Tampere University

Tapani Landén

DEVELOPING A CROSS-PLATFORM MULTIPLAYER GAME FOR VR AND TABLET

Faculty of Information Technology and Communication Sciences
M. Sc. Thesis
May 2023

ABSTRACT

Tapani Landén: Developing a Cross-Platform Multiplayer Game for VR and Tablet
M.Sc. Thesis
Tampere University
Master's Degree Programme in Software, Web and Cloud
May 2023

Multiplayer gaming is a big market and there is a lot of supply for different platforms, but it seems that there are not that many games that could be played together with different platforms. In addition to that, there seems to be a lack of supply for systems to test the games suitability for several platforms.

In this thesis, I tackle this problem by introducing a novel way to assess these properties of a game by focusing on the accessibility of the sensory information the game's objects contains by trying to access them on different platforms. In my experiment, I found out that this method is useful in finding asymmetries in accessibility of sensory information between the devices, though I also found out that not all of it is necessarily bad for the game.

I also focus on the development process of a cross-platform multiplayer game by introducing a game called Forest Friends – a project that I have been developing as a part of a team. It is a children's game, that can be played with both VR and tablet.

Key words and terms: Virtual Reality, Virtual Environment, Game, Forest Friends, Sensory Information, Mobile Game, Tablet Game, Unity, Mirror, Asymmetric VR, Game Design, Game Development, Game Testing.

Contents

Abbrevations

AR	Augmented Reality
CVE	Collaborative Virtual Environment
GO	GameObject
HMD	Head-Mounted Display
I/O	Input/Output
MR	Mixed Reality
UI	User Interface
VE	Virtual Environment
VR	Virtual Reality

1 Introduction

In January 2023, there are over 70 000 games in Steam (2023). At the same platform, the number of VR games is over 4500, but the number of games in the genre Asymmetric VR - that means the games that can be played simultaneously with VR and some other device - is only 95. It seems that there is a potential market for a larger supply of Asymmetric VR than there is now.

The size of the market is not the only reason to be interested in developing cross-platform games that combines VR and mobile. In my case, I am a part of a two-person group that is developing a VR game to be played in a Tampere University Hospital's new child and youth compartment's "Aistihuone" (translates to "sensory room"). Our target group is the child and the youth patients of the hospital. Since some of the potential players are not able to move out from the ward there is a need for a mobile version of the game.

There are also other things that prevents the usage of head mounted device (HMD) for part of the target group, like relatively high age suggestions for using VR devices. For example, Meta (2023) states that only over 13 years old are allowed to use their VR systems and that they prevent account creation from children under that age. There are also some medical conditions that prevents the usage of HMD, so there are many reasons for multi-platform approach. Adding the possibility to join the game with a mobile device increases the accessibility of the game and gives the possibility to be a part of the gaming experience for a bigger share of potential players.

Even though there are game engines, like Unity and Unreal Engine, that makes the development and deployment of cross-platform games easy and straightforward, there are not that many solutions to test how the game itself fits for the cross-platform development: especially when the platforms are very different. I am going to tackle this problem by considering the game as a Virtual Environment that is built with objects that consists of sensory information, like shapes and sounds. By extracting these sensory data points from each object of the game, and by testing and evaluating their accessibility with every selected platform, I obtain measurable data on the differences in accessing sensory information between selected platforms, revealing any asymmetries in their capabilities. My presumption is that the smaller the asymmetry – that means that there are similar possibilities to access the sensory information of the game with different devices – the better fitted for cross-play the game is.

The purpose of this thesis is to explain the process of developing a multiplayer multiplatform game and to create a system to measure the asymmetry of accessing the sensory information between different devices. To achieve this second goal, at first, I must find and classify all the sensory data points of each Game Object; then I must test that those data points are accessible with selected devices; and lastly, the results of the testing with each device must be compared. This information can then be used to improve the game's fit for several platforms. This target can be formatted as a research question: Can a game's accessibility of sensory information with different devices be measured and compared, and can that information then be used to improve the game's quality?

Instead of an actual theory part, I will start this thesis by explaining some main concepts of the game development tools we used and after that, I will present the game that we've developed. This is because the theory part uses the terminology that is presented in these first three main chapters of this work. That said, the second chapter will be all about the key concepts of Unity. In the chapter three, I will continue that topic by explain the key concepts of a Unity's framework called Mirror, that our team selected to use for handling the networking in our game. The game that I am referring to is called Forest Friends, and the chapter four is all about the development of that - especially about the multiplayer version of the game. After that I am going to start the actual theory part and go through the concepts concerning Virtual Environments and Virtual Reality in chapter five, and in that same chapter, I also introduce a way to consider Unity's Scene as a Virtual Environment. In that same chapter I answer to the question of how to classify a Game Object's sensory information. In the chapter six I use this classification to the Game Objects of Forest Friends, and test it with both tablet and VR, and evaluate the results. I end this thesis to the seventh chapter, where I summarize everything and make some suggestions for further research.

2 Scenes and GameObjects: Introduction to Unity

Unity is a game engine with a graphical editor that is used to create games, apps, and experiences (Unity, 2023). It is a 3D game engine that suits well for multiplatform development because it supports all the biggest platforms. They also promote the “create once, deploy anywhere” idea, that means that you can develop a game once and then build it for any platform that they support. In this thesis I’m using some terminology that is taken from Unity, and next I’m presenting the key concepts that Babtiste et al. picked to their book (2022).

2.1 GameObject

GameObject (GO) is the most important concept in Unity (Unity, 2023). Everything that is inside your game’s environment - or Scene as it is called in Unity - and everything that you can experience in the game, like player characters, non-player characters, sounds, items, and special effects, are contained in GOs. GOs can encase other GOs, and some GOs are just containers that does not have any other functionalities. GOs can have as many Components as is needed, but each GO must have a Transform component that gives the object it’s position, rotation, and scale in the Scene.

2.2 Component

Components are what gives functionality to the GO (Unity, 2023). In practice, self-made Components are C# classes, that developers can write to add functionality for the GO, but there is also a huge number of ready-made Components in Unity. There are different types of Components that makes different things, like renderers, that are used to give some appearance for the GO, or audio source, that you can use to add sounds for the GO.

2.3 Scene

In Unity, Scene means the place where the game happens (Unity, 2023). A game can consist of one or many Scenes, and each Scene consists of GOs. In games Scenes can be levels, but levels can also be made of several Scenes. There is no limit for the number of Scenes in Unity.

2.4 Asset

Every file in the game is called Asset in Unity (Babtiste et al. 2022). Assets can be things like 3D modellings or scripts or sounds, and concepts like Scenes and Prefabs are Assets. All the game project’s files should be under the Assets-folder.

2.5 Prefab

GOs can be transformed into Prefabs, that are templates of GO. Prefabs can be used in many scenes so that when the prefab is changed, the change will affect all the GOs that are created out of said prefab. So, each prefab can hold a bunch of items and values, and each instance of them can have some additional values and items (Babtiste et al. 2022). There are cases where GOs does not exist in the Scene at the beginning but are spawned during the gameplay and this can be done by using Prefabs.

2.6 Script

Script is a certain type of Component in Unity, that is written by some developer. Script becomes a Component when it is attached to a GO. Like other Components, Scripts add functionality to GOs, and with them the logic of the game is implemented to the game. The (primary) language of Unity's Scripts is C# (Babtiste et al. 2022).

2.7 Package

Package is a modular component that can be exported or imported (Babtiste et al. 2022). For instance, in Forest Friends, all the animal characters are from a Unity Package, that was bought from Unity's Asset store. The Package included 3D animal characters and their animations, so our task was just to add them to the Scene and control their animations with our own Scripts.

Other good example of Unity's Packages is Mirror, and the next chapter is all about it.

3 Concepts of Mirror

Mirror is a high-level networking library for Unity that can be downloaded from Unity's Asset store for free. It is based on Unity's own, now deprecated UNet multiplayer and networking solution, that was launched on 2015 (Unity, 2023) (Mirror, 2023). Here I have listed some of the Components that Mirror offers, that are crucial for understanding parts of this thesis.

3.1 Network Manager

Network Manager handles things related to managing the networking of multiplayer games (Mirror, 2023). This Component offers the possibility to choose what transport to use in networking - Mirror has five built-in transports, but there are also third-party options available. When a server, a client, or a host needs to be started, it must be done through the instance of this class, and because it handles this kind of major networking things, there can be only one instance of the Network Manager class.

All the networked GOs need to be registered before they can be used, and this can be done by adding those GOs to the "Registered Spawnable Prefabs" list of this Component. Network Manager Component can also be customized by inheriting the NetworkManager class and by overriding its functions.

3.2 Network Identity

This is the Component that all the networked GOs must have. This gives GO an identity, that networking system uses to keep track of that object. All the GOs with this Component needs to be spawned to the Scene, though if the networked GOs are already added to the Scene, Mirror handles this spawning automatically by deleting each of the networked Scene objects from the Scene and spawning them to the Scene before it is loaded (Mirror, 2023).

3.3 Network Discovery

Network Discovery is used to send messages between clients and servers to exchange information about connections. For example, in Forest Friends, the Network Discovery is used to find out if anyone is hosting the game already, and if a host can be found, then the client will connect to that.

3.4 Network Transform

Network Transform Component is used with GOs whose position and rotation needs to be synchronized over the network. For example, this kind of GOs can be objects like player characters or some items that are moved by a player.

In the next chapter I will explain how Unity and Mirror can be used in real life, as I go through the process of developing Forest Friends.

4 Case Forest Friends: A Multiplayer Multiplatform Game

Forest Friends is an asymmetric VR game - that means a game that can be played with both VR and some other device - that is designed to be played by children and youth in a hospital environment. The game can be played either with Oculus Quest 2 device or with a tablet that runs on Android OS.

The objective of the game is to feed the animals of the surrounding forest by throwing or carrying food for them. Each animal favors certain food, and each animal-food-combination gives a different number of points: more points from the foods that the animal likes the most, and less points from the foods that the animal likes the least. The scale is from five to one point, but only the most and least favorite foods are explicitly presented in the Tutorial Scene.

4.1 Introducing Forest Friends 1.0 – a VR Game for Children with Anxiety

Forest Friends, the game that my team has been developing, started as a project in Software Engineering Project course at Tampere University, and at first the goal was to make an activation game for children with anxiety. In our team, we had six members: Tapani Landén (me), Zhihao Tan, Clara Reichert, Nicolas Boucht, Hanna Suhonen, and Laura Launonen. Our main responsibilities were as follows: I was the project manager; Nicolas Boucht, Zhihao Tan, and Laura Launonen did most of the programming; Hanna Suhonen and Clara Reichert organized the testing event and did level designs with Laura Launonen. Our responsibilities were not strictly divided, and everyone participated to designing and to developing the game. Our customer was TAUCHI (Tampere Unit for Computer-Human Interaction) of Tampere University, and our contact persons from there were Rucha Tulaskar and Ilmari Jyskä. The course's supervisor was Timo Poranen.

My team came up with an idea of a game where the player would feed animals in a forest by throwing them different foods. Each animal would prefer different food over other foods and their preferences would be introduced in a tutorial scene. When an animal gets food, it will make a small jump and starts to eat it – whether it is something it likes the most or not. Points gained from the throw are shown as a number that appears for a short time in front of the animal.

Since the task was to make a game for children with anxiety, we tried to acknowledge and discard all the features that would make the game competitive or otherwise stressful. That is also the reason why we decided to use low-poly-graphics – we felt that too realistic creatures and surroundings can be distressing if not done properly. We chose to use minimalist soundscape and sound effects to support relaxing and peaceful effect of the game.

The game was developed with Unity for Meta's Oculus Quest 2 -device and it was ready in May 2022.

4.2 Towards Forest Friends 2.0 – a Multiplatform Multiplayer Children's Game

Our team was asked to continue with the development of the game at the beginning of the fall semester 2022. Me and Nicolas Boucht from the original team agreed to proceed with the game, and from TAUCHI we had Professor Markku Turunen as the head of the project and Ilmari Jyskä and Rucha Tulaskar as contact persons. Professor Kaija Puura from Tays participated the project by assessing the game from the psychiatrist's point of view.

Some further ideas for development were introduced: Because some of the potential players of the game were in a ward, our team was requested to make a version of the game that could be played with a mobile device. Multiplayer version of the game was also requested. The game was meant to be launched at the opening ceremony of Tampere University Hospital's "Aistihuone" at the beginning of the year 2023, but due to several problems in development, the launch was postponed to somewhere near future.

We chose a 10" tablet (Samsung Galaxy Tab S6 Lite 2022) as our mobile device because it is small enough for the use of children but still bigger than mobile phones and for that reason a better option for playing our game with its bigger view. It was also preferred that the device uses Android OS because Oculus Quest 2 also runs on it, and we hoped that we would have less compatibility problems that way - even though with Unity it is possible to build the same game for different platforms.

For the multiplayer version of the game, we needed to add a networking system for the game. We were recommended to use a free Unity library called Mirror, and after quick testing we decided that it would serve us well for the purpose. With Mirror we could just add networking components to our existing GOs, and it seemed simple enough.

In addition to multiplayer and tablet option we also had some more ideas for the game, but during the development process it became clear that only these two new things would be something that we could achieve in given time.

4.3 Forest Friends for VR Player

For a VR player, the game is in the first person. Movement happens by looking around and moving inside the Guardian area, that is an area that you need to set when you start using Oculus Quest 2, and that is used to supervise that the player stays safely in a clear area (Meta, 2023). Also, in each Scene the movement of the VR player is limited to a small area that is bordered by tree stumps. This way the needed area in real world to play the game gets limited naturally, and there is no need for a locomotive system inside the game when all the movement needed can be done by just reaching out with hands and turning in place.

The interaction with the game environment happens with Oculus controls. Inside the game they are presented as hands that follows the movement of the controllers. They also follow how the player's fingers are on the controller. For example, if the player's thumb is not on the button, then the hand inside the game shows that the thumb is up, and when the thumb is on the button, or if the button is pressed down, the thumb is presented to be down also in the game. These hands, though, are presented only for the player itself, and in the multiplayer game a VR player is presented as a robot for the other player, and that robot does not have this kind of hand tracking set up. VR players view is presented in the Figure 1.

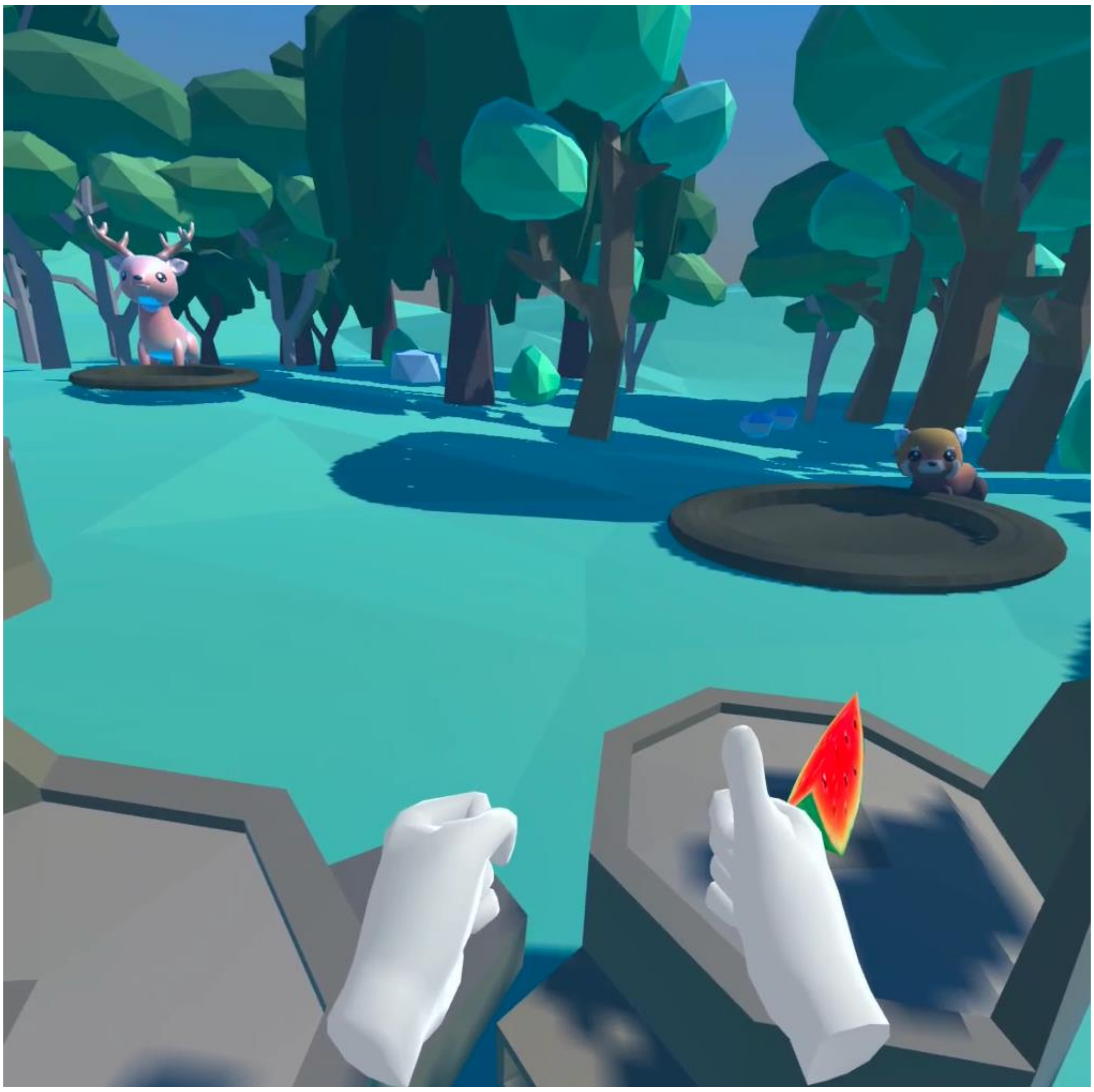


Figure 1: VR player's view during the play.

4.3.1 Tutorial Scene

The game starts from the Tutorial Scene for VR player. This Scene is made for practicing and for starting either a single-player or a multiplayer game, and it is shown from the bird perspective in Figure 2.

All five animals are present in this Scene, and they have a sign behind them that tells which is the most preferred food for the animal. The sign also shows the food that the animal likes the least, indicating it with a heart that is not colored. There are also additional two signs in front of the player: the first shows the button that pauses the game and the second shows how to grab a food item. All these signs are shown in Figure 3.

The player is surrounded by five tree stumps, each with a different food item on top of them. When the player grabs them, a similar food item spawns, so in this Scene there are no limits for throwables.

Behind the player there are two targets, and both have a sign with an explanatory text: the first starts a single-player game and the other one starts a multiplayer game.

In the game you can grab a food item by moving your hand on top of it and then pressing down and holding the button with your index finger. You can throw the item by imitating a throwing gesture and releasing the button when you would release the grip just like you would if you were throwing in real life.

In the Tutorial Scene there is an area in front of each animal, that you try to hit with the food. If your throw succeeds, the animal jumps and starts to eat the food. There is also a sound that indicates that the throw was successful. If that food happened to be the food that was marked with a red heart on the sign behind the animal, then also a burst of floating hearts in front of the animal appears for a short amount of time. This indicates that the animal favored that food and is shown in Figure 3.

When the player has had enough practice, they can start the actual game by throwing a food item to one of the targets opposite to the animals. In the next chapter 6.1.2, I will go through the Single-player Scene, and in the chapter 6.3 I'll explain the Multiplayer Scene.

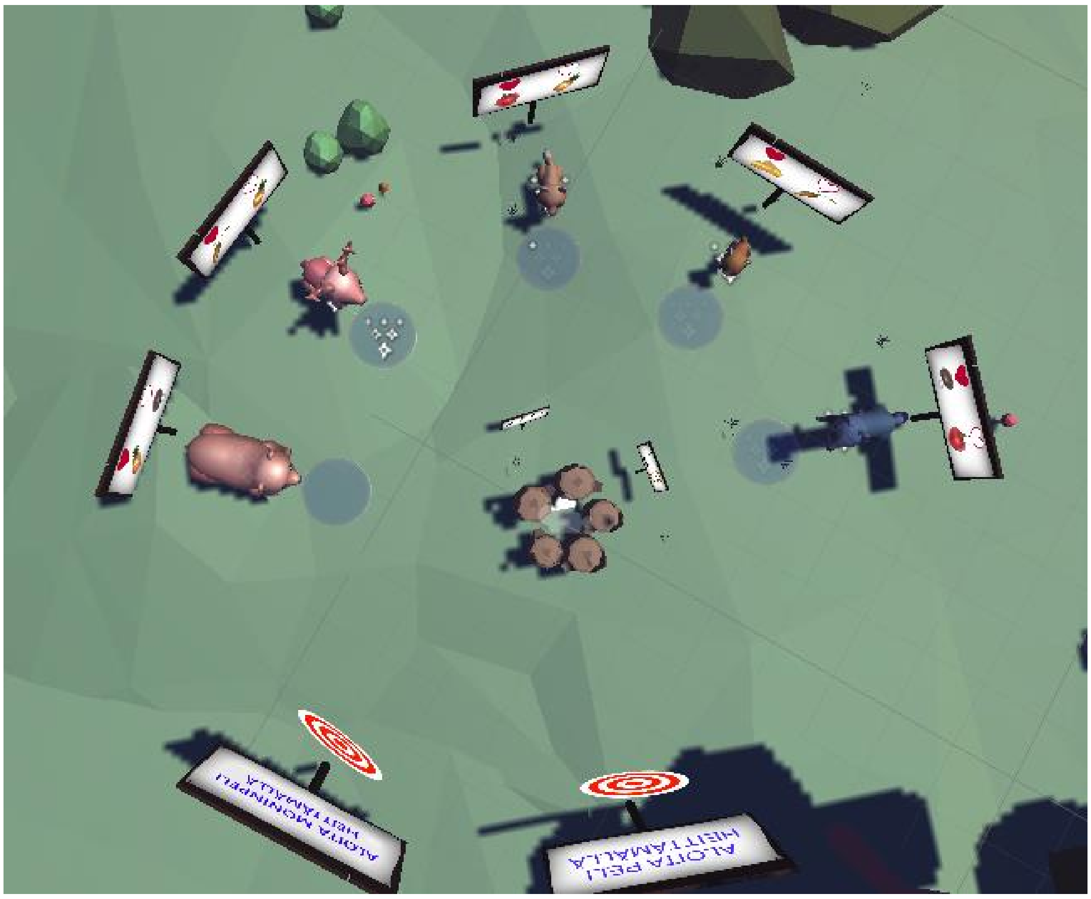
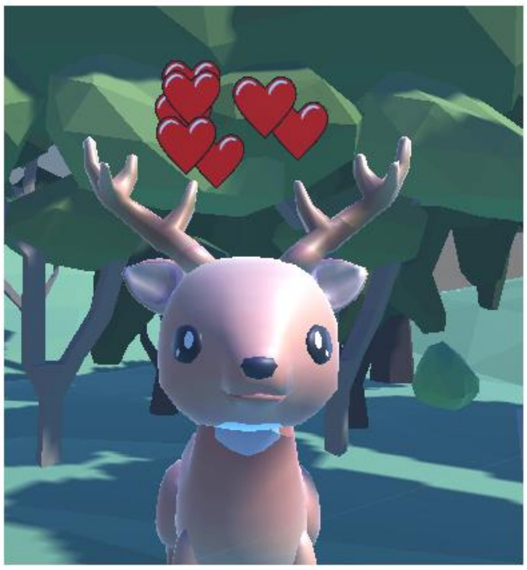


Figure 2: Tutorial Scene from above in the Unity editor. At the centre of the image there is the player's area in between the tree stumps. Animals are lined up in half circle on the upper part of the image, and both targets that start the game are on the bottom part.



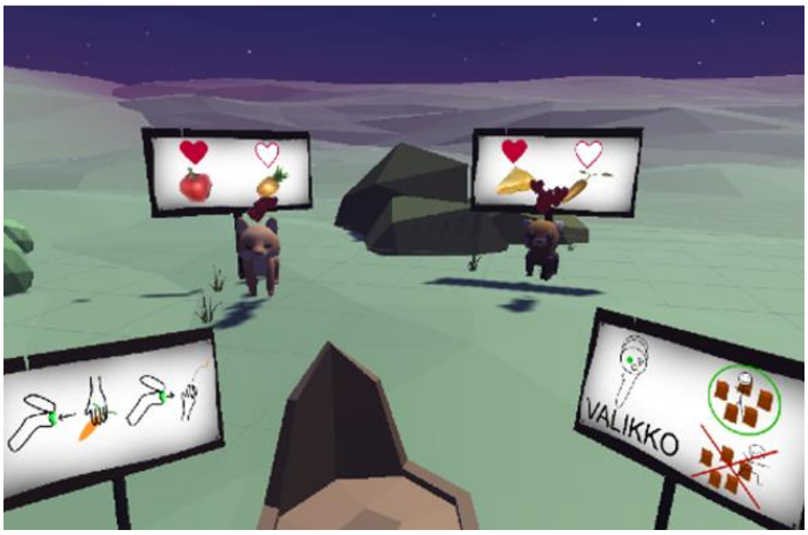


Figure 3: Upper part shows a deer with hearts above indicating that it has had the most preferred food. Lower part shows some signs of the Tutorial Scene, where two closest signs show how to crab a food item and how to get the menu. Two other signs show what foods fox and red panda prefers the most and the least.

4.3.2 Single-player Scene

When the player starts a single-player game, the player character is spawned in the center of tree stumps – just like in the Tutorial Scene. A bird's-eye view of Single-player Scene is shown in Figure 4.

Compared to the Tutorial Scene, this environment is lighter and has more trees, and all five animals are lined up in a circle formation surrounding the player. There are also no signs that would help the player to choose the right food for the right animal, so the player must remember this information from the Tutorial Scene.

Each of the five tree stumps, that surrounds the player, has five food items. When one food item is picked, another one will not appear.

Instead of a target area, there is a wooden bowl in front of each animal, that the player is supposed to throw the food. This works just like the areas in the Tutorial Scene: when the player succeeds in throwing the food item to the wooden bowl, the animal jumps and starts eating the food. There is one sound if the food item touches the bowl, and if it stays on that bowl there will be a set of three sounds, that indicates that the throw was successful. So, if the player throws a food item so that it touches the bowl but continues moving and gets off the bowl, then the throw is not accepted, and there will be only one sound and no points.

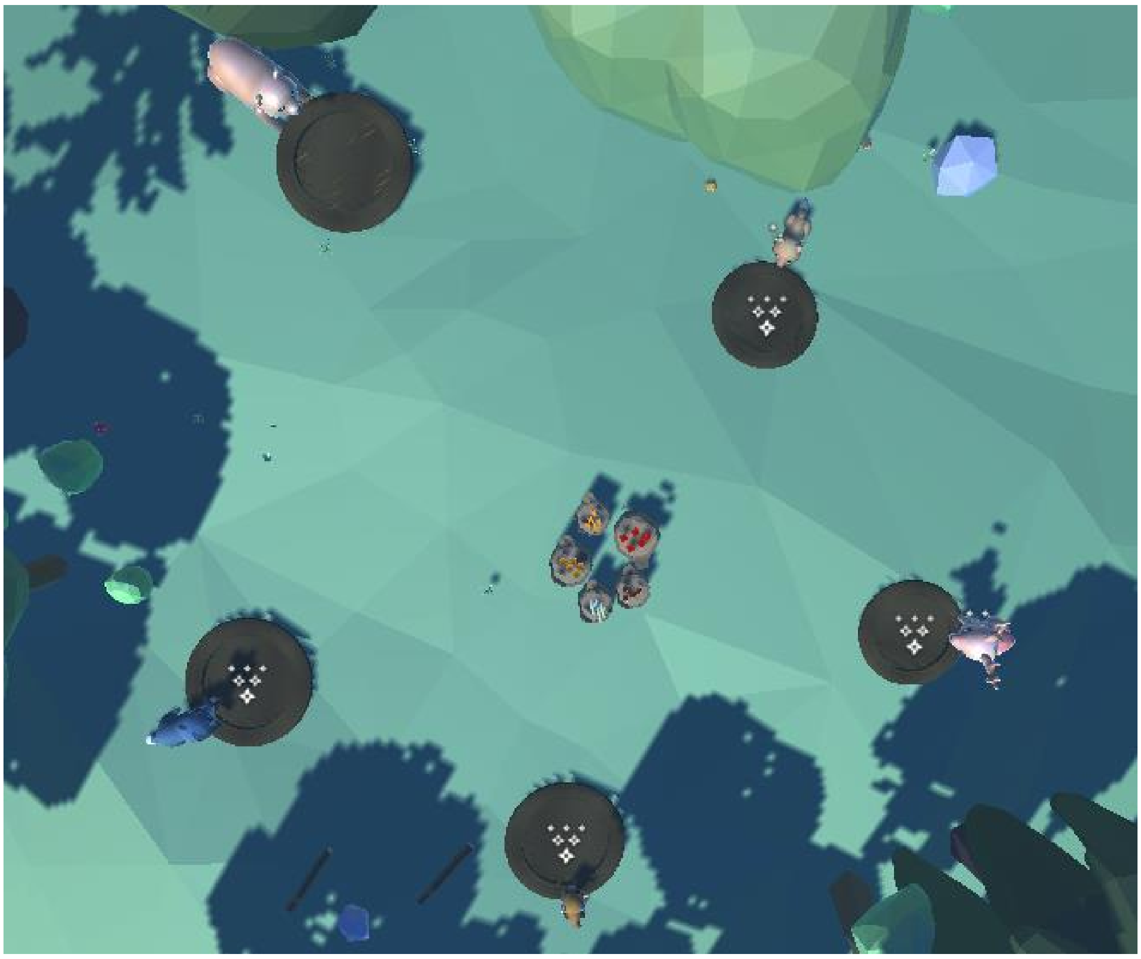


Figure 4: The Single-player Scene from the bird's-eye view in the Unity editor. Animals are lined up on the circle surrounding the player area, that is at the centre of the image.

4.4 Tablet Player

As seen in figure 2, for tablet player the logic of transitions between the Scenes is the same as for the VR player, with one exception: instead of the Tutorial Scene there is the Menu Scene.

4.4.1 Menu Scene

Menu scene contains simple menu made with UI buttons. It has simple and uplifting background music, and the player can choose from starting a single-player or a multi-player game (the first two buttons from the top) or closing the app (the third button).

4.4.2 Single-player Scene

In a single-player game the player’s mission is to choose a food item by tapping it and then flying it over an animal and trying to drop it to the bowl that is in front of the animal. This start position is depicted in Figure 5, and from that it is possible to see, that the view is from the third person.

The player controls the bird by tapping the UI elements. When the bird is sitting on a bird post, there are three UI buttons for moving the bird plus one button to change the camera, and one button to quit the game and go back to menu. Buttons on the sides of the view turns the bird around. The button on top launches the bird to fly.

When the player changes the camera by pressing the button on bottom of the view, then the view changes to the back of the bird. This view is depicted in image 6. There also appears an arrow, that helps to assess the direction of the bird. In this view it is possible to turn the bird, change the camera, and end the game, but it is not possible to pick a food item or launch the bird to fly. The flight button is therefore removed from this view.

When the player launches the bird to fly in the main camera view by tapping the flight button, that is shown in the top part of Figure 5, the bird sets off to fly straight to the direction that it is facing. The camera starts to move also by following the movement of the bird. In this view, that is shown in the Figure 6, it is not possible to chance the camera, but instead that button has changed to drop button, that makes the bird to drop the food that has been picked. If the player has not picked a food item, then this button does nothing.

Even though the appearance of the flight button has not changed, the function of it has; by tapping it the bird flies back to the bird post.

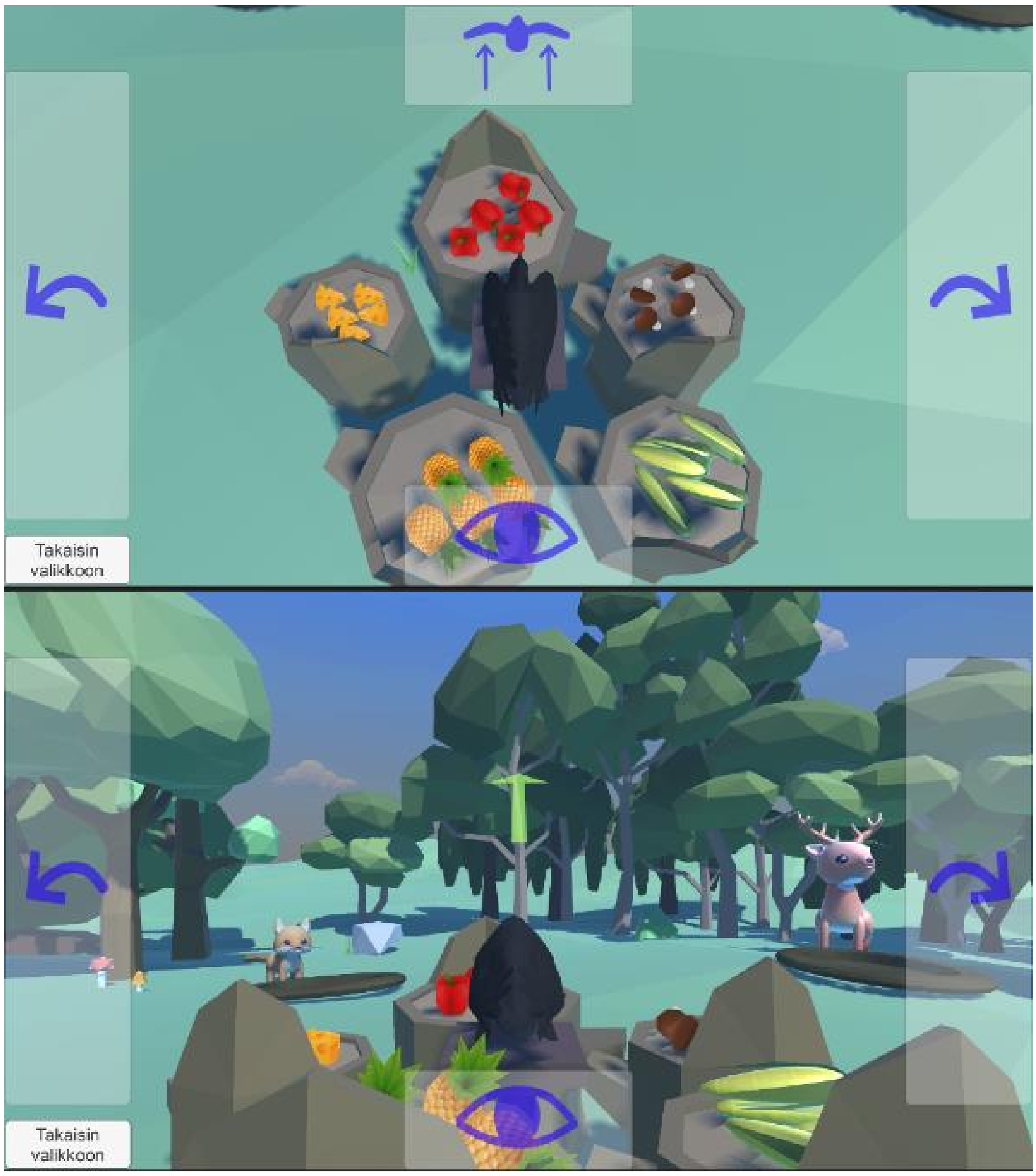


Figure 5: Upper part: Main camera view of tablet player. The bird is not flying, and the UI elements are accordingly. Lower part: Tablet player's view when camera is changed.

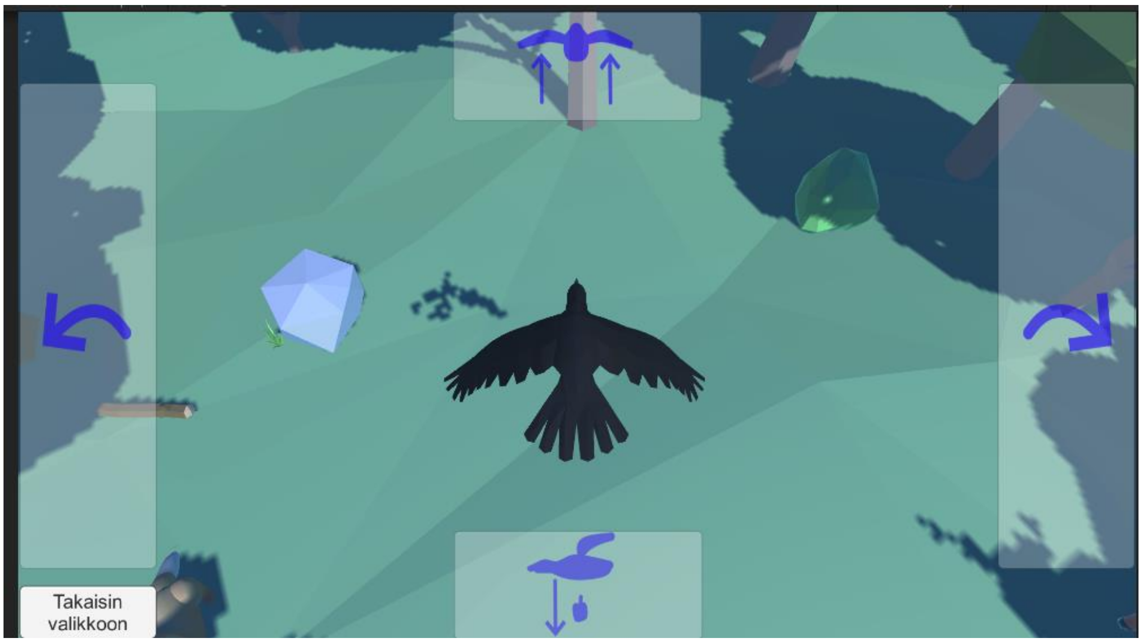


Figure 6: Tablet player’s view when the bird is flying.

4.5 Multiplayer Scene

Multiplayer Scene, that is accessible for both tablet and VR player, is shown in the Figure 7. The main difference between single-player Scenes and this is that there are two circle formations of tree stumps, and that there are no food items nor player characters in this. This is because the Scene is made for two-players, and player characters with their food items are created and spawned when they connect to the game. There are also five different new food for the second player. The goal of the game is the same as in the single-player versions: to get all the food items, that are on the tree stumps, for the animals that are surrounding player characters. When all the food is carried or thrown away from the player areas of both players, then the game ends.

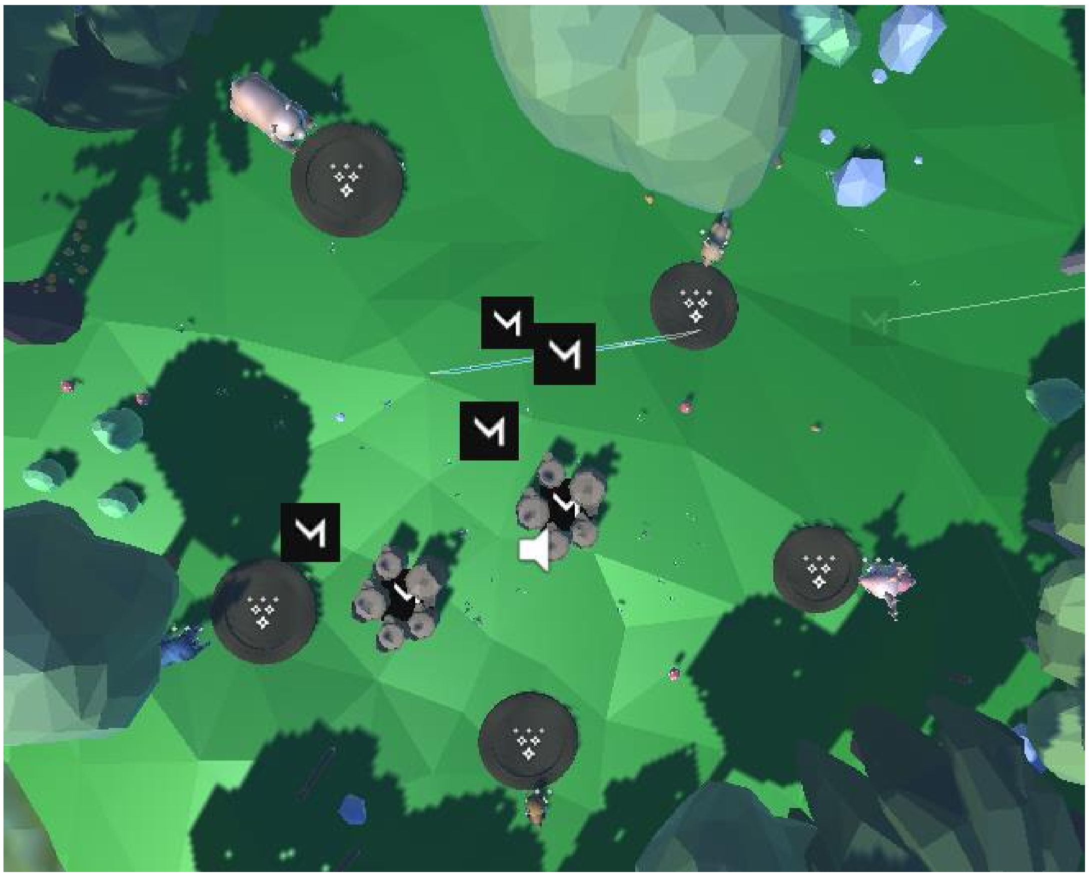


Figure 7: Multiplayer Scene from above in Unity editor. Black squares with letter M are GO's that use Mirror's Network Identity component and are not shown in the game.

4.5.1 The Process of Creating the Multiplayer Game

Technically the multiplayer version of Forest Friends was created by creating a copy of the single player Scene and then adding Mirror components to the GOs that needed to be networked. The reality was not that straightforward and the most of the GOs and Scripts needed some tweaking to make things work.

4.5.2 Join the Game or Start Hosting

In Figure 8 the logic of joining the game has been depicted. So, when a player starts a multiplayer game, it starts a Network Discovery instance to find servers. If Network Discovery did not find any servers, then it either will wait for number of seconds (this is set to 2 in Forest Friends) and try to find servers again, or if the limit of tries is exceeded (set to 3 in Forest Friends), then it starts hosting.

Because there are not supposed to be many instances of the game running at the same time, the game does not offer any way to choose the host that the player wants to connect. This makes it easy to start the multiplayer game, but on the downside, in case that there would be more instances of the game running, this could become a problem.

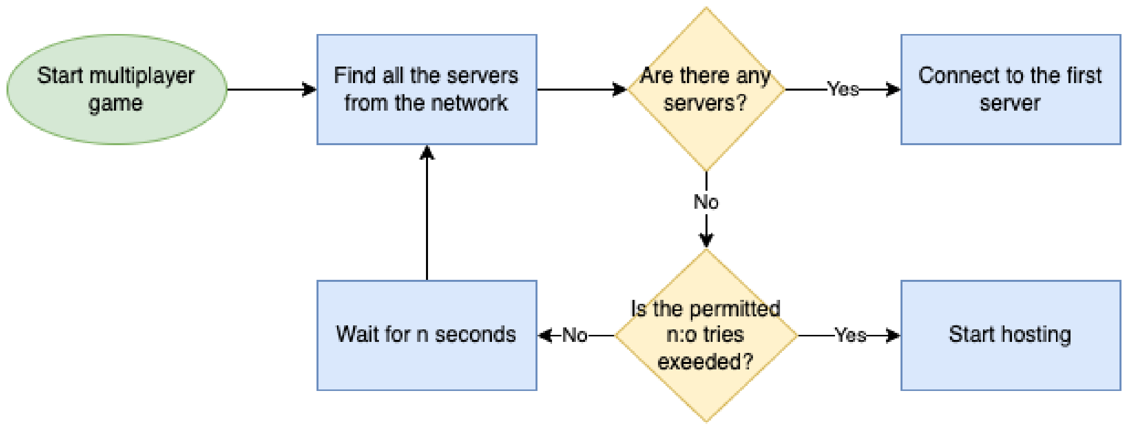

Start multiplayer game
Find all the servers from the network
Are there any servers?
Yes
Connect to the first server
No
Is the permitted n:o tries exeeded?
Yes
Start hosting
No
Wait for n seconds

Figure 8: Logic behind choosing whether to join an existing game or to start hosting.

4.5.3 Spawning the Player Characters

Unlike in a single-player game, in a multiplayer game player character cannot be in the Scene if players join the game at different times. This is one of the reasons why spawning the player character when the player joins the game is needed in forest Friends, and it is done by overriding some functions of Mirror's Network Manager class. To be able to do this, a new class that inherits the Network Manager class is needed.

In Forest Friends, this custom spawning of player character starts when the server gets started. There Network Server class's RegisterHandler function is used to handle message of type CreateMPCharacterMessage, that is a custom struct that inherits a NetworkMessage class, to fire a custom made OnCreateCharacter function when a message of this type is sent from a client.

This CreateMPCharacterMessage struct is used to save information if the client's device is a VR device or not, and so when the client connects, the client's device is checked and saved to the CreateMPCharacterMessage struct, and the Mirror's NetworkClient class's Send function is then used to send that struct as a message to the NetworkServer, that fires the OnCreateCharacter function.

This OnCreateCharacter function gets the CreateMPCharacterMessage as a parameter, and can decide, based on the information about the client's VR device, which prefab to use as a base for the player character: is it VR player's character or tablet player's character. In this function the number of players is also checked, and based on that the spawning position can be decided: if there are no clients, then the player will be spawned to the player one's spawning position, and if there already is a client connected to the server, the player gets spawned to the player two's spawning position.

After the player characters have been spawned, other items needed related to the character needs to be spawned. If the player uses a tablet, then the player character is a bird, that needs a bird pole. The set of food items is decided based on the player's number: if the player is the host, and therefore the first player in the Scene, they will get the set of food items that for player one, and if the player is just client, then they will have the set of food items that is for player two.

4.5.4 Gameplay

After the players are spawned to the Scene, the actual game begins. Each player has a set of food items, that can be thrown or carried for the animals surrounding them. Just like in the single-player games, the idea is to get rid of all the food items and get as many points as possible by throwing right foods for right animals. This time, though, the points are shared between the players. VR player's view of co-player's flying bird character is shown in the Figure 9.

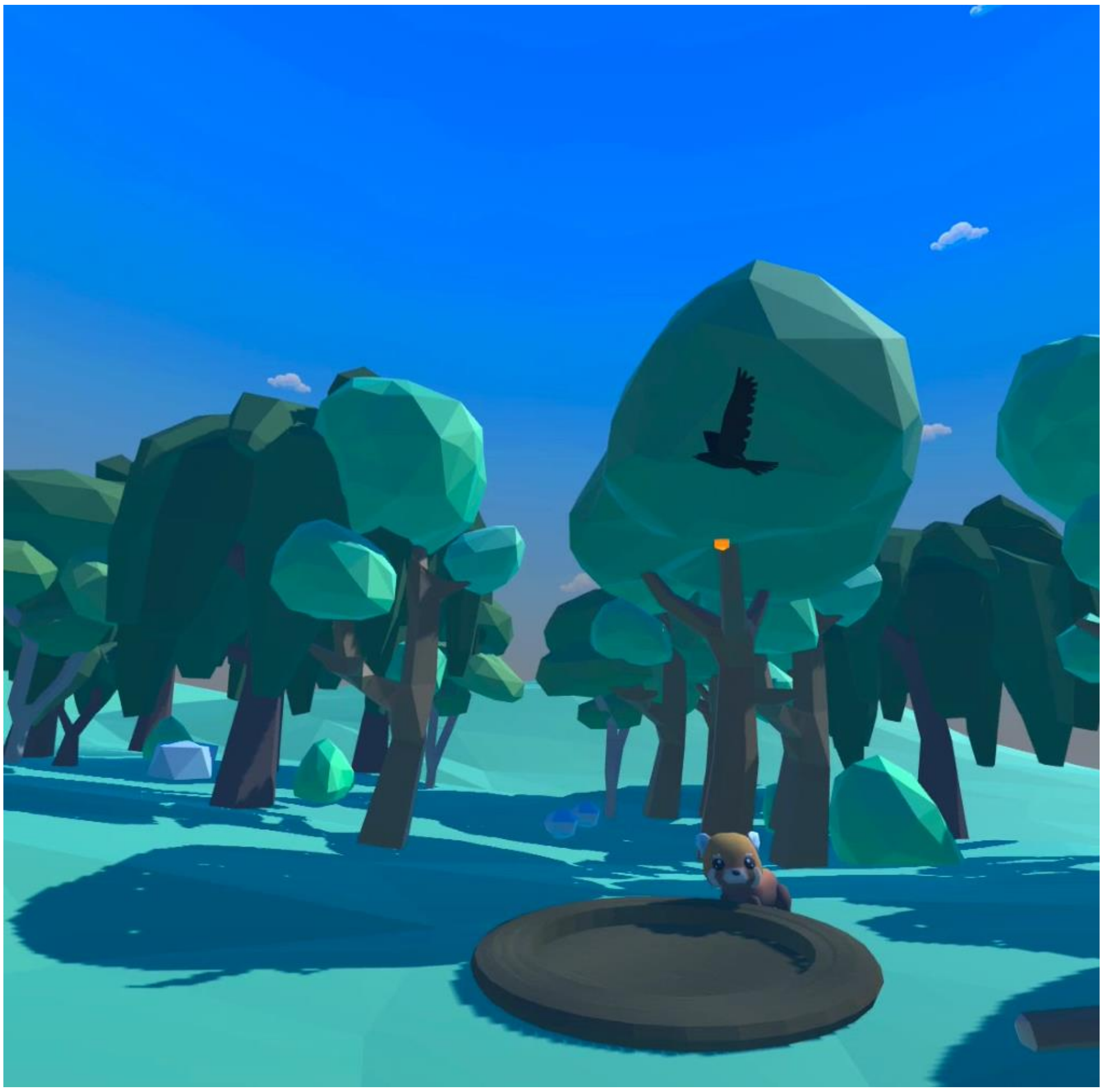


Figure 9: VR player's view of a tablet player's character dropping food. VR player's hands are not shown in this picture because the VR player's gaze is directed to a different direction than where the controllers are.

Each food item has client authority, and each of them is assigned to the player it belongs to when it is spawned. This means that instead of the server, the client has the authority to tell what happens to the object. They also have Network Transform Component, so Mirror knows that the movement and the rotation of this objects are monitored in the network. In practice, all of this means that when a player picks up and throws a food item, the position and the rotation of said item is updated from the client to the server and from there to all other clients – in this case, to the other player's client. If the food item gets thrown within the other players reach, the other player cannot pick up that food, because the other player's client does not have authority over that item.

The player character also must be in the control of the client that it is assigned to, but this is something that Mirror does automatically when the player character is spawned. What needs to be done is to restrict other player's possibility to move the character. This is done by using Mirror's function isLocalPlayer, that returns true if the client is the owner of the character, and this must be added to the Scripts that handles the controls of the game or else the controls will be mixed and by controlling the other character the other character gets also controlled.

When the player achieves to throw or carry a food item to the bowl that is in front of an animal, they are granted points. In Forest Friends, granting points is not networked, because the points are shared, and the player will get the points every time a food item is in the bowl regardless of whether it got there by the actions of the player itself or other player. This is a point for possible problems, because the state of different instances of the game are never the same but since Forest Friends is a simple game, it has been robust enough to not be modified to be networked and done in the server.

So, player characters and food items are networked with Mirror, but points and animations happen in each instance of the game independently. The game ends when the food items are used for both players, and this also needs some networking logic.

4.5.5 Ending the Game

As said previously, the game ends when both players have used their food items. For VR player, this is done by checking if there are any food items on the player area or if there are any food items in the hand of the player. If there are no food items inside the area and certain amount of time has passed, the game ends. For tablet player, instead of checking the play area, the areas under the food items are checked, because tablet player can pick food only from the top of the tree stumps, and it would prevent the ending of the game if the player would have food items inside their play area but was not able to pick them up.

This check is done only at the server side, and when the food items are gone, Game State Checker component is called and the number of players who have ended the game is increased by one. When both players have ended their game and the count of Game

State Checker is 2, the game ends, animals start to dance, ending music starts, and both players are shown their points. After a while the Scene is changed to either Tutorial Scene or Menu Scene depending on whether the player has VR or tablet, and at this point everything related to the networking is shut down.

There are also other possible ends: the player leaves the game, the other player leaves the game, or the connection of either one closes. In each of these cases the tutorial scene is loaded, and all the things related to networking are shut down.

4.6 The Scene Transition Logic of the Game

The logic behind the transitions between the Scenes for VR and tablet players is depicted in Figure 10. The only difference in the flow is that instead of the Tutorial Scene, the tablet player has the Menu Scene, that is explained in the chapter 4.2.1.

Player can end the game in Single-player Scene and in Multiplayer Scene by throwing or carrying all the food items away from the player area – that is the area made by the tree stumps. Tablet player can also quit the game and go back to the Menu Scene by tapping the UI button dedicated to that operation.

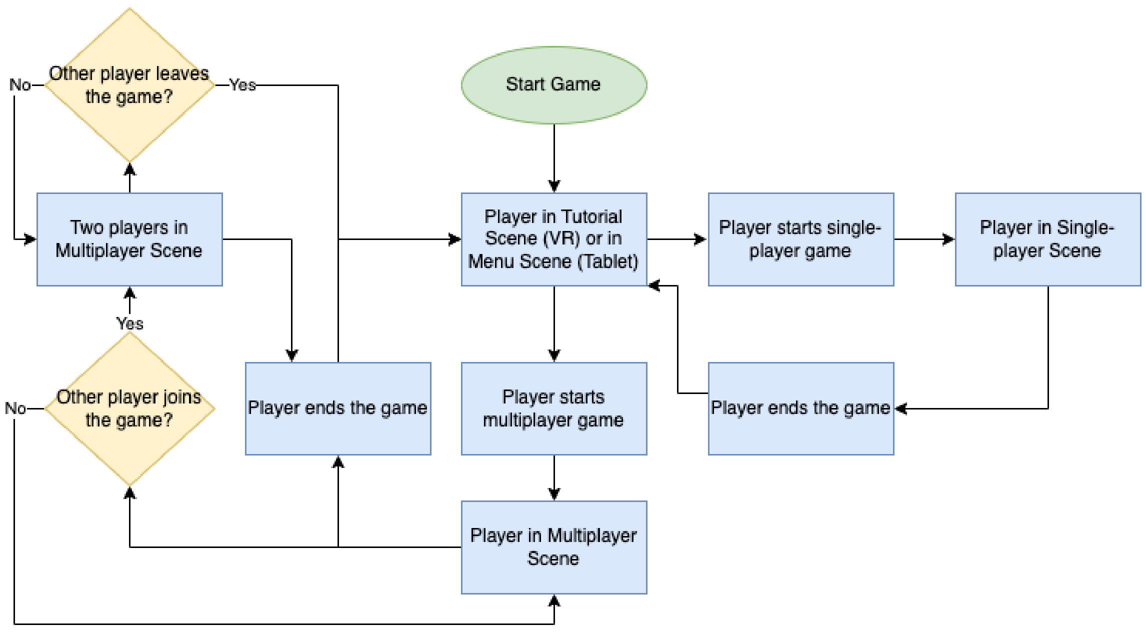

Start Game
Player in Tutorial Scene (VR) or in Menu Scene (Tablet)
Player starts single-player game
Player in Single-player Scene
Player ends the game
Player starts multiplayer game
Player in Multiplayer Scene
Other player joins the game?
Yes
No
Two players in Multiplayer Scene
Other player leaves the game?
Yes
No
Player ends the game

Figure 10: The logic of transitions between Scenes for VR and tablet players.

4.7 Summary

In this chapter I have explained in detail how Forest Friends came to existence as a part of the Software Engineering Project course at Tampere University on the spring of 2022. The development was continued on the fall of 2022, and on the spring of 2023 the game was ready. The game's motive is to feed the animals that surrounds the player's character, and it can be played either in a single-player or a two-player mode.

Forest Friends can be played with Oculus Quest 2 or with a tablet powered by Android OS. When played with Oculus, the player enters the game in first person, and tries to feed the animals by throwing the food items to the animals. When played with a tablet, the player controls a bird avatar, and tries to feed the animals by carrying the food items to them.

I have also explained how the multiplayer mode of the game works by going through the logic behind it. In a nutshell, when the player starts the multiplayer game, the game starts to search for other game instances that host a game. If it can find a host, it tries to connect to it, but if there are no hosts, it starts to host the game itself. The networking is handled with a library called Mirror, and in the multiplayer game it handles the synchronization of the gameplay.

In the next chapter I will start the literature review part by explaining some of the concepts related to Virtual Environments and Virtual Reality. The reason why this part is not in the beginning of this thesis is that I wanted to give an insight to the real-world use for the concepts, like Scenes and GameObjects, before I move on and use them in theoretical context. By doing this, I hope that I have made the theory part more comprehensible.

5 Virtual Environments and Virtual Reality

In this thesis, Virtual Environment (VE) and Virtual Reality (VR) are important concepts, and in this chapter, I will go through the history and the theoretical basis of them. By adding the concepts from Unity and game development in general to the concepts of VE and VR, and by analyzing them critically in that context, I will create the basis for my own system that concentrates on the sensory information that the VE holds. At the end of this chapter, I will create a model by categorizing the sensory information that GOs hold. This model can then be used to assess the accessibility of each category of sensory information in the real-world cases.

5.1 Asymmetric Collaborative Virtual Environments

Collaborative Virtual Environment, a term coined by Churchill et al (1997), can be depicted as a digital landscape that is made for users and data to populate. It can be a virtual space that is represented in a textual form, like Slack or Discord, or it can be complex 3D environment, like VRChat or World of Warcraft. To differentiate it from the term Virtual Environment without the word Collaborative in it, it should be designed so that it enhances the collaboration between the users. In a context of games and specifically Unity, a single-player Scene can be considered as a VE, whereas a multiplayer Scene would then be a CVE.

Asymmetry, as used by Grandi et al. (2019) in the context of Collaborative Virtual Environments (CVE), represents the possibility to interact with different means in the same virtual environment. Asymmetry rises from the matter that people who interact in the same virtual environment with different kind of devices will not have the same experience. By this definition, a game that can be joined with HMD and mobile device, is asymmetric, because interactions made to the virtual environment with mobile device are limited to touching the two-dimensional, flat screen (regarding that the interface is on the screen, and not made some other way), whereas using the HMD, like Oculus Quest 2 gives the possibility to be and to act in all three spatial dimensions with its controllers.

According to Ouverson and Gilbert (2021), in informal discussions “asymmetric VR” refers to the games where players share the real-world space in addition to the virtual environment, and the term “cross-platform multiplayer VR” is used to describe games that are played remotely, sharing only the virtual environment. However, in my own research, I did not find this kind of separation of terms: at least Steam (2023) uses category “asymmetric VR” to all the games that can be played with HMD and some other device. I also found the term “cross-play VR” that seems to refer to a game that can be played on different VR-devices, like on Oculus Rift and on Steam VR. All in all, in many cases these terms seem to be used interchangeably - at least in colloquial use.

The mixed terminology in question may stem from the fact that when virtual environments are being discussed, there is always also physical environment behind it all, and these two can easily be mixed when referring to, for example, "space" or "environment".

Ouverson and Gilbert (2021) also describe asymmetric VR as something that happens between co-located people who share the same virtual environment with different devices. They base their interpretation of the term to a former article by Steed et al. (2012), where asymmetric telepresence system called "Beaming" is presented. My interpretation is different: in that paper Steed et al. describes a system, where one person is "transported" with an immersive VR system to some location, that has virtual model that is moved by the visitor, and that can interact with the people in that location who are called "locals" in the paper. To me it seems that in this case asymmetry rises from the usage of different devices between the visitor and the locals, and therefore it is valid to use the term "asymmetric VR" also when users are not co-located.

There is also an early example of how to take asymmetry into consideration in VR environments in the paper of Churchill et al. (1998), though the term "asymmetry" is not used. In that paper, a distributed VR system for tele-conferencing, called MASSIVE, is introduced. Said application was designed so that a user with a powerful computer could do a limited interaction with a user with a not-so-powerful computer and vice versa, even though they had totally different interfaces: the other had a plain text interface, whereas the other had a graphical interface and sounds. This shows that the question of asymmetry has been a thing to be recognized in software development a long time before it was called that.

In this paper I use the term "asymmetric VR game" to describe any kind of multiplayer game that happens in virtual reality and that can be accessed via HMD and some other, non-HMD device – whether they are co-located or not.

5.2 Reality-Virtuality Continuum

Reality-Virtuality Continuum is a term introduced by Milgram and Kishino (1994). It connects the completely real world, called "Real Environment", and completely virtual world, called "Virtual Environment", and by doing that it creates an area in between that they call with a – now ubiquitous – term "Mixed Reality", that merge elements from both ends. This mixed reality can then be divided into "Augmented Reality" and "Augmented Virtuality". To make this continuum, they first differentiate real from virtual. In their paper, Milgram and Kishino (1994) introduce the distinction between real objects and virtual objects:

- Real objects are any objects that have an actual objective existence.
- Virtual objects are objects that exist in essence or effect, but not formally or actually.

For example, a ball is an instance of a real object, when completely simulated object, like a 3d rendered ball, is an instance of a virtual object.

However, Skarbez et al (2021) have criticized this continuum by stating that it is discontinuous, because technology-mediated reality is always mixed-reality and thus complete virtuality is unreachable. They base their claim on the fact that even when the technology has total control over exteroceptive senses, the interoceptive senses of the user will still be in their own control, and thus things like gravity (you will know in what direction down is) will prevent the total transportation to the virtual environment. Instead of "Virtual Environment", they propose that the other end of the continuum should be "External Virtual Environment", and it would be still part of mixed reality, and that the "Virtual Environment" would just be an unreachable endpoint of the continuum. At least until the technology, where the VR is created inside the user's brain, is invented.

In addition to the Reality - Virtuality continuum, Milgram and Kishino (1994) also presented a taxonomy, that was revised by Skarbez et al. in their paper (2021). The revised taxonomy, that is used to assess the Augmented Reality and Augmented Virtuality systems, contains following three dimensions:

1. Extent of World Knowledge (EWK)

The Extent of World Knowledge is taken straight from the original taxonomy that Milgram and Kishino presented in their paper (1994). This dimension tells how well augmented reality system is aware of its surroundings and how well it responds to changes in them.

2. Immersion (IM)

Immersion combines the last two dimensions from the original taxonomy: Reproduction Fidelity and Extent of Presence Metaphor. Both can be seen as a part of sensorimotor valid actions, a concept of immersion that result in changes to a user's perception of the environment. In addition, immersion also has effectual valid actions, that result in changes to the environment.

3. Coherence (CO)

The last dimension of this taxonomy, Coherence tells how well sensory inputs create a unified experience (Skarbez et al. 2021). This means that instead of telling what the system is intended to do, CO tells how consistently that intention is conveyed to the user (Skarbez et al. 2021). What this dimension tells depends on the context: used with the VR, it focuses on the internal things, like how consistently virtual objects interacts with each other, but when used with AR, the focus is on interaction between the real-world objects and the virtual objects (Skarbez et al. 2021).

5.3 Scene as a Virtual Environment

Games and Virtual Environments are usually considered to be different things. For example, at the IEEE Xplore service the search for “virtual environment” gives over 10 000 results, for “game” it gives over 49 000 results, but for both combined it gives only 626 results (IEEE, 2023). When browsing the results, it seems that the concept of Virtual Environment seems to be linked more to other things, like teleconference systems or e-commerce, instead of games.

There is still a good reason to see games as VEs: as Collaborative Virtual Environments were described in the paper of Churchill et al. (1997), CVEs are digital landscapes that are meant for users and data to populate. This is also how multiplayer games can be seen: the VE (or CVE) part of the game is a digital landscape that data and users populate. The games, though, have also different parts that cannot be seen as a part of VE, like the UI elements.

Even though concepts like Scene and GameObject are from Unity, I’m using them here in a general sense. Because Scene is the place where all the game content is, and GOs are the objects that the game consists of, then Scene (or in some cases, *multiple Scenes*) can be considered as a Virtual Environment, that is made from GOs – like floors, players, trees, cameras, etc. Now, when the Scene is considered a Virtual Environment, then each GO can be seen as a virtualized idea or concept, that originates from the mind of the creator of the GO, and that it tries to replicate in a virtualized form.

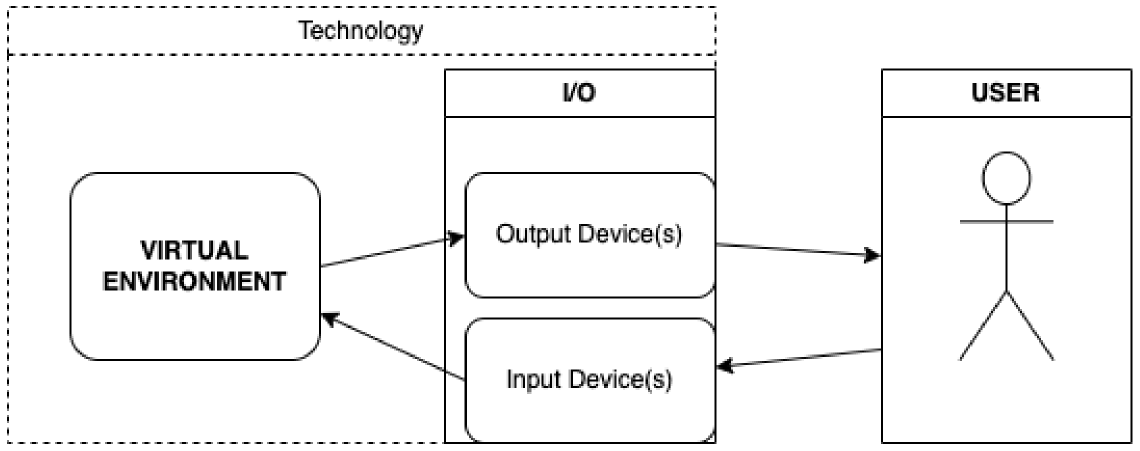


Figure 11: Three main entities of the system.

I recognized three main entities when considering the Scene as a VE: the VE, the I/O, and the User. The relations of these entities are depicted in Figure 11. To clarify the concept of I/O, the input and output devices can be separated, but they can also work as both. For example, in VR systems the HMD shows the image, and thus works as an output system, but it also tracks the head movements, and so it is at the same time also an input system.

5.4 Breaking the Reality – Virtuality Continuum

In Figure 12, I have broken the continuum: the virtual and the real are separated, and they only communicate through I/O. If there is a mixed reality, it's on the I/O system where the device translates things that happen in real world to the form that the technology can process and vice versa. The separation of the terms *virtual* and *real* moves the focus form the linear connection between the two to the part where they both end – the interface of virtuality and reality, the I/O. Making the I/O the central part is essential when assessing the difference between different platforms – a lot of the differences between the platforms can be pinpointed in that area. Especially in the case where the other platform is VR system.

By breaking this continuum into dichotomy of virtual and real we can consider the I/O as a *translator* that translates both the user input from the real world into signals that computer can handle and change the state of the VE according to them, and also express the changed states of the Virtual Environment via the output device(s) in a form of *sensory information* into the real world, that is then received by the user.

Seeing the I/O part of the system as a translator gives rise to the insight that each GO can be reduced to the information it contains, that can be then translated into the real world. This means that we also need a fourth entity in addition to the VE, the I/O and the user, that needs to be added into this diagram: the creator. The information that is added to the VE as a GO and its properties needs someone, who has had the idea that the properties are based on. There is always someone (or a group of people) that has made the VE. The VE can be considered as a collection of ideas that the creator has had and that are then manifested into the VE as GOs. This concept is depicted in the Figure 13.

Because the GO is inherently an idea of the Creator, it is logical that they should assess the quality of the translation. This idea is presented in the Figure 14.

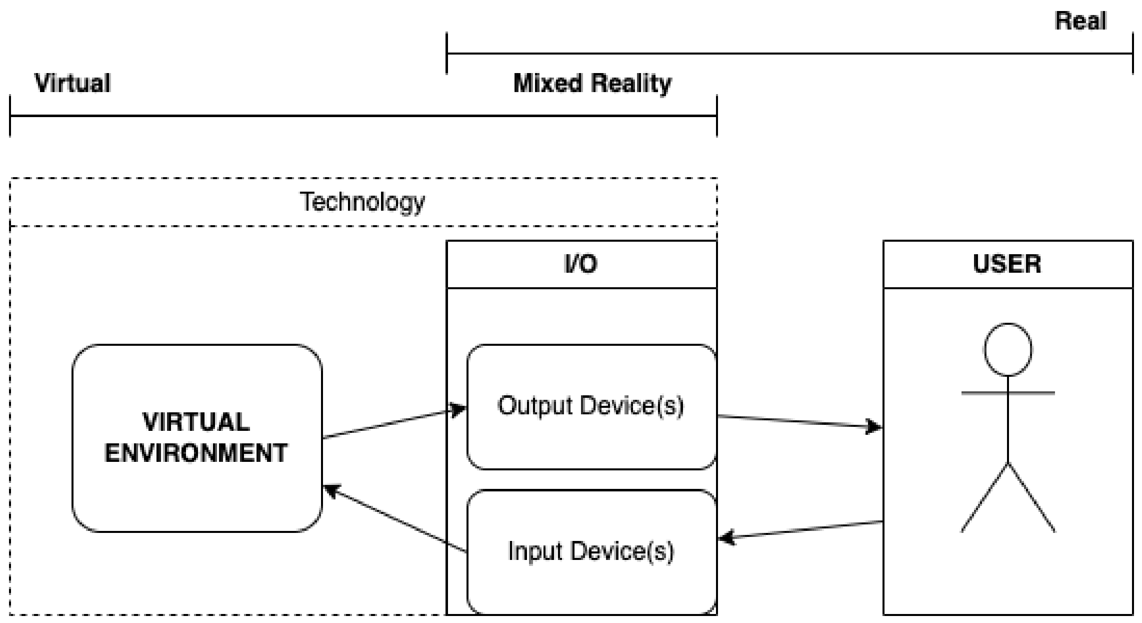


Figure 12: Virtual and real as separated continuums over main entities of the system.

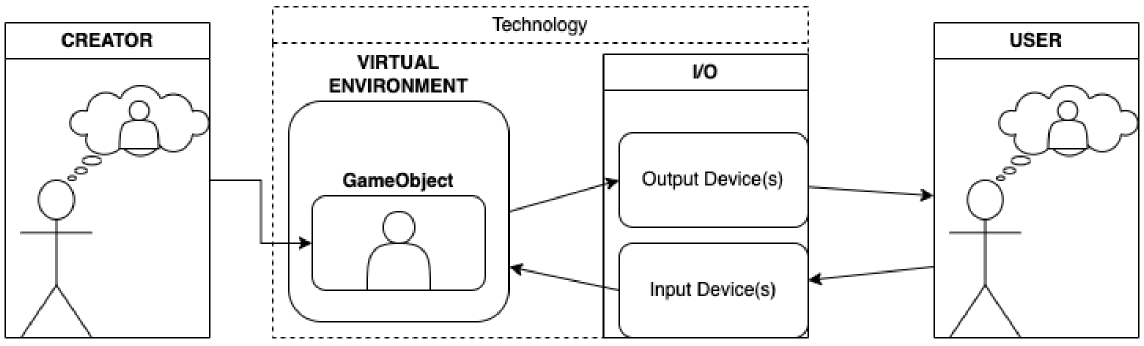


Figure 13: A GO is a virtualized idea of the creator, that then gets interpreted by the user via output system.

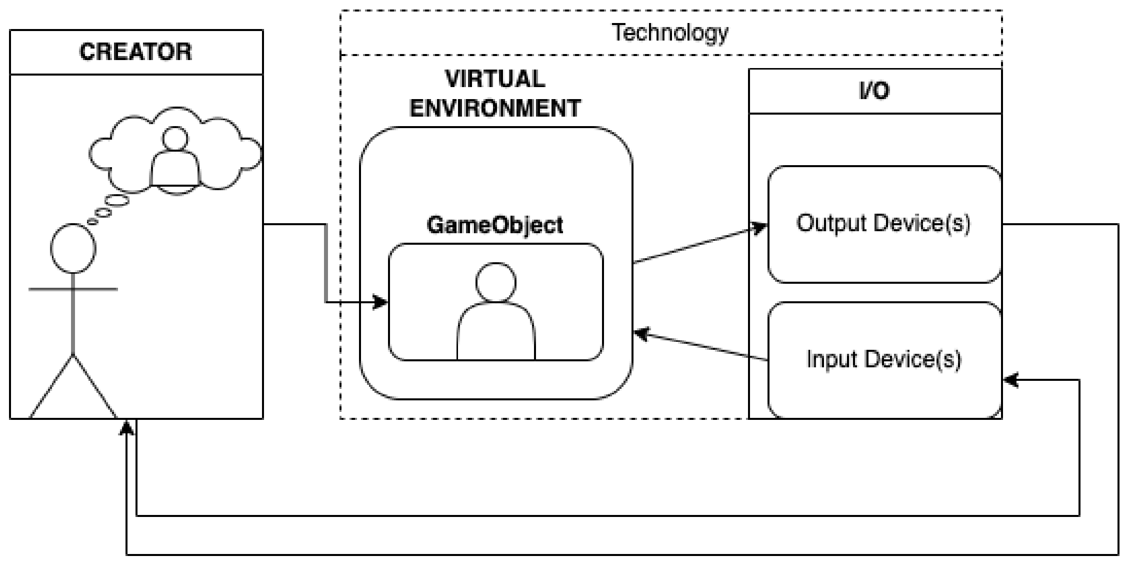


Figure 14: The Creator is a good source for assessing the translation of the GO because they know the original idea behind the GO.

5.5 It from Bit

Physicist John Wheeler presented a concept "It from bit" in his conference speech (2018), that can be shortly explained that everything in this universe is in the deepest level a simple question that can be answered with yes or no; that is, everything is *information* at the quantum level. In the field of physics this may be a radical view of the reality that can be questioned and debated, but in the context of Virtual Environments, this is the fundamental truth: everything *is* - in the deepest level - just bits, because they are created by computers that inherently operates on bits.

This idea of everything being information can be taken from that deepest level and moved further to be applied on larger concepts. In the context of VE's, for example, a virtual bunny is - in a sense – information, that the creator of that virtual bunny wants to transmit through the output system for the user. The roots of that virtual animal are in the information that the creator has about bunnies. That information is then transcribed to the VE as properties of GO. For example, the concept of bunny can be a mixed set of properties, like sounds, smells, appearances, and verbal descriptions. Based on this information, the creator then adds properties to the GO to shape it to have at least the minimum set of properties that the concept of bunny needs to be recognized as a bunny. This set of properties depends on the creator – everyone has their own sets of information, and this is something that cannot be generalized.

In this thesis I am going to focus on the information: what information the virtual object holds, and can the output device translate it so that the user can recognize it? To make this possible, first I must categorize the information that a GO can hold; then I need to measure that information by counting classified data points; lastly, I need to assess that the information the VE holds gets translated to the real world via the chosen output technologies. After this I can compare asymmetries between the information accessed via different platforms.

Seeing the GO as an idea of the creator, that is communicated into the VE and then translated through the output system to be then interpreted by the user sets the abstraction level of this work. By focusing on the higher-level concepts rather than breaking things into atoms I can make manageable classification of the GOs that the VE holds.

5.6 The Distinction Between the Player and the User

In this context, I use the term *player* to refer all the GOs inside and outside of the VE that are used as an interface for the user to interact in the VE, and the term *user* to refer to the person in the real world who access the VE through the player. In this classification, the player contains all the interfaces and the user interface elements that are needed for the user to access the VE. The player should not be mixed with *player character*, that is only one part of the player. In this classification, that is depicted in Figure 15, the user interface elements are excluded from the VE and the player character is included to it.

This separation must be done because if UI elements would be included to the VE, it would cause distracting asymmetry between the devices because they are something that are not noticeable in the other user's instance of the game.

This distinction, though, is not always easy to make. For example, in the Forest Friends the player character of a VR player has two avatars: one, that is visible for the player itself, that contains only hands that tracks the movement of the controllers; and one that is visible for the other player, that is a robot figure. In this case, the robot figure is more of a part of the VE, whereas the hands can be considered part of the UI for the VR player. In the context of multiplayer game this distinction is still quite easy to make, but when the same thing is assessed in the context of a single player game, there would not be any reason to consider that the hands would not be a part of the VE, since the only viewpoint in the VE is the VR player, who sees the hands as a part of the VE.

In a more philosophical sense these hands, that the VR player sees in Forest Friends, that unnaturally floats in the air and reflects the movements of the controllers, are not made for creating the immersion and feeling that the user is sucked inside the Virtual Reality, but they are giving the impression that the user has access to the VE from real world with them. In this sense, you can see them more as an interface than as a real part of the VE, but as I said previously, the distinction is not clear, and it depends vastly on the design and the implementation.

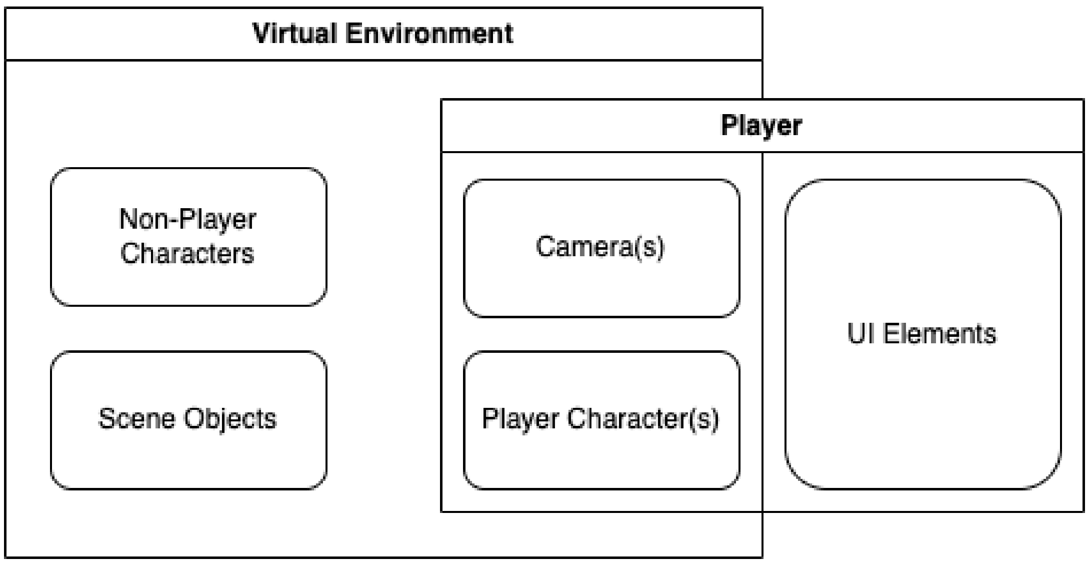


Figure 15: The relation of the Player and the VE. The components inside the VE and the Player are just exemplary to show the distinction between the VE and other parts of the game.

5.7 Classifying the Game Objects

In their book, Madhav (2015) classified the Game Objects into three different categories based on their drawing and updating:

1. Game Objects that are updated and drawn.
2. Game Objects that are drawn but not updated.
3. Game Objects that are updated but not drawn.

These categories are practical for this topic also, though I am going to adjust them a little so that they are more general and not so tightly bounded to the visual information. Therefore, instead of *drawn*, I am going to use the term *notice*. There is also one missing category, so I'm going to add that to the list also. New four categories would then be:

1. Game Objects that are updated and noticeable.
2. Game Objects that are noticeable but not updated.
3. Game Objects that are updated but not noticeable.
4. Game Objects that are not updated and not noticeable.

The first category, GOs that are updated and noticeable, means objects like characters, that are the most important objects of the game. This class can have things like player character, collectible items, and non-player characters.

The second category has items like scenery objects, that are noticeable but not updated. For example, background ambience sounds, that are played in the Scene, goes to this class, and so does trees that are not in the area that the player can reach and therefore not interactable.

The third category, Game Objects that are updated but not noticeable, has things like camera, that follows the player but is not noticeable itself.

The fourth category, that has things that are not updated or noticeable, has only some edge cases, like GO that works as a container for a Script that has some data that only gets read. This category might not be practical or common, but possible anyway.

5.8 Classifying Game Objects by Sensory Information

Game Object can be seen as a container that holds different types of sensory information. I divide that information into five main categories based on the human senses: *visual*, *auditory*, *tactile*, *gustatory*, and *olfactory*.

This classified sensory information can then be divided into two categories: *static* and *dynamic*. Static information means the information that lets us know that the object exists. Dynamic information tells if the object has movement, like movement in the spatial dimensions, animations, or transitioning. If the object has dynamic information, it does not have to have static information; for example, if the object's movement is visible via some other object, then the static visible information can be zero, but it still has some dynamic information.

Static information can be - and usually is - a bundle of information. Like the concept of a fox has a lot of visual data – it must have some colors and shapes that are considered to belong into foxiness. Or the other way around, the object has some colors and shapes and we *interpreted* it to be a fox, and we can do that only if we already have a concept of fox in our minds. This is the reason why I will not divide static information into subcategories; if the Game Object in the VE has some static sensory information, then it has it regardless of whether the user can sense it or not. Just like in the real life, a fox just has the visual information, and that information is then interpreted by the viewer as a fox. In the VE, the concept of a fox is a game developer's or designer's idea, that they have "materialized" (if that makes sense in the context of virtual environments) into the VE.

The dynamic information can be divided into three subcategories: The first subcategory is *movement*, that tells if the object moves in spatial dimensions as well as if the object rotates. The second subcategory is *animation*, that tells if the object is animate, and it includes things like animations or sounds related to those animations. The third and last subcategory is *transformation*, that tells if the object appears or vanishes or changes to a different object.

Not all the information is available all the time, and this is the reason why the information can be divided into *potential* and *actualized* information. Actualized information is something that is in the VE and does not need any actions to be accessed – though actualized information may be something that *the user* needs to do something to get access to it, like moving the player inside the VE. In contrast, *potential* information is something that needs something to be done before it is accessible – like an animation that plays when the user succeeds in something; or an items potential to be moved by the player.

Though, I am not going to make the distinction between the potential and actualized information in this thesis but knowing that not all the information is in plain sight helps finding the hidden.

One important point in this classification process is that it does not consider how the output system will interpret the sensory information – the sensory information is supposed to be classified by the internal logic of the VE. For example, if a GO had information about its smell, it would be classified as olfactory information even if none of the devices that the game is played are supporting that modality. It is then designers' and developers' task to decide how they are going to translate this information for the modalities they have access to. This approach provides knowledge about the original idea of the GO and this knowledge can be then used, for example, when new modalities are brought into use.

The classification, without division to potential and actualized, is presented in the table 1.

Game Object					
Name	Information				
	Type	Static	Dynamic		
			Movement	Animation	Transition

Table 1: A table for the classification of the sensory information of a GO. Game object has a name and a list of information objects. Information object has the type and if it is static information. If it has dynamic information instead, it can be either movement, animation, or transition.

5.9 Addressing the Information of the Scene

The Scene has also some information that needs to be addressed. In Unity, Scene has the information about the spatial dimensions – though in practice there are always three dimensions. There is also some information that is revealed via the actions of several GOs and therefore cannot be found by concentrating on single entities. For example, in Forest Friends, when the game ends, all the animals start to dance. This information is something that you cannot find by focusing on a single character. This kind of information can be found by analyzing the code and searching for functions, that call functions of more than one other GO.

These things may be obvious but are also needed to be addressed for drawing the limits for the information gained from the classification of the GOs, that I presented previously.

5.10 Summary

In this chapter I have presented the theoretical background for my endeavors in creating a system that can be used to assess the asymmetry between different device’s capabilities in accessing the sensory information of a game.

In short, a game can be seen as a VE that is accessed by the user with some kind of I/O system. VE is built with a collection of virtual objects, and these virtual objects are composed by adding different kinds of sensory information to them. And because everything that can be experienced by the user in a VE is made from the sensory information, then an I/O device’s capability to access this information tells a lot about the game’s compatibility for that specific device. This capability can be tested by trying to access the sensory information of all the GameObjects with different devices.

In the next chapter I will use the concepts that I have defined in this chapter to create a practical test case for Forest Friends and to execute it.

6 Testing the Accessibility of Sensory Information of Forest Friends with VR and Tablet

In the previous chapter I explained the theoretical basis of VE and VR and introduced a model that I created by combining the concepts of those two to the concepts of game development – especially to the concepts of development in Unity. In this chapter I will use that model to in a real-world case by using it to create a test case for Forest Friends.

6.1 Classifying the Sensory Information of GameObjects

To be able to test the accessibility of sensory information, first I must classify the GOs in a way I described in the previous chapter. In this experiment, though, I will concentrate only in static information, because finding the dynamic data of GOs is much harder task. This is because most of it can be found only by analyzing the code itself. I am also ruling out the Scene information. For now, the static data added with the animation data will give enough information to showcase how well this model works in finding asymmetries in accessing sensory information between different platforms. My experiment is done in the Multiplayer Scene of Forest Friends.

There is no gustatory or olfactory information in the game, so we can concentrate on searching just visual, tactile, and auditory data.

To get the static visual data, *Renderers* must be searched for. So, if a GO has a renderer and it is enabled, then that GO has static visual data. As described in Unity manual *"A renderer is what makes an object appear on the screen."* (Unity, 2023). To get the animation data, one must find the Animator components and then count how many animations each GO has.

The auditory data can be found by searching *Audio Source* Components. Each of these Components adds one auditory data point.

If a GO has a *Collider,* then it can be considered to have tactile information. Each Collider Component adds one tactile data point.

I made a Script that runs through all the GOs in the Multiplayer Scene and saves the name of that GO, the type of sensory information, and the count of that typed data points, and the names of each of those data points, to a simple data structure, and then writes that information to a file. This data structure is shown in the Figure 16.

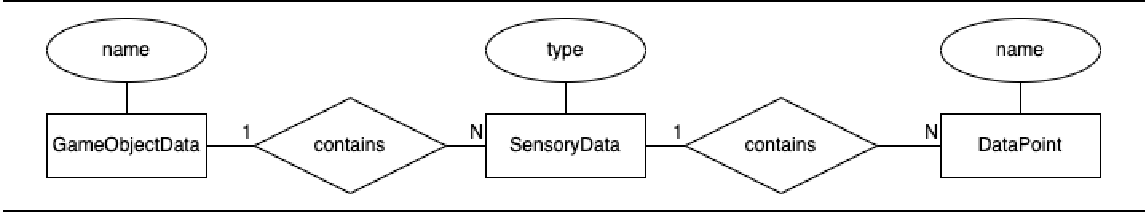


Figure 16: The structure of the GameObjectData class.

The Script found 407 data points, that were distributed as follows:

- Visual data points: 198
- Auditory data points: 52
- Tactile data points: 157

6.2 Testing the Accessibility of Sensory Information with a VR Device and a Tablet

I tested Forest Friends by playing it with both VR and tablet devices and meticulously going through all the sensory data points that my script had found. I marked 1 if the data was accessible, 0.5 if the data was partly accessible, and 0 if the data was not accessible at all. The results that I got and that show the asymmetry between different platforms are shown in the table 2.

Type	**Total no. data points**	**VR Accessed**	**VR Accessed %**	**Tablet Accessed**	**Tablet Accessed %**
Visual	198	198	100	173	87,37
Auditory	52	52	100	52	100
Tactile/Visual	157	137	87,26	116	73,89
Total	**407**	**387**	**95,09**	**341**	**83,78**

Table 2: Sensory information datapoints and their accessibility on different platforms. Tactile data is interpreted visually, thus the notation “Tactile/Visual”.

To be able to check the name of the GO I was testing in the game, I had the Unity editor open during the testing. There it was relatively easy to find the object by typing it to the search bar and then click the object from the hierarchy to highlight it from the Scene view.

I tested the visual data by seeking the objects that were on the list: if I could see the object and recognize it, it was usually considered accessible, though depending on the context: a tree that was mostly blocked by another tree was considered accessible, but an animation that was sometimes blocked by the player character was considered only partly accessible. This is because a tree is a part of the environment, and it is normal for environments to have things scattered randomly so that they are not visible from all the viewpoints all the time. Partly shown tree then serves its function of being a part of the environment perfectly, whereas blocking an animation, that happens because the player did something, hides information that is important for the game. This kind of blocking happened only with tablet, and it is the only reason for the asymmetry in visual information between VR and tablet in Forest Friends.

To fix this, the information that the animation is indicating could be transmitted via other modalities or shown in the UI. In the case of Forest Friends, this animation that I'm referring to indicated that the player had achieved a goal to get the food to the target bowl, and that the animal was excited and started to eat the food. The achievement part is already indicated with a sound, but the excitement of the animal and the eating parts could also be expressed with sounds.

Most of the auditory data was hidden so it was needed to do something to access it. The only exception was the background soundscape, that started playing immediately when the Scene was loaded. Other kinds of sounds had to be activated by doing some tasks; in practice, by getting a food item to the bowl in front of an animal or by finishing the game. The auditory data was totally accessible with both devices, so there is no asymmetry here. Though, if spatial audio was used then the VR device would probably have better access to the spatial part of the auditory information.

The biggest finding was the unnecessarily small use of sounds compared to the use of visual and tactile information. For example, every collision could make a sound and animals could express things both visually and auditorily. Using different modalities makes the system more robust by getting the responsibility off from a single modality and sharing it with others. Like I pointed out before, if sometimes the player character blocks the animation, it does not matter that much if that same information is expressed also with sounds.

With tactile data, the first thing to notice is that it must be interpreted visually, because the game does not produce haptic feedback. Adding haptic feedback to the game would be possible with the interfaces that both Oculus and Unity offers (Oculus, 2023; Meta, 2023) and adding it could increase accessibility and immersion of the game.

Since the game does not take an advantage of these interfaces yet, the tactile data was interpreted so, that I used another object with Collider Component and tried to collide it with the object that I was testing. If the object did not go through the object that I was testing, but bounced off, then the tactile data point was considered accessible. In practice, this was done with the VR player by throwing food items at the object under testing, and with the tablet player by picking a food item, then flying over the object that was under testing, and dropping the food at it.

During the testing, two different types of accessibility problems emerged: either the object was out of reach, or the tested object was so small that the collision was not noticeable. The problem of objects being out of reach happened only with tablet and was due to the limited movement area of the tablet player. In contrast, with the VR player I was able to hit all the objects with food. This is the sole reason for the asymmetry in tactile information between the devices, though I do not think that in this case the asymmetry of the devices shows any actual problems of the game, since the play area of the tablet player was restricted on purpose, and the objects that it could not reach were merely decorative. The other problem with objects being too small for the collision to be detected can be fixed by deleting the Collider from those objects. The objects, that I am referring here, were merely a decorative element, like small branches and mushrooms, so removing Colliders from them would not affect the gameplay in any way. Depending on the number of this kind of objects it may have some small benefits for the performance of the game.

6.3 The Evaluation of the Results

The classification and the assessment of the game by sensory information gives two types of information about asymmetries of the game: the asymmetry between different types of sensory information *withing the game,* and the asymmetry of accessibility of certain sensory information *between the devices*. The first one tells how well different modalities are considered in the game, and it can be estimated after the classification process by comparing the number of datapoints between different sensory information types. The second one tells how accessible different data points are with different devices, and it can be measured after going through all the found data-points with different devices and assessing the accessibility of each of them.

The information about the distribution of sensory information within the game gives some interesting insights about the game's environment, and it can be found from the table 2. The amount of visual data is much larger than other types of sensory data, especially compared to the amount of auditory data. Tactile data is also something that is experienced visually by seeing that the GOs are not going through the surfaces of such objects that have a Collider component (that is not set to be a trigger). So, all the tactile information is translated to visual information, and thus, it means that almost all the sensory information of the game is experienced visually.

When we compare the numbers between the devices, we can see that almost everything was accessible with the VR device, except some tactile datapoints. In comparison, there were a little more tactile datapoints that were not accessible for the tablet player and in addition to that, also some visual datapoints were not fully accessible for them. From these the partial accessibility of some of the visual datapoints was an actual problem that was caused by the tablet player character blocking the visibility to some animal characters animations.

All the other asymmetries in the accessibility of the sensory information between the devices were not design problems, but properties, that were caused by the limiting of the play area for the tablet player. This kind of asymmetry happens naturally when the players have distinct limitations, and whether they are bugs or properties depends on the case.

6.4 Discussions

One problem that this approach has is that all the data is considered equally valuable. This means that every tree outside the play area gives same one data point as the object that is more relevant to the game, like some food item or animal. This problem can be solved by giving weight for each data point – the more relevance it has, the more weight it carries. This weight would be some constant, that the data-point was multiplied with.

One way to assert weight for a data point is to assess its importance for the game: one can play the game when some scenery object is missing, but when the player object is missing, the playing gets impossible. These objects with different level of importance should have weight based on that importance.

In some cases, there are also some objects that are used together to form a bigger entity. For example, in Forest Friends there are trees and other objects, like mushrooms and bushes, that are used to form a forest. This kind of concepts are something that cannot be assessed by going through all individual objects in the Scene, but they need to be recognized by the creator of the Scene and assessed independently. For example, after the testing it was clear that the game's environment did not seem like a forest, but had a more island-like vibe, because there were only few trees that were all scattered close to the player. To tackle these kinds of issues, there should be some kind of checklist for the information contained in the whole Scene.

7 Summary

The objective of this thesis was twofold: first, to show how a multiplatform multiplayer game can be developed by going through a case example as I wrote about my game project called Forest Friends, and second, to create a system to assess a game's suitability for multiplatform playing by comparing the asymmetry of accessible sensory information of the game between different platforms.

I conducted a literature review of existing research on topics related to Virtual Environments and Virtual Reality and used the concepts I gathered to create my own model for a system to assess the asymmetry in the sensory information of the game's environment. Based on the model I created, I wrote a script that gathered all the data points that contained some sensory information from the GOs of Forest Friends. After this I tested the game with both Oculus Quest 2 and a tablet, assessed the accessibility of each datapoint, and lastly compared the asymmetry of the sensory information between the devices.

My analysis revealed that my system can be used to find asymmetries in the sensory information between the devices. I was then able to use that data to make improvements to the game. The system also revealed useful information about the division of the sensory information within the game.

In conclusion, this novel approach to assess games by their sensory information can provide useful information that helps developers and designers to improve their product and make it a better fit for different platforms. This approach, though, may fit only for certain types of games, where the environment is a big part of the gaming experience. The system is also very heavy for games that have big environments with lots of objects, because each of them are tested independently. Nevertheless, these problems can be tackled by giving more weight for the most important objects and leave out the less important parts of the game from the testing.

8 References

Baptiste, Travis; Craig, Russell; Davis, Anthony & Stunkel, Ryan. (2022). *Unity 3D Game Development*. Packt Publishing.

Churchill, E. F., & Snowdon, D. (1998). Collaborative virtual environments: An introductory review of issues and systems. *Virtual Reality : the Journal of the Virtual Reality Society,* 3(1), 3–15. https://doi.org/10.1007/BF01409793

Grandi, J. G., Debarba, H. G., & Maciel, A. (2019). Characterizing asymmetric collaborative interactions in virtual and augmented realities. *26th IEEE Conference on Virtual Reality and 3D User Interfaces,* VR 2019 - Proceedings, 127–135. https://doi.org/10.1109/VR.2019.8798080

IEEE. (2023, 13 March). Search for "game", for "virtual environment", and for "game" AND "virtual environment". IEEE Xplore.
https://ieeexplore-ieee-org

Meta. (2023, January 16). Oculus Safety Center.
https://www.oculus.com/safety-center/

Milgram, Paul & Kishino, Fumio. (1994). A Taxonomy of Mixed Reality Visual Displays. *IEICE Trans. Information Systems*. vol. E77-D, no. 12. 1321-1329.

Mirror. (2023, 17 March). Mirror Networking.
https://mirror-networking.gitbook.io/docs/

Ouverson, K. M., & Gilbert, S. B. (2021). A Composite Framework of Co-located Asymmetric Virtual Reality. *Proceedings of the ACM on Human-Computer Interaction*, 5(1), 1–20. *https://doi.org/10.1145/3449079*

Skarbez, R., Smith, M., & Whitton, M. C. (2021). Revisiting Milgram and Kishino's Reality-Virtuality Continuum. *Frontiers in Virtual Reality*, 2. https://doi.org/10.3389/frvir.2021.647997

Steam. (2023, January 16). Steam Store.
https://store.steampowered.com

Steed, A., Steptoe, W., Oyekoya, W., Pece, F., Weyrich, T., Kautz, J., Friedman, D., Peer, A., Solazzi, M., Tecchia, F., Bergamasco, M., & Slater, M. (2012). Beaming: An Asymmetric Telepresence System. *IEEE Computer Graphics and Applications*, 32(6), 10–17. https://doi.org/10.1109/MCG.2012.110

Unity. (2023, February 9). Version 2021.3: Unity Manual.
https://docs.unity3d.com/Manual/

Wheeler, J. A. (2018). Physics of information: Information, physics, quantum: The search for links. In Complexity, Entropy and the Physics of Information*: The Proceedings of the Workshop on Complexity, Entropy, and the Physics of Information* Held May-June, 1989 in Santa Fe, New Mexico (pp. 3–28). https://doi.org/10.1201/9780429502880