, it was decided that this solution provided a better user experience and was ultimately chosen for the final product.

The second part included the use of shooting and AI. The enemies and the projectile system run completely on the server. The position from which the projectiles spawn is sent through a serverRPC and there is no need for client authoritative functionality.

Conclusions:

With the research and testing conducted, the first part of the game of the final product became almost completely client authoritative. Due to the use of VR controllers, which are not available to the server, the client must have the authority to interact and change the position of the objects. The second part remained server authoritative as the was no need to keep track of any controller information.

Humanoid Collision

Question:

5. How will humanoid collision information be available for all the players in a session?

Source and type:

Primary, secondary and qualitative

Initial Approach:

Look into Oculus hands and arms detection and attach collision boxes. The researcher looked into the motion-tracking capabilities of the Oculus headset and mesh colliders. Then, a prototype was made to test the possibility of humanoid collision. Finally, the collision information was networked and tested along with the latency.

Expected Results:

The result will be networked collision information.

Development:

The researcher looked into how avatars move their arms in the VR lobby of the Oculus system. While Oculus includes hand tracking, it does not support accurate arm tracking. The arms move with the use of a VR body. A VR body can be created through the use of Inverse Kinematics. A humanoid rig is needed. A constraint for the hands and the head is added to the rig. Through a script, the hand bone is moved to the positions of the controllers. Through the constraints, the arms would move with the hands (*How to Make a Body in VR - PART 1 - YouTube*, n.d.). Although the movement is not completely accurate (E.g., the rotation of the elbows could be inaccurate), it is realistic enough to attach UI panels to different bones.

Testing:

The initial solution was to use mesh colliders to simulate proper humanoid collision (Figure 6). The mesh collider needed to be set to a trigger. In order to make the collider a trigger, it must be convex. When the mesh is set to convex, the collider changes shapes and does not hold the shape of the mesh (Figure 7). This makes the collider unreliable. Furthermore, the default collider does not update in real-time. This means that when the player is moving the arms, the collider does not update with the correct shape.

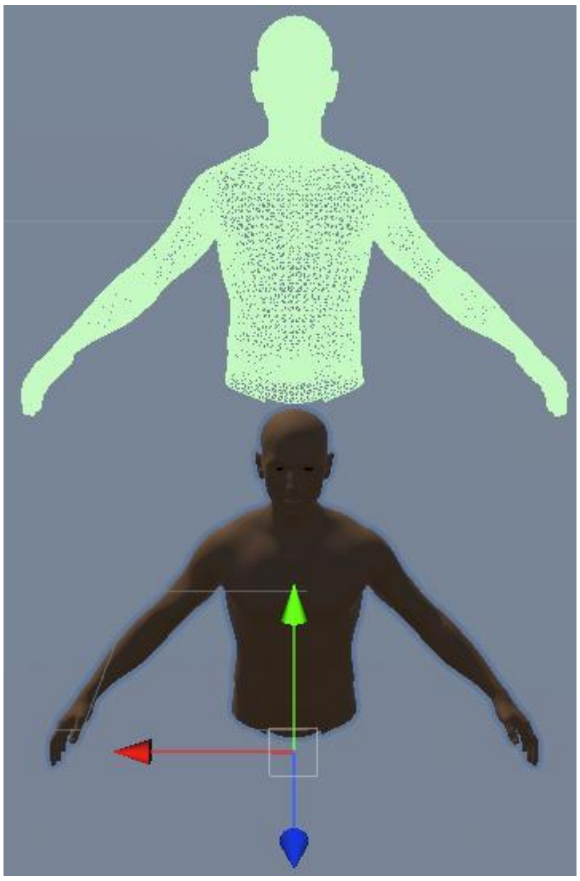


Figure 6 - Default Mesh Collider

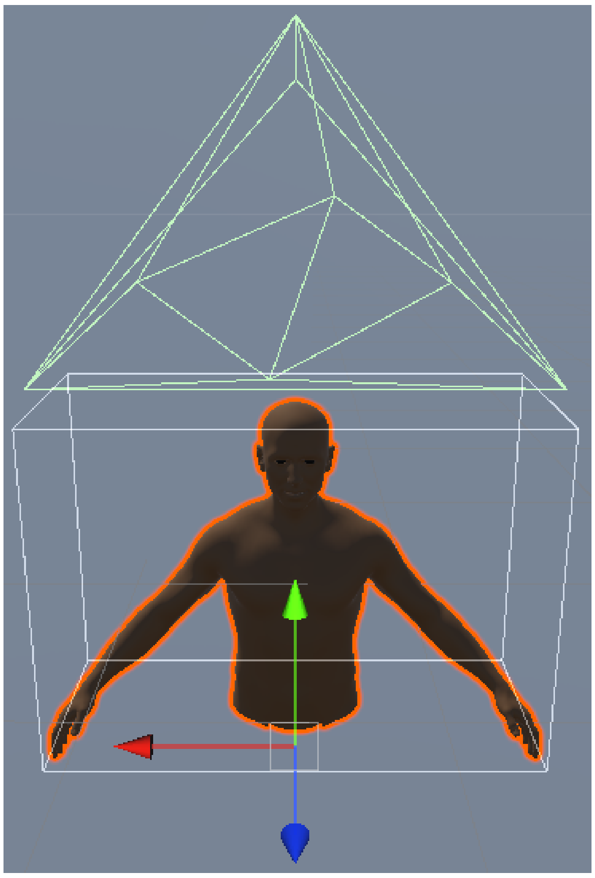


Figure 7 - Convex Mesh Collider

Conclusions:

Due to the behaviour of the mesh collider, humanoid collision is not feasible. Due to the constraints of the mesh collider and the behaviour observed for the used 3D model, true humanoid collision was not implemented.

A box collider was used to simulate the hitbox of the upper body in the final product. This solution was more suitable for the project as the final model did not include the head and arms. Another possible solution for implementing humanoid collision would be to add different capsule or box colliders to several bones, creating a similar system and setup to the Unreal Engine physics asset.

Networked game mode and player states

Question:

6. How will different game modes (during runtime - parts 1,2 and before the session starts - in the lobby) be integrated over the network?

Source and type:

Primary and Qualitative

Initial Approach:

The researcher looked into how events are transmitted over the network. The researcher conducted testing in order to determine the proper software architecture so that different game modes can be encapsulated.

Expected Results:

The expected result was a framework which included three parts with different gameplay for one session and two game modes for two different game experiences.

Development:

This research question was divided into two parts: implementing the gameplay transition between the first and second part of the game and implementing the player vs player mechanics and the possibility to choose between the different game modes. It is worth mentioning that the second part of the research question was concluded later in the project, as the CoOp game mode was implemented and tested first.

The first task of the researcher and the team was to make the game playable from start to finish in CoOp mode. At that point, most features have been implemented. Parts one and two could be set up to run individually due to the use of different spawn points which could be enabled and disabled and the new Input System from Unity which allows different methods to be subscribed and unsubscribed from the event system. By enabling and disabling different spawn points and not subscribing to different input events, both parts could be tested separately.

In the first part of the CoOp mode, the scene included a Player Creature Handler. This manager kept track of how many creatures the player collected. When every player collects three different creatures, the second part must start. For this reason, the first solution was to include an event for the beginning of part 2 in the Player Creature Handler. The scripts which spawned objects for part 2 (E.g., the enemies and the charging stations), the shooting script and the friendly creature scripts would subscribe to the event declared in the Player Creature Manager. The manager included a public static instance so that every script could access the event.

Changing the gameplay from the first part to the second part required a combination of server and client actions. Therefore, the Player Creature Handler included two events for changing part 1 to part 2, one for the server side and one for the client side. All the scripts which spawned objects or set up the friendly creatures had methods subscribed to the server event as spawning, despawning (as mentioned in the third research question) and the friendly creature's logic runs on the server. The grabbing and shooting scripts have methods subscribed to the client event as input must be handled on the local client application. The Player Creature Manager contained public methods which first invoke the server event and then, through a client RPC, invoke the client event. The reason why this manager in particular held the events is because the condition to advance to the second part is determined by the script.

The setup described above allowed the team to test the first and second parts in one session. Due to this, a second set of server and client events was added to the Player Creature Handler to invoke the end of the game. The setup for invoking the ending is identical to the one invoking part 2. With the addition of the ending in the Player Creature Manager, the team could play the game entirely.

With the CoOp game mode playable from start to finish, the researcher focused on implementing the Player vs Player mechanic. Immediately, the event setup within the Player Creature Manager created some problems. The script had to include a variable to decide which game mode is played – the CoOp or PvP (Player vs Player). The addition of another game mode added different functionality to a script designed to keep track of the creatures collected by the players. This contradicts the Single Responsibility rule within Software Architecture. This rule states that every script should have one function. For this reason, a Player State Manager script was created and the events were moved to this manager. The scripts which subscribed different methods to the events were modified to include the static public instance of the Player State Manager.

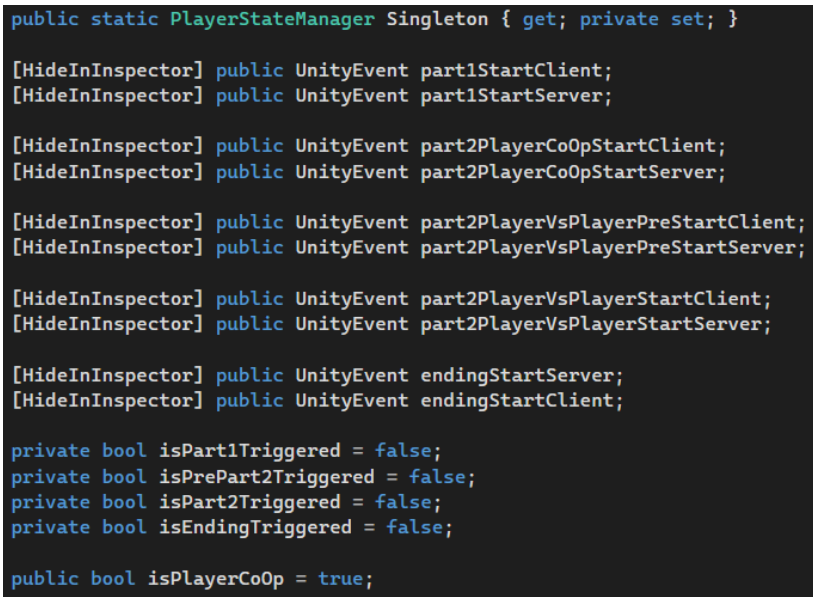
For invoking the PvP mode, the Player State Manager would contain a different set of events. Some functionality had to be subscribed to both the CoOp part 2 start event and the PvP part 2 start event (E.g., changing the behaviour state of the creature on the server, disabling the input for grabbing on the client). With the addition of this script, the Player Creature Handler became separate which improved the architecture of the code. The Player State Manager also allows the developers to easily invoke any part of the game – part 1,2 or the ending, without needing to comment code or enable and disable GameObjects.

The Player vs Player game mode must include a transition part between the first and the second part of the game. This transition exists so that the players could form teams. For this reason, in addition to the PvP part 2 start events, a set of Pre PvP part 2 start events were created for spawning the team assigning trigger boxes.

It is important to mention that during internal testing of the full game, the team found out that the events could be called multiple times when they should only be called once. For this reason, variables to keep track of what events were invoked were added to the Player State Manager.

Conclusions:

The final product includes a Player State Manager which handles the events. This manager allows for easy testing of separate parts of the game. When the server changes the game mode, the manager will invoke the correct event. The final setup of the Player State Manager includes four sets of events, with one set including one server event and one client event (Figure 8). The setup of the script allows future game modes or game parts (E.g., tutorial parts) to be added easily.



```
public static PlayerStateManager Singleton { get; private set; }

[HideInInspector] public UnityEvent part1StartClient;
[HideInInspector] public UnityEvent part1StartServer;

[HideInInspector] public UnityEvent part2PlayerCoOpStartClient;
[HideInInspector] public UnityEvent part2PlayerCoOpStartServer;

[HideInInspector] public UnityEvent part2PlayerVsPlayerPreStartClient;
[HideInInspector] public UnityEvent part2PlayerVsPlayerPreStartServer;

[HideInInspector] public UnityEvent part2PlayerVsPlayerStartClient;
[HideInInspector] public UnityEvent part2PlayerVsPlayerStartServer;

[HideInInspector] public UnityEvent endingStartServer;
[HideInInspector] public UnityEvent endingStartClient;

private bool isPart1Triggered = false;
private bool isPrePart2Triggered = false;
private bool isPart2Triggered = false;
private bool isEndingTriggered = false;

public bool isPlayerCoOp = true;
```

Figure 8 - Player State Manager Final Setup

Lobby interface

Question:

7. How will the interface for connecting players and starting a session look like?

Source and type:

Primary and qualitative

Initial Approach:

Research how a game session is started. The developer implemented a server and a client lobby in which the game session can be set up before the

Expected Results:

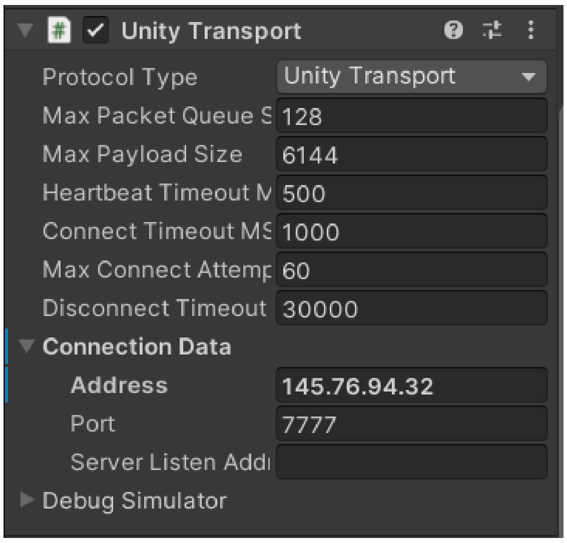
The result will be the start game and reset system used by arcade managers.

Development:

This research question was divided into two systems: a client lobby and a server lobby. The client lobby was implemented when the map calibration system was implemented. The map calibration system was implemented by Kristyna Pavlatova. With the system, the player must have some time to calibrate the map. Therefore, the game should not start until the map calibration process is completed by every player. To accomplish this, in the Player State Manager, events for starting part 1 were added. This ensured that the input for the grabbing mechanic was disabled and did not interfere with the input needed for the calibration. Additionally, this setup prevented the grabbable objects and the friendly creatures from spawning as soon as the server starts. Thus, the player can focus on the map calibration rather than the environment.

The server lobby initially consisted of one button for starting the CoOp mode. Once the Player vs Player mechanics were added, another button was added for the PvP game mode. The two buttons were then changed to a dropdown menu. A server UI script was made to access the Player State Manager to start the first part of the game.

The LAN multiplayer system in Netcode is created by imputing the IP address of the machine which is running the server. The IP address must be the same for both the server and the client build and must be written in the Unity transport component (Figure 9). The issue with this setup is that the IP Address tends to change very frequently on the same machine. This means that the game would have to be rebuilt every time the IP Address changes or the machine running the server changes.


Unity Transport

Protocol Type	Unity Transport
Max Packet Queue S	128
Max Payload Size	6144
Heartbeat Timeout M	500
Connect Timeout MS	1000
Max Connect Attemp	60
Disconnect Timeout	30000
Connection Data	
Address	145.76.94.32
Port	7777
Server Listen Addr	
Debug Simulator	

Figure 9 - Unity Transport IP Address

One potential fix is to include files which store the IP Address on both the server and the client build. This solution, however, is not user-friendly as it requires the arcade managers to change game files on the computer and on the Oculus headset.

Another solution would consist of using an Input field. The clients would get the IP address from the arcade manager when the application is launched and write it in the input field (Figure 10). This solution though, is not user-friendly, as typing in VS could be quite challenging.

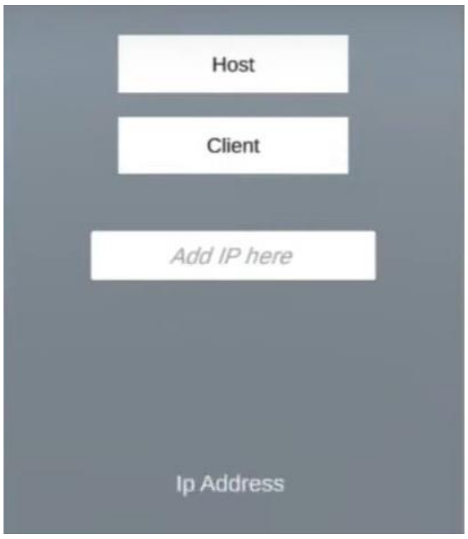


Figure 10 - Connection UI Example

The best solution is to use a Network Discovery script. While other networking solutions (E.g., Mirror, Photon) include this functionality, Netcode does not. However, similar to the Client Network Transform, there is code for an Example Network Discovery made for Netcode (*Multiplayer-Community-Contributions/Com.Community.Netcode.Extensions/Runtime/NetworkDiscovery at Main · Unity-Technologies/Multiplayer-Community-Contributions · GitHub*, n.d.). The example script must be attached to the Network Manager and would find the right IP Address (Figure 11).

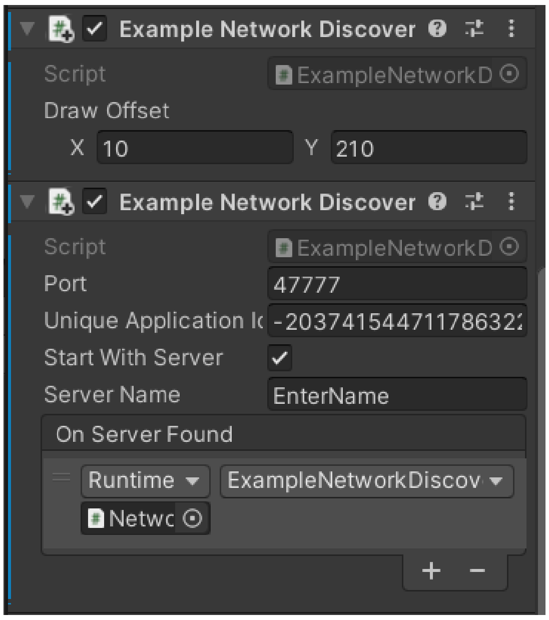

Example Network Discover
Script ExampleNetworkD
Draw Offset
X 10 Y 210
Example Network Discover
Script ExampleNetworkD
Port 47777
Unique Application Ic -20374154471178632
Start With Server ✓
Server Name EnterName
On Server Found
Runtime ExampleNetworkDiscov
Netwc

Figure 11 - Example Network Discovery

Conclusions:

The final product includes a Custom Network Discover, created by extending the Example Network Discovery. This solution made it possible to run both the server and the client without changing the IP Address.

Final Product and Networking Framework

The final product consists of a Unity package (Appendix B). The package includes a square map which corresponds to a room at the XR Lab. The scene can be imported into Unity or played with the server and client builds (Appendix C). A video (Appendix D) was made to showcase the entire gameplay.

The final gameplay and networking framework can be divided into four parts:

- The lobby part includes:
    - A client lobby where the players can calibrate the map without any gameplay objects or functionality.
    - A server lobby where the game mode can be chosen and the number of players is visible.
    - A custom Network Discovery Script extended from the Example Network Discovery, used for easy connection between the client and the server over LAN.
- The first part of the game includes:
    - Client authoritative collision detection and player and object movement.
    - Server authoritative friendly creature logic and player creature handler.

- The second part of the game includes:
  - Server authoritative enemy logic and projectile system.
  - Server authoritative player life system and collision detection.
  - Server authoritative PvP team assignment.
- The ending part includes:
  - Server authoritative scoring system for both CoOp and PvP.
- Miscellaneous features include:
  - Restart system which resets the game when the ending was reached.
  - Server authoritative Player State Manager to change the part of the game.
  - Server authoritative sound system able to play sounds for all players or specific players.
  - Inverse kinematics which creates a VR body, used for displaying the Player Creature UI Panel in part 1 and the player lives in part 2.

Recommendations

To improve the game and the multiplayer framework, several recommendations could be taken into consideration.

- Change the stream shooting mechanic

Particles are not being synchronized over the network. This is a limitation of the networking system. The system plays the particles systems play on both the server and the client and the collision of the server side is considered for any checks. However, if for any reason, the client's visuals are not corresponding with the server's visuals, the collision will be unreliable. Changing the system to use a trigger or collision box which gets enabled and disabled can improve the functionality of the mechanic. Additionally, as a general rule, particle systems should be used sparingly and mostly for visual feedback or environmental design. In the final product, the stream shooting mechanic was replaced by projectiles.

- A main menu scene should be added

Sometimes, the client build freezes when the application is launched. This could be because the connection to the server was not successful. The current solution is restarting the client application, which seems to fix the issue. To prevent the build from freezing, a main menu on the client side should be implemented so that the player could retry to connect if the first attempt was not successful.

- Using a network variable to hold the client ID

From the server, currently, it is not possible to determine which player is which. Determining which client got hit by projectiles or client died in the PvP mode is important. Currently, the system uses a combination of server and client RPCs to determine which player logic should be called. An example of this process is used in the Sound Manager, as some sounds must be played on one specific client. Having a variable which holds the client ID over the network eliminates the need for the client and server RPCs structure.

Conclusion

The team achieved a proof of concept. The end product includes different game modes and presents opportunities for future development. Netcode proved to be a powerful tool, though it does have its limitations. Working with VR always presents a challenge for development, especially when working with multiplayer functionality.

References

Add-ons - Fish-Net: Networking Evolved. (n.d.). Retrieved from https://fish-networking.gitbook.io/docs/manual/general/add-ons

Arena Amsterdam (H2o Campus) - HADO Benelux. (n.d.). Retrieved from https://www.hado-sports.com/arena-amsterdam-h2o-campus

Baker, H. (2021, February 9). *SideQuest for Oculus Quest: Everything You Need To Know*. https://uploadvr.com/everything-you-need-to-know-sidequest/

Exclusive Experiences |ENG| VRGH Arena|Virtual Reality Arena in Amsterdam Westpoort. (n.d.). Retrieved from https://vrgh.nl/en/vr-experiences/#games

General - Mirror. (n.d.). Retrieved from https://mirror-networking.gitbook.io/docs/manual/general

Heaney, D. (2023, February 6). *Quest 2 Finally Supports Automatic Shared-Space Local Multiplayer*. https://www.uploadvr.com/quest-2-finally-supports-colocation-shared-spatial-anchors/?fbclid=IwAR1MZSaU-ywBPFeSKu2ufn12vWLp5VH27o_CJmpCoLocWk6RN5b8tEnckig

How to make a Body in VR - PART 1 - YouTube. (n.d.). Retrieved from https://www.youtube.com/watch?v=tBYl-aSxUeo

Introducing Meta Reality: A Look at the Technologies Necessary to Convincingly Blend the Virtual and Physical Worlds | Meta Store. (2022, December 19). Meta Quest Blog. https://www.meta.com/blog/quest/mixed-reality-definition-passthrough-scene-understanding-spatial-anchors/

Introduction | Photon Engine. (n.d.). Retrieved from https://doc.photonengine.com/pun/current/getting-started/pun-intro

Introduction - Fish-Net: Networking Evolved. (n.d.). Retrieved from https://fish-networking.gitbook.io/docs/

Licenses | Photon Engine. (n.d.). Retrieved from https://doc.photonengine.com/server/current/operations/licenses#

Mirror | Network | Unity Asset Store. (n.d.). Retrieved from https://assetstore.unity.com/packages/tools/network/mirror-129321

multiplayer-community-contributions/com.community.netcode.extensions/Runtime/NetworkDiscovery at main · Unity-Technologies/multiplayer-community-contributions · GitHub. (n.d.). Retrieved from https://github.com/Unity-Technologies/multiplayer-community-contributions/tree/main/com.community.netcode.extensions/Runtime/NetworkDiscovery

NetworkObject | Unity Multiplayer Networking. (n.d.). Retrieved from https://docs-multiplayer.unity3d.com/netcode/current/basics/networkobject/

NetworkObject Parenting | Unity Multiplayer Networking. (n.d.). Retrieved from https://docs-multiplayer.unity3d.com/netcode/current/advanced-topics/networkobject-parenting/

Onze Games - VR Amsterdam. (n.d.). Retrieved from https://vr-amsterdam.nl/onze-games/

Rokhsaritalemi, S., Sadeghi-Niaraki, A., & Choi, S. M. (2020). A Review on Mixed Reality: Current Trends, Challenges and Prospects. *Applied Sciences 2020, Vol. 10, Page 636*, *10*(2), 636. https://doi.org/10.3390/APP10020636

RPCs and RaiseEvent | Photon Engine. (n.d.). Retrieved from https://doc.photonengine.com/pun/current/gameplay/rpcsandraiseevent#targets__buffering_and_order

Spatial Ops — Resolution Games. (n.d.-a). Retrieved from https://www.resolutiongames.com/spatialops

Unity - Manual: Multiplayer and Networking. (n.d.). Retrieved from https://docs.unity3d.com/Manual/UNet.html

Unity vs Unreal for VR/AR! Which Engine Should You Choose? (n.d.). Retrieved from https://xrbootcamp.com/unity-vs-unreal-engine-for-xr-development

Appendices

Appendix A – List of Feature From TDD

Gameplay Mechanics

Part 1 - Befriending the creatures

- 3 types of creatures that players have to befriend (More creatures can be added in future development stages)

1. (Fire) One runes away → player must bring food
2. (Water) One must get to a place → player must help by bringing planks of wood to make bridges
3. (Earth) One is waiting for food → players must bring water to help the plant grow so the creature can eat

- Player can interact with individual creatures after befriending them (example: petting)
- Player creature control check
    - Each player must collect 1 creature of every type in order to progress to another part of the game
    - Each player has visual feedback on what creature they collected
- Core Mechanic: Grabbing mechanic → player interacts with the world by grabbing objects, grabbing is done with the controllers. Objects are set around the map, the player needs to figure out what to do with them. Grabbing is done with the trigger buttons

Part 1.5 - Learning creature magic

- Charging stations appear, this is where the players get the magic ability
    - Players can choose which element to choose by going to a specific charging station
    - Players have one ability at a time
- Core Mechanic: Shooting mechanic → projectile for x ability and stream for y ability.
    - Fire as a projectile
    - Water as a stream
    - Earth as a projectile
- Shield Mechanic → Player crosses the controllers in order to charge a shield where they are at the moment. Each shield is unique for each player,

Part 2 - Boss fight
Boss:

- Boss has 3 stages correlating to the magic abilities
    - Stage 1: Fire
    - Stage 2: Water
    - Stage 3: Earth
    - The stage changes with the boss' health
    - The boss will have some weak points or hit points where the players need to figure out where they need to aim at.
- The boss can disappear and spawn on another part of the map (predetermined spawn points)
    - The boss can spawn minions when it disappears

Player:

- Players have their individual shields to absorb boss' projectiles by toggling the grip button of the left controller
- Player's projectiles depend on the currently selected magic ability
- When hit, the player loses part of their current magic ability. If hit multiple times they lose the ability and can't shoot with it anymore.
- Player acquires a type of shooting ability by going to a charging station with friendly creatures with a specific ability type.
- The player gets score based on the damage done to the boss

Networking Mechanics

LAN connection

- Server is a computer (Windows app)
- Client is an Oculus Set (Android app)
- Lobby system for the server computer to start the session
- Lobby room for the players

General

- Player movement based on the headset position
- Player collision for upper body
- Haptic feedback(when the player gets hit-vest, when the player pets a creature-controller)

Appendix B – Professional Product - Unity Square Scene Package:
https://drive.google.com/file/d/131Ui__QHN2XVbkPAehAOBqbwWABDsf8f/view?usp=sharing

Appendix C – Professional Product – Final Game Builds:
https://drive.google.com/file/d/18k5XlpWCs5s_GthirSFZja_c6ZGMWBJA/view?usp=sharing

Appendix D – Professional Product - Playtest Recording:
https://drive.google.com/file/d/1cm5M7XYofOoO4noq2SMIO_4ec3OnHQBE/view?usp=sharing

Or

https://youtu.be/3Rs4zOxSl44

Appendix E – Professional Product - Unity Project Repository: https://github.com/Ana-MariaIlea/GraduationProject-VRAmsterdam