CHALMERS | GÖTEBORGS UNIVERSITET

Mixed Reality Real Time Strategy Game

Bachelor of Science Thesis in Information Technology and Computer Science and Engineering

KEVIN BJÖRKLUND, ANDERS ERIKSSON, JIMMY MALMER, DANIEL OLSSON, CHRISTIAN ROOS, RICHARD WECKE

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Göteborg, Sweden, June 2016

Bachelor of Science Thesis

Mixed Reality Real Time Strategy Game

KEVIN BJÖRKLUND, ANDERS ERIKSSON, JIMMY MALMER,
DANIEL OLSSON, CHRISTIAN ROOS, RICHARD WECKE

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Göteborg, Sweden, June 2016

Mixed Reality Real Time Strategy Game

KEVIN BJÖRKLUND, ANDERS ERIKSSON, JIMMY MALMER,
DANIEL OLSSON, CHRISTIAN ROOS, RICHARD WECKE

Examiner: Olof Torgersson

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
SE-412 96 Göteborg
Telephone +46 (0)31 772 1000

Cover: The room and the table containing the game field as well as the boards for communication and squad interaction.

Department of Computer Science and Engineering
Göteborg 2016

Mixed Reality Real Time Strategy Game

KEVIN BJÖRKLUND, ANDERS ERIKSSON, JIMMY MALMER,
DANIEL OLSSON, CHRISTIAN ROOS, RICHARD WECKE
Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg

Abstract

Making video games in Virtual Reality (VR) is a fairly new and undeveloped territory and there are a lot of new problems that arise when developers step into the world of VR. The goal of this project is to create a Real Time Strategy (RTS) game in Mixed Reality (MR) and to tackle any problems that may arise. Mixed reality is VR but with real life elements incorporated in the virtual world. During development the VR headset used is an Oculus Rift Development Kit 2 (DK2). To handle the input from the users a Leap Motion controller is used and the game itself will be made in the game engine Unreal Engine 4.

An RTS game was created where the player is placed in front of a virtual table inside a virtual room. The game field, or game world, is then projected on the table much like a board game. The Graphical User Interface (GUI) is constructed around the player inside this VR room. A simple menu connected to the player's hand was created for unit production and main menus were placed on banners around the room. There are three kinds of units in the game, a musketeer, a soldier and a mortar.

The focus of the project was to minimize cybersickness and at the same time create an innovative input system for VR. Most of the problems that surfaced were due to the lack of support for a small world inside a larger one, impairing both the scale of which the player sees the models and the navigation mesh for the AI. It was essential to have the two separate worlds, the room and the table, to get the board game feeling sought after, so these problems had to be solved.

Keywords: Leap Motion, Virtual Reality, Oculus Rift, Unreal Engine, Blender, Real Time Strategy, Tablemen, Stenkross Studios, Chalmers, Bachelor Exam Project.

Acknowledgements

Chalmers University of Technology
Daniel Sjölie - For great supervision and large interest in the project and VR progress in general.
Arne Linde - For much assistance with many practical things and making the foundation of the project possible.
Mickaäl Fourgeaud - For taking time and helping out with Leap Motion problems in Unreal Engine 4.

Chalmers Ventures
Viktor Brunnegård - For taking time and granting supervision in business design.
Alexander Hars - For dedicating time to help us in details regarding a potential startup.

Gothia Innovations
Fredrik Örneblad - For showing interest and providing motivation and assistance to the future of the project.

Leap Motion
Alex Colgan - For his continuous support and input towards the development.

Contents

List of Figures

Glossary

VR - Virtual Reality: An artificial environment that is generated with a computer in order to simulate physical presence and real environments.

DK2 - Oculus Rift Development Kit 2: The second and last development kit of Oculus Head Mounted Display.

Leap Motion: An IR camera used for hand tracking.

UE4 - Unreal Engine 4: A high end game engine made by Epic Games.

GUI - Graphical User Interface: A visual interface that allows for easy interaction.

Cybersickness: The feeling of nausea that occurs when motion in VR does not relate to real motion.

AI - Artificial Intelligence: A machine intelligence that evaluates the surroundings and works towards an arbitrary goal.

Blender: A free to use 3D modelling software.

RTS - Real Time Strategy: A game genre.

MR - Mixed Reality: A broad concept featuring a combination of virtual as well as real worlds and objects that can interact with each other.

AR - Augmented Reality: A concept where virtual objects are projected and interacted with in the real world.

Motion Controller: A position tracked hand controller which can be used to interact with things in a virtual enviroment.

HMD - Head Mounted Display: The hardware used for simulating Virtual Reality.

Razer Hydra: A motion controller made by Razer Inc.

API - Application Programming Interface: A programming library consisting of functions and tools for a specific component.

W2M - World To Meters: A variable used for scaling in VR within Unreal Engine.

IPD - Interpupillary distance: A measurement of the distance between the eyes.

Mixamo: A company that produces motion capture Animations.

Behaviour Tree: A mathematical model used for AI which contains information about the different tasks the AI characters can do.

Sculptris: A sculpting tool used for soft body mesh creation.

Navigation Mesh: An abstract data structure used for units to navigate complicated spaces.

1

Introduction

1.1 Purpose

The purpose of the project is to create an RTS game in VR with a focus on implementing a way of controlling the game that feels smooth and natural while enabling an immersive and aesthetically pleasing experience for the player.

Additionally, finding ways to decrease or avoid cybersickness, which is a common problem with VR experiences, was a priority. Exploring the different options of interaction in VR will also be an objective for this project.

The game itself was a squad based tactical multiplayer game for two players, where the goal is to conquer the opponent's fortress by sending in troops and take over resource points in order to build more troops.

1.2 Scope

MR offers many new possibilities but implementing these ideas, and MR itself, can be difficult. An early idea was to make use of Microsoft Kinect cameras in order to scan a room, make a 3D reconstruction of it and then render it in real time in the game. This will give the illusion that the user really is in the same room while the environment can change and be modified right in front of the players eyes. The concept of MR will be further explained in subsection 2.2.

This, however, is not an easy task and it has not been previously implemented for use in a game. As a result, there is an uncertainty regarding the amount of time needed to implement this feature and as such the core focus of this project was to use the Leap Motion controller to enable natural interaction and drive the feeling of immersion.

A focus on implementing many different control options for the game was not of high priority. Leap Motion was the main source of input and mouse controls was added for development purposes. Support for gamepads such as the Xbox One controller

and motion controllers like HTC Vive and Oculus Touch will eventually be added in the continuation of the game, outside of the project, but it is not within the scope of this project.

There are two possible modes of play, multiplayer or single-player. Multiplayer was to be completed first because single-player mode was not a priority for the game. This is due to having to create an Artificial Intelligence (AI), that can mimic the player's decisions and control flags and troopers. This will take too much time, and time is better prioritized elsewhere. Instead, the core features of playing the game should be implemented and completed first.

The planned number of maps is just one. If there is time, once all the core features are completed, work can be put into making another map for the game.

Early in the game planning process, there was an idea that the player should be able to take control of a trooper, a "commander", and for a brief moment be able to move around the field in first person and interact with the enemies. This idea has moved down the list of priorities and only if there is enough time in the end, will this feature be implemented.

1.3 Problem Definition

There are three main problems when it comes to game development in virtual reality that was dealt with in this project. When developing a game in virtual reality all three of these problems must be considered and dealt with:

- *Cybersickness* - Cybersickness is referring to the sense of moving but not feeling that you are moving. Often causing nausea this is a common problem in VR. Often confused with motionsickness, that is essentially the opposite, where motionsickness refers to the sense of feeling that you are moving but not seeing it. [1]

- *Input* - Input and input devices is for the majority of cases not as effective for VR application as they are in 2D environments. Mouse and keyboard for example is not as viable in 3D space as it is currently in 2D applications.

- *Interface* - As for input devices, interfaces that are common today are tailored for 2D applications and are therefore not as effective for VR environments. These interfaces are commonly stuck on the screen which works fine on a computer screen but not in VR. Because of this, a new type of interface needs to be invented. [2]

2

Background

2.1 Real Time Strategy Game

The RTS genre is vast and there are many different types of RTS games. All games in this category share at least one feature, the players must make their decisions in real time[3]. This can be compared to other strategy games, such as Heroes of Might and Magic, originally by 3DO, or the more classic board game, Chess, where every player can take their time before making a move.

Taking a look at an iconic RTS game, Starcraft II[4], some common elements of RTS can be distinguished. Elements such as, gathering resources, training and upgrading troops, planning attacks and battle the enemy. All of this happening in real time has the effect of games often becoming very fast paced. This is due to the fact that the player who makes smart decisions faster than the opponent will have an advantage in battle.

2.2 Virtual Reality

A new genre of gaming is emerging, live-action virtual reality games. This is a result of Virtual Reality is entering the gaming industry and many other industries, and it requires different kind of game play.

" We say that live-action virtual reality games are "live-action games" because a player physically acts out his/her "avatar" (his/her virtual representation) in the game stage, the mixed-reality environment where the game happens." , says Valente et al.[5]

Virtual Reality (VR) is the concept of a computer generated alternative reality, in which you can step into with the help of various technologies. The concept of VR has seen a huge increase in attention over the past few years due to the commercial release of the first generation VR headsets, where Oculus Rift and HTC Vive are the major products on the market. These headsets use head tracking sensors to determine the head's position, and can then use these coordinates to position your

view in a virtual world. Another feature is the use of two screens, one for each eye, to create a stereo vision effect with sense of depth.

Another concept within the area is Augmented Reality. This technique applies enhancements to the real world, instead of stepping into a virtual world. Microsoft HoloLens is a recent example of this, where all kinds of 3D rendering are applied to real world environment.

There exist articles that describes how Augmented Reality works. Engadget posted an article describing Microsoft's HoloLens, an AR headset, and how it can change the way viewers perceive and interact with Minecraft[6].

Figure 2.1: Interaction in Minecraft through Augmented Reality (where virtual objects augment the real world) using Microsoft HoloLens. Image courtesy of Microsoft Sweden [7].

In between these two there's a concept called Mixed Reality. Mixed Reality introduces the concept of blending real with virtual, where physical real world objects can interact with the virtual objects, creating new ways of interaction. In a sense, Mixed Reality is Augmented Reality within Virtual Reality. An example of this could be cameras capturing the surroundings, allowing the rendering of these surroundings in the virtual world.

Aspects of Mixed Reality can be a part of the gameplay of live-action VR games, as mentioned in a an academic article from Brazil.

"The game may also track physical objects that are part of this physical environment and map them in the virtual world as 3D models. Examples of these physical objects are furniture, interactive objects carried by a player, and players' own bodies. However, the player does not see the physical world and physical objects, the player only sees virtual representations through the HMD.", writes Valente et al.[5]

There has been a lot of excitement regarding VR in the last few years. A VR headsets lets you dive into a 3D environment with a higher level of presence (the perception of truly being in the game world) than what is possible with only a stereoscopic 3D display. With VR headsets the players can not only look, but also move around the

environment which truly makes for an immersive and enhancing experience that is not limited to just games but is also available to other applications such as military training and medicine.

There are challenges with VR, however. Cybersickness (see below) can easily erupt if the headset does not respond correctly to the players' perception of movement. Another challenge is to handle the players input in VR. The mouse and keyboard is not optimal anymore e.g as the players cannot see them, instead new methods such as hand motion detection or VR motion controllers must be explored. The input part is especially challenging in real time strategy games, where the players must be able to do many things at once, and be able to navigate quickly through options and maps.

The VR Head Mounted Display (HMD) used in this project is an Oculus Rift DK2. This version of the headset is not representative of the final product. The Oculus Rift DK2 is solely for the purpose of letting developers get ahead for the actual release of the consumer variant (CV1). The CV1 however has been commercially released during the project, but has not been available for testing. The CV1 has considerably better specifications than the DK2. This means that many, especially negative, observations relating to the HMD could possibly be considerably improved with the new hardware.

The Oculus Rift is essentially a headset consisting of two screens, one for each eye. In front of each screen is a lens, which makes the screen appear larger than it really is. This is to increase the field of vision and allow the eye to focus at a good distance. This increase in field of vision, does however make the screen resolution more critical, as it makes the individual pixels more apparent. This could be compared to staring at a 40" LCD TV from 10 cm away.

Oculus uses a camera to track the position of the HMD to keep the real world head position synchronized with what the players sees. It does this with the help of several infrared LEDs with their relative positions and patterns calculated to determine the relative position of the HMD.

2.2.1 Cybersickness

Cybersickness in virtual reality can be likened to nausea and occurs when the response of the headset does not match the movement of the player's head, especially in games with a first person perspective where the players have the option of not only moving the view with their head but can also control it with an input device like a gamepad or mouse. Other things which can cause cybersickness include acceleration and input lag.

Since game development in VR is relatively new it is also possible that new things might be discovered that cause cybersickness. Finding solutions for these problems while retaining a great sense of immersion is one of the key focuses of this project.

One of the core issues in this project will be to investigate the placement of the players in the game environment and their ability to move around the map. The question is whether the players should be able to navigate a larger map or if the player should be stationary but able to reach and look everywhere on a smaller map. It will be a part of our goal to measure and hopefully minimize the amount of cybersickness in this game.

Road To VR published an analysis on how one can avoid cybersickness in VR games[8].

2.2.2 Graphical User Interface

Since the invention of computing technology, people have had a need for easier interaction with their computers. People have always done this with the help of a User Interface (UI). The User Interface's complexity can range from a label and a lever to a fully text based operating system. Ever since computers made their way into the world we have sought to make computers more accessible to common people.

At first, computers were fully text-based, meaning that people only wrote commands in a prompt and the computer executed those commands. This was effective, but not very user friendly, since it required the user to know the commands to be able to use the computer.

Later, during the late 1970's a company named Xerox made the first Graphical User Interface or GUI. A GUI is a way for the user to graphically interact with the computer, making it much easier to use the hardware without education on the system. It was not however until the coming of the Apple Macintosh that GUIs became popular. One reason why the Macintosh became so popular was because of its GUI and therefore ease of use for the public[9].

Over the years developers and end users alike have refined the term GUI. They've made it easier to understand, optimizing the way it looks and the information it presents and so on. Some go through extreme effort to make their GUI tailored to the way they want to use it, resulting in GUIs others couldn't possibly hope to understand let alone use. One thing however has always been the same, the GUI is projected on a 2D monitor. Having a fixed frame of where the GUI would go, and bounds that the GUI would play inside.

Now, with the VR revolution (As predicted by Motherboard[26] there will be 7-16 Million users that buy a HMD), developers need to rethink the term GUI and a new standard needs to be set. As LeapMotion writes in their blog, GUIs are going to change[25]. Now that the bounds are removed, we're probably going away from a 2D type of GUI. The 2D world is limited to what you can do and the 3D benefits from the possibilities that VR enables. A few examples of this would be Google's Tilt Brush[24] where the GUI is bound to the players left hand and the player is able

to summon the GUI on command. Another example is Fantastic Contraption[23] by Northway Games where as little GUI as it is, it is still bound to the player's hands.

2.3 Input

Input and interaction is a core problem when dealing with Virtual Games, but in Mixed Reality there are ways to alleviate the problems.

"Having Agency is the ability to act in any given environment. It's the feeling of having power and ability to manipulate the things around you. Without it, you have the Swayze Effect which is an issue that 360° Video Storytelling is still dealing with. As I've mentioned before, your hands are one of the most important things you need to bring into VR with you. Why? Because you'll spend more time playing with balloons than you would've ever imagined.", writes Azad in an article on Virtual Reality interaction[10][11]

When it comes to input in VR there are a lot of interesting choices that are both on the market and under development. The two most common options right now are gamepads like the Xbox One controller and motion controllers such as the Razer Hydra, HTC Vive and the upcoming Oculus Touch. Motion controllers are controllers that use different types of sensors for tracking the player's hand motions. The Razer Hydra, for example, uses magnetic fields to determine position and orientation.

A third option is the Leap Motion controller which is a hand tracking camera that can be mounted on the headset. It uses optical sensors and infrared light in order to scan the user's hands and then feeds that data via a USB cable to the computer where a person as a developer can access it through the Leap Motion API (Application Programming Interface) or via game engines that support it.

The Leap Motion controller has a field of view of 150 degrees and has an effective range from 25mm to 600mm in its forward direction[12]. With the beta version of the new driver (Orion) being released, the hand tracking has been elevated to a new level and is now a much more reliable source of input in games and other applications than it was before this project was started.

3

Method

3.1 Game Design

The first idea of the game was to make a game in MR, blending real with virtual. This way, the playfield of the game would actually appear right out of the table that was scanned in, leading to a greater sense of immersion. In order to do this the first plans were to scan a room, with depth sensing cameras such as Microsoft's Kinect, and recreate it virtually within the game world. It was quickly realized, however, that this would most likely be a project on its own and the idea was abandoned. Instead it was decided to make a fixed room in the game, where the aspect of mixed reality would be achieved by the players having a physical table in front of them (typically the desk where the player's computer is).

The gameplay of this game follows the core principles of the RTS genre. The players never wait for their respective turn, instead they always keep playing. This makes for a gameplay where players must react in time to their opponents' moves and plan and execute their actions accordingly.

In most games that employ the RTS formula, the soldiers have no will of their own; the player decides where they go, when to attack or defend as well as upgrading the soldiers.

It would prove to be a little too difficult and cumbersome to steer the soldiers as precisely as one normally does in an RTS game, in VR. In order to make the controlling experience smooth, it was decided to make use of flagpoles. Each of these flagpoles have their own squad of soldiers. When a flagpole is grabbed and moved across the playing field, the squad that belongs to that flag will move to its location. In other words, the idea is that the soldiers would be intelligent and move to the flagpole, attack and seek cover automatically on their own.

Initially there were plans for having an artificial intelligence acting as another player to be the human player's opponent. It was decided early that this was not interesting and could also take too much time to implement. Instead, the game is supposed to be played online versus other human players. This way, it is also easier to make sure that both players get the same or similar VR experience, since it is very difficult to drive two VR headsets on the same PC hardware through local multiplayer. Handling the network code for the multiplayer feature was done within the game engine.

3.2 Unreal Engine 4

This game was created using the game engine Unreal Engine 4, made by Epic Games. All logic in the game was implemented using this engine and its feature called "Blueprints". The Blueprints Visual Scripting language is a visual way of programming, which requires no code to be written. Instead, the user creates functions and commands represented as boxes and then connects these boxes by dragging lines between them. This makes it easy for beginner developers to create a project in the engine, although it is still necessary to have some programming experience to fully understand what the commands do and how to properly connect them. [13]

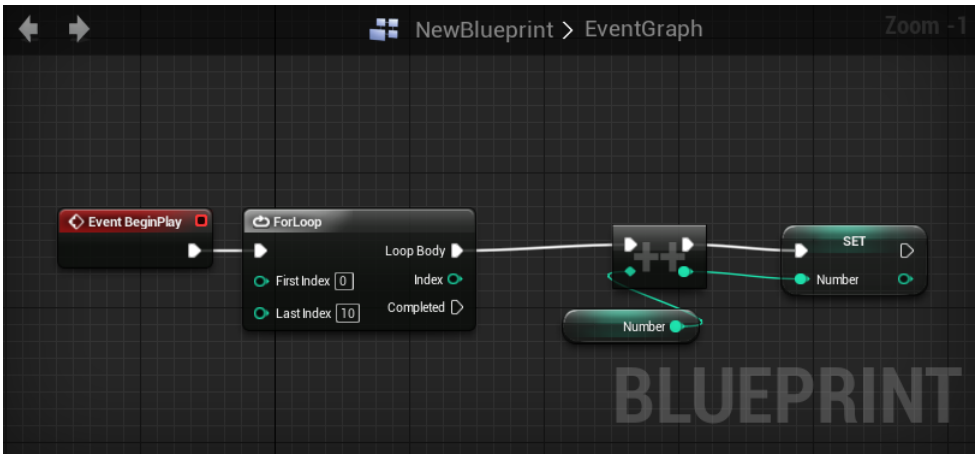

Figure 3.1: Figure shows a loop in blueprints that increments an integer eleven times.

Since no one in the project group had any previous experience of Unreal Engine 4, reading the documentation and tutorials gave much useful information on how the project should be implemented.

It is possible to make the game entirely with programming code. In Unreal Engine, the programming language one has to use is `C++`, unless one wants to use the blueprint approach. Blueprints can easily be combined with `C++`, which is preferable in some cases where one has to implement functions with heavy calculations like AI

and physics, as blueprint can be less efficient than pure code. In our case this has not become a performance issue as of yet, but if it does we will have to port some of the blueprints scripts to C++.

When it comes to Leap Motion in Unreal Engine we used a plugin originally made by Mark Weiser. However the development of the plugin was taken over by a Unreal community member named Jan Kaniewski, also known by his username "Getnamo", whom made the plugin more event driven. The plugin was an unofficial plugin in version 4.10 since Epic and Leap Motion had tried making an official one but it was put on hold due to many bugs. Getnamo's plugin has now become the official Leap Motion plugin as of version 4.11 of Unreal Engine and it does a great job exposing the Leap Motion API to Blueprints which makes it easy for developers to get started creating content.

3.3 Artificial Intelligence

Artificial intelligence (AI) is a major part of many games today, especially when it comes to the RTS genre. In this genre, the player himself is often busy coming up with strategies and commanding whole troops to complete certain tasks. As a consequence, the player has no time to tell every single unit exactly where to move, whom to attack or what way is the fastest. Therefore an AI is implemented to take care of such tasks.

The player controlling the army simply orders the soldiers to move to a location and then the AI takes responsibility to move each and every character in the army to the marked location. Because of this, a smart AI is necessary so that every obstacle can be dealt with in the best possible way, whether it is a static object blocking the path or an incoming enemy.

Another use for AI in the RTS genre is a human-like opponent to battle. Many RTS games, such as Starcraft II and Rome Total War II, both offer a chance to play against other human players and to try to battle an AI. In the case of the AI opponent there will be two types of AI in the game. The first one is the helping AI that will help the units make small decisions and the second one is actually making all the strategy decisions and acting as the foe.

3.4 Artificial Intelligence design in Unreal Engine 4

The method used to design the decision making AIs in Unreal Engine 4 is called behaviour trees.

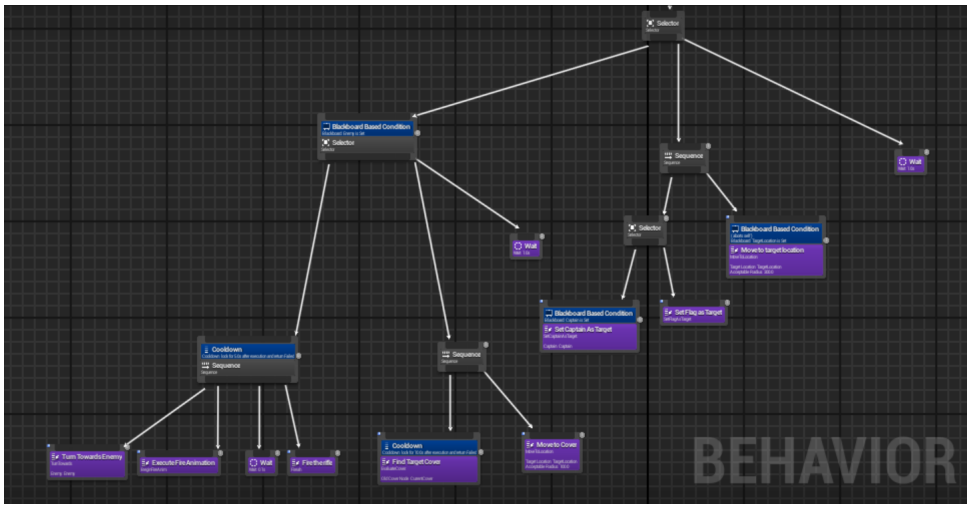

Figure 3.2: The AI behaviour tree for the Musketeer unit type.

The character AI is designed to control pawn characters throughout the game. The AI is running a specific behaviour tree that will let the pawn react to and handle certain events. The behaviour tree is constantly running and is going through the tree until it runs out of branches to follow; Then it starts over from the top once again.

By creating branches and adding conditions that must be met to get access to a certain branch, the player is able to guide the soldiers through the behaviour tree by triggering different events and giving specific inputs. For instance in this project the soldiers will be following a squad captain. The behaviour tree is waiting for the captain to start moving and will then use an algorithm to follow the captain. The captain is in turn waiting for the location of the flag to change so that he can start moving to his next target. All the player has to do is to move the flag and suddenly a whole squad are on their feet moving across the table. By moving the flag, the squad captain unlocks a branch that will tell him to start moving to the location of the flag. This in turn will open up the branch to move for all other members of the squad. By using this type of AI the developer can give the soldier the ability to detect a range of different predetermined scenarios and react differently to each and every one of them.

Another important aspect of any RTS game is pathfinding. Since the player most of the time does not want to bother telling every unit exactly which path to take to get to a target, a smart pathfinding AI must be implemented. Unreal Engine 4 has a

"move to target/move to location" function that allows a character to move directly to a given target or location. This function results in very simple movement, the character will calculate a path and move in a straight line towards the given point. When objects or characters block the calculated path the result is a collision between the moving character and the obstacle. The algorithm can not handle changes in the calculated path. For the "move to target" function to work there needs to be a navigation mesh present on the map. The navigation mesh is used to mark the parts of the map that will allow characters to move on. By using a navigation mesh the characters will know where they are allowed to move and where they can not go, i.e. on top of the mountain that exists on the current map. Unreal Engine 4 uses a simple object to automatically generate the navigation mesh, which can then be modified by the programmer.

3.5 Input

For this project it was decided that a Leap Motion controller would be the most interesting and most immersive source of input to develop for. It was also one of the cheapest options which made it pretty simple for us to convince the University to buy one. In the discussion section this choice will be motivated in more detail. Support for mouse and keyboard controls were implemented for development purposes since we only had access to one HMD and one Leap Motion controller.

After the decision was made that Leap Motion would be used it was time to start looking at what kind of support Unreal Engine had for it. It turned out that there were two existing plugins for the Leap Motion controller, one official and one unofficial. The official plugin was integrated in the engine and it was developed by people at Leap Motion and Epic. So, obviously, this sounded like the best option and experiments with it started.

However the HMD that was to be used in this project had not been acquired yet so testing out the plugin in the beginning was done in a non-VR environment resulting in a false belief that everything worked fine. When the HMD finally arrived and the same tests were run in VR mode a lot of issues arose that had to do with orientation.

It turned out, after reading in the Unreal forums in more detail, that the built in plugin had severe flaws when it came to VR and that the issues were very hard to track down, in fact the whole development of the plugin had been completely abandoned.

One of the reasons that development had stopped was due to the success of the unofficial plugin of integrating support for VR and exposing almost all of the core functionality of the Leap Motion API to blueprints in Unreal Engine which made it that much simpler getting started with development. After downloading the unofficial Leap Motion plugin and seeing how well it worked in VR and how simple it was to use, the process of implementing it in our game started.

3.5.1 Scaling Issues

When the integration of Leap Motion in the project started, an issue that had to do with scaling was encountered, and it would prove to be extremely difficult to find a solution. The problem had to do with a parameter called World To Meters (`w2m`), which is located in world settings within the Unreal Engine editor under the section *VR settings*. This parameter is used in order to scale everything up or down in the world in order to make the player feel smaller or larger in VR. The higher the value of `w2m` the larger will the player feel in the world and vice versa.

Experiments had been made with `w2m` in order to find a value that made it feel like the player sat in front of a real table and could look and reach almost everywhere on the map. However two issues came up when modifying this parameter. The first issue had to with something called Interpupillary Distance (IPD) which is the distance between the eyes and this will be discussed in more detail in the next section 3.6.

The second issue had to do with the Leap Motion plugin. When increasing `w2m` to the amount that was needed, the hands started behaving really strange. The Leap hands started jittering and their movement got delayed a lot. This problem turned out to be a really hard one to solve but at first we tried to avoid it by scaling the world down so `w2m` could be kept at default value.

After scaling everything in the world down to "real life" sizes everything seemed good at first. It felt like one sat behind a real table and the hands felt normal. So now it was time to implement functionality for grabbing and releasing the flag that we had decided would be the way to move your forces around and then connect it with the rest of the game. In the first iteration of this feature one could pick up and drop the flag anywhere on the map and it would fall down with a specified speed and land on the map.

When it had landed, a waypoint was set so the squad of soldiers connected to the flag would be able to find their way there. However when it was time to test this functionality a new issue arose. It appeared that, on the very small scales that we now worked on, the generation of navigation meshes used for pathfinding broke down. There are a lot of parameters one can change in order to tweak the generation of navigation meshes but it appeared our world was way too small to achieve a solid navigation mesh, which meant we had to go back to changing the `w2m` parameter and tackle the strange problems that came with it.

3.5.2 Scaling Issues Solution

The strange issue with the Leap Motion hands had to be solved. As described earlier they got a jitter effect and moved with a great delay, making them impossible to use for interaction in the game. To get a grasp at what might cause this we had

to learn how the plugin worked from the ground up by looking at the source code, which is written in C++. A lot of different changes to the source code was made but the solution came from studying the official plugin that was described earlier. This plugin had a lot of issues with orientation in VR but when increasing w2m the problems with jittering and delayed movement that Getnamo's plugin had, did not show up.

This was interesting and when we looked at the source code for the built in plugin we saw that w2m was taken into account when updating the positions of the hands, which meant that something similar had to be done in the plugin we used. However the architectures of the two plugins differed so tremendously that it took a long time to figure out where w2m needed to be integrated. But eventually after a lot of experimenting and discussing with the creator of the plugin, Getnamo, one part of the problem was solved.

The jittering effect of the hands and the delayed movement problem was fixed with an addition to the function ConvertAndScaleLeapToUE located at LeapInterfaceUtility.cpp in Getnamo's plugin [14]. The code that needed to be added was:

```
float w2m = GWorld->GetWorldSettings()->AWorldSettings::WorldToMeters;
Vector vect = FVector(-leapVector.z * LEAP_TO_UE_SCALE * w2m,
                      leapVector.x * LEAP_TO_UE_SCALE * w2m,
                      leapVector.y * LEAP_TO_UE_SCALE * w2m);
```

This addition to the source code meant that w2m got taken into account when calculating the position of the hands. The function that needed modification handles the translation from the Leap Motion coordinate system to the Unreal Engine coordinate system.

One issue still remained though. The actual size of the hands did not scale along with the changes made to the code. It was expected that the addition of w2m to the function above would also scale the size of the hand meshes. After trying to scale up the hands on various places in the source code and also in the Unreal Engine editor, it was concluded that this was not possibly.

There was no way to change the actual size of the hands in an easy way within Unreal Engine but one could export the original hands, that came with the plugin, to blender and modify them there. So after the hands got exported to blender they were scaled up with the same amount that w2m was increased from its default value. After this the hands were imported back into Unreal Engine and when doing so it is very important to check a box called "Use TOAs Ref Pose" [15]. If this method is not used then a really strange orientation issue will arise that gave us a lot of headaches.

Finally with working hands in VR, the implementation of functionality for the Leap Motion controller could be resumed.

3.5.3 The Flag

The functionality for moving the flag around was implemented as described earlier but had to be refined and adjusted to handle different scenarios such as when the player drags the flag through the map and drops it. To solve this an indicator was implemented so the player always can see where the flag will land when dropped. The indicator is ray casted from the flag onto the map and it is represented as a circle that will blink when the player holds the flag and stop blinking once the flag has landed.

When the flag gets dragged too far below the map the indicator will always be visible so there is no confusion as to where the flag will land.

3.5.4 Leap Motion GUI

Next up was the creation of a GUI system and it was decided that a simple and easy to use GUI was to be created which would at first only handle squad production. The GUI was to be attached to the left hand and only be visible when the player rotates the left hand so that the palm faces the player's face.

At first a system for interactive buttons in VR for Leap Motion had to be implemented from scratch since this did not exist to our knowledge. When creating a GUI in VR for Leap Motion there were a lot of new things to think about compared to ordinary 3D or 2D games. For instance, the player should not be able to click the button from behind and not accidentally press many buttons at once. Also the size of the buttons and the collections of all GUI components would have to be a proper size so the player does not accidentally press buttons due to them being too close to each other.

The issue of the player pressing a button from behind was solved by adding another collision box behind the button so it could easily be decided which way the player interacted with the button from first. The problem with pressing many buttons at once was solved by adding an extra check for the player's index finger, which means that the finger top has to reside within the boundaries of the button and overlap with the collision boxes for the buttons in order for an event to occur. And the last issue of the size of the buttons was solved after a lot of testing. More user testing has to be made in order to decide whether or not we succeeded in making good sizes.

Another important thing when making GUI in VR with Leap Motion is to make the buttons interactive. This was done by giving interactive responses to the user in the form of changing light around the frame of the button and by translating the button back a bit.

In the first iteration the player could only create one type of squad and it was no time restriction on squad creation. Functionality for queueing up production of

the three different squad types was later implemented and also a progress bar that visualized the remaining time of the creation of the selected squad.

3.6 Virtual Reality

Getting started with VR in Unreal Engine 4 is very simple and straightforward. When using Oculus Rift DK2 which was the HMD used for this project it was just plug and play. The issues however, started when integrating VR into our game.

As mentioned in section 3.5 Input, issues with the IPD arose when increasing the `w2m` parameter. The problem was that a "crossed eyed" effect occurred when looking too closely at objects in the game world which made it impossible to get good focus with both eyes opened. Since we want the player to be able to look really close at things in the world this issue had to be investigated further.

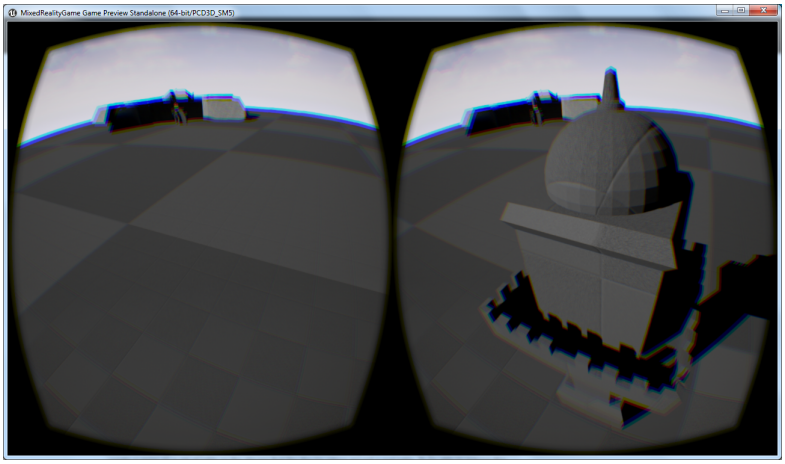

Figure 3.3: Illustration of the cross eye effect issue in the game. To get good focus at the object you have to look with one eye

It was found that the issue arises due to the narrow field of view of the lenses in the HMD which is around 100 degrees. Also the lenses in the HMD are completely static which means that when trying to focus on things that are too close in VR the lenses will not adjust at all in the same way as our eyes do. Humans tend to automatically cross their eyes when looking at objects more closely which means that the area where you can focus on objects with both eyes is much larger. This area is also larger due to the fact that the human eyes have a much larger field of view, around 180 degrees. As illustrated in Figure 3.4 the human eyes have a larger field of view than the lenses of the HMD. The green area in the figure is also more dynamic as mentioned earlier which means it will vary in size.

The following figures are just approximations and are simply used to better illustrate the issue being discussed.

Figure 3.4: Illustration of how the human eyes have a larger field of view. Objects in the yellow area can only be seen with good focus when looking with one eye. Objects in the green area can be looked with both eyes and have good focus. In the red area objects cannot be seen at all

Figure 3.5: Illustration of the narrow field of view that the lenses in the HMD has.

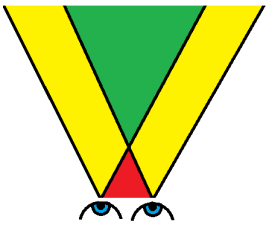

Figure 3.6: Illustration of how by decreasing the IPD, one can enlarge the green area where objects can be looked at with both eyes in good focus

So the reason we cannot look as close as we want in VR, in comparison to real life, is that the lenses of the HMD have a much more narrow field of view then the human eyes. They are also completely static which further limits the area of focus with two eyes.

To solve this problem with the "crossed eyed" effect there are a few things one could do. The first thing that was tried was to lower the IPD. By lowering the IPD the area in which one could focus on objects with both eyes gets larger meaning one could look more closely at them. But the effect of lowering the IPD is that everything will appear larger for the player. This is due to the fact that a decrease of the IPD means that the player's eyes gets closer to each other meaning the player feels smaller. The result of this is a weird sense of scale since the movement of the player was still related to a "large" player while the player felt small. However one could now look

really close at objects in the world with both eyes but the weird sense of scale was not something we wanted. So a more flexible solution needed to be implemented and we came up with two ideas.

The first solution was to specify a region around the map and when the players' stuck their heads inside the region the IPD would instantly change to a lower value. However this method was quickly dismissed since the immediate change of IPD made all people who tried it feel unease or cybersick.

The second solution was to change the IPD dynamically depending on how close the player got to objects in the world. The way it was done was by ray casting from the HMD world location in its forward direction and then measure the length of the ray cast between its impact point and the HMD location. Then an interval was specified with a maximum and minimum range in where the IPD should be calculated with a linear interpolation function. So the closer the player got to a world object the lower the IPD would become if the player is inside the specified interval. This solution gave us the best of both worlds. Players' could now look really close at objects with good focus and when looking away the sense of the right scale and depth reappeared.

3.6.1 GUI in Virtual Reality

The VR-GUI used within this project was developed over a series of iterations. Evaluating what was working and not working shaped the GUI into the final version. The first few iterations the GUI was more like a control panel with buttons and joysticks that moved with the player, providing the player with all necessary tools for movement, communication and unit production. When this was more of an 2D solution kind of GUI it was quickly developed further, disconnecting the player from the GUI.

This was made possible with moving the GUI into a room the player was placed in, and then placing the menus onto banners that were placed around the room and displaying information on painting billboards on the walls of the room. A chat consisting of preset messages and receivers were placed on a banner and the chatlog on a wall painting, this was however changed in a later iteration.

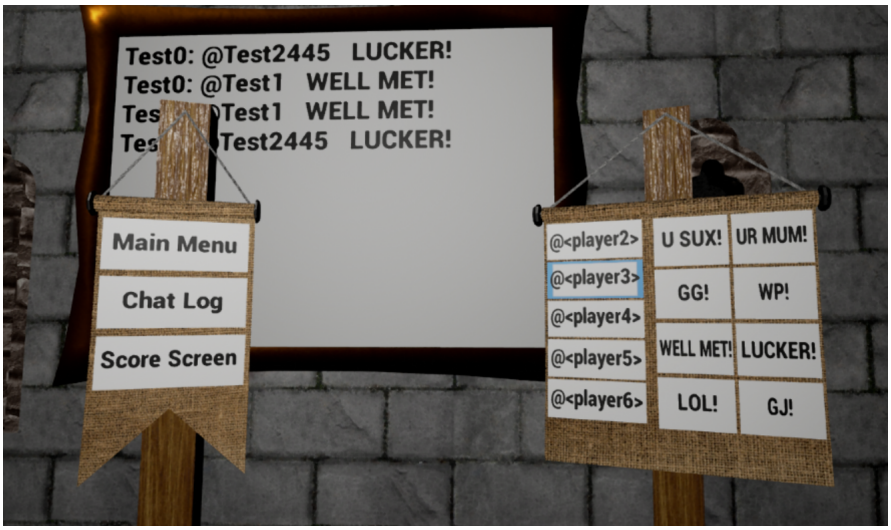

Figure 3.7: Figure showing the main menu and chat banner, as well as the chatlog displayed on monitor in the background.

3.7 User Testing

To investigate what causes cybersickness in our game some user tests were done. Since there was a lot of technical issues surrounding VR and Leap Motion this resulted in that a lot of time that was initially planned for a more comprehensive user test session had to be postponed. However the tests that were made was done mostly during a demonstration day where a lot of people tried our game. The different solutions for the IPD issue was tried and people reported how they felt. It was concluded that the dynamic IPD change would cause feelings of cybersickness for some people. A few of the people who felt unease with the IPD change had never tried VR before which means they might not be the best subjects for testing. Other people who felt unease had previous experiences of motion sickness , from car driving and likewise, which means they too might not be the best test subjects. The reason these people might not be the optimal testers is due to that they might feel unease in VR for almost anything.

Even though some tests were made, a lot more has to be done before we can fully conclude if changing the IPD should be avoided completely.

3.8 Modelling and Animations

3.8.1 Blender

The models (troopers, buildings and the map) and animations for meshes were not created within Unreal Engine but instead a 3D-modelling software called Blender was used. Blender is completely free and is suitable for professional modelling and animation.

The program has the capability to create meshes and to texture them by using 2D maps, called UV-maps to project textures on 3D-models. It can also create skeletal rigs that can be attached to meshes and moved around. Using this technique animations can be created for games or movies.

3.8.2 Sculptris

Sculptris is a freeware created by Pixologic, and is a lightweight sculpting software that has few features but it is good at brush modelling. Sculptris was used to create organic models for the character models.

3.8.3 Modelling of the units

The model's base shape was first created within Blender using box-modelling and subsequently brought into Sculptris to be sculpted into a human character. Sculptris was used to achieve organic details with its dynamic mesh algorithm that adds new polygons where needed to achieve mesh deformations. The model was thereafter brought back into Blender to be optimized. This was needed to make the mesh less demanding on the computer hardware. The mesh was thereafter given an UV map to apply textures on the model. UV-map is named after its axes U and V, by convention. With a human mesh completed, unit differentiation was done by adding different clothing and apparel to the mesh. There were no options for this project when choosing tools to use in the modeling process. The reason for this was because of our previous experience with Blender and Sculptris.

Figure 3.8: Each of the units displayed side by side. From the left, the musketeer, the soldier and the artillerist.

Figure 3.9: The deployed version of the mortar launcher

Each character was then uploaded to a studio called Mixamo where each model was automatically skinned. The mesh had to be skinned to skeleton bones before it could be animated. Mixamo has a large amount of available animations, and it was possible to find almost all of the needed animations. The models and skeleton animations from Mixamo are too down scaled to be used in Unreal Engine, so scaling up has to be done inside Blender.

4

Result

4.1 Game Design

The basic goal of our game is for the players to move their soldiers into the opposing player's fortress. Once the soldiers are in place a progress bar will indicate how much of the fortress has been conquered. After some time, when the progress bar has been filled, the fortress has been conquered and the player wins.

When starting out, the players have limited resources needed to deploy new squads of troopers or artillery units. Throughout the game there will be resource points that the players can capture, in order to procedurally gain more resources.

The purpose of this setup is to implement strategic elements to the game, where players will have to defend their captured resource points or position their squads to hinder the opposing players.

4.1.1 The Different Types of Soldiers

The battles in the game are fought by three different types of soldiers. The mid range musketeers, the close range swordsmen and the long range mortar launcher. These types of soldiers have different advantages and disadvantages to each other.

The Musketeers are the mid range soldier class. Their armor is not very thick putting them at a clear disadvantage against swordsmen at close range. Their main benefit is their ability to fire their rifles, killing off enemy units from a mid range distance. The catch, however, is that the musketeers will not always hit their target. A unique trait to the musketeers is their ability to seek cover when under attack, making them less vulnerable to enemy fire while still able to return fire towards the enemy.

The Swordsmen are the close range units. They sport thicker armor than the musketeers but they can only attack at close range. Swordsmen do, in contrast to the musketeers, always hit their targets. As long as the swordsmen can get up close

to the enemy, they are a very reliable battle force.

The Mortar Launcher is the heavy artillery unit capable of long range attacks. When their projectiles hit their mark they explode, hitting any enemy units within the blast radius. The drawback of the mortar launcher is its slow firing rate and low defensive capabilities. The mortar is operated by four artillerists. When all of the artillerists are killed, the mortar will be rendered useless and be destroyed.

The development of the mortar demanded a solution for the firing arc of the mortar. Even though an "animation version" (with a set explosion point and only animation to show the projectile fly) was considered, the final implementation came with the fairly complicated task of calculating the force which a projectile should receive and be fired in a set angle to hit a target location.

The calculation of the force (F) was made with the following equation that was derived from Newton's laws.

$$F = \sqrt{\frac{dg}{2sin(\theta)cos(\theta)}} * sin(2\theta)$$

Where g is the gravity constant, d is the distance to the target point, where the cannonball will land, from the mortar and θ is the angle between the barrel and the ground.

4.1.2 Armor and the Strategy of Battles

Every class of soldier wear armor but no two classes of soldiers wears the same armor. While they have the ability to fire from a distance, the musketeers are very susceptible to damage. They easily fall in battle if they are hit by any attack. Getting musketeers to hide behind cover and keep a distance from the enemy is the key to using this class of soldiers.

The swordsmen may require to be up close with their targets but they wear a more durable armor making them more resistant to attacks. If a swordsman can get close enough to take a swing at a musketeer, the musketeer will certainly meet his end.

While being next to indestructible by the weapons of the musketeers and the swordsmen, the mortar still needs to be operated by artillerists who do not wear very protective armor. Their main defense is what the mortar provides. The best way to take out an enemy mortar would be to send in a squad of swordsmen to dispose of the artillerists or to blast it with a mortar of one's own.

4.1.3 Resources

Resources are what determine how many and what kind of squads the player can deploy in battle. The more resources the player has, the more and stronger squads the player can deploy. When the player runs out of resources the player will not be able to deploy any squad until the resources have been refilled.

4.1.4 The Fortress

This is where the player's troops will be spawned into the battlefield. The fortress is the player's most important building and must be defended from the opposing player's forces. When the troopers of an opposing force enters a fortress a progress bar will tick down, indicating how much of the fortress that has been taken over. Once the bar has reached zero, the fortress has been completely conquered by the opposing force and the opposing force wins the game. Keeping the enemy from the player's fortress, while advancing into the enemy's fortress, is the most important factor of this game.

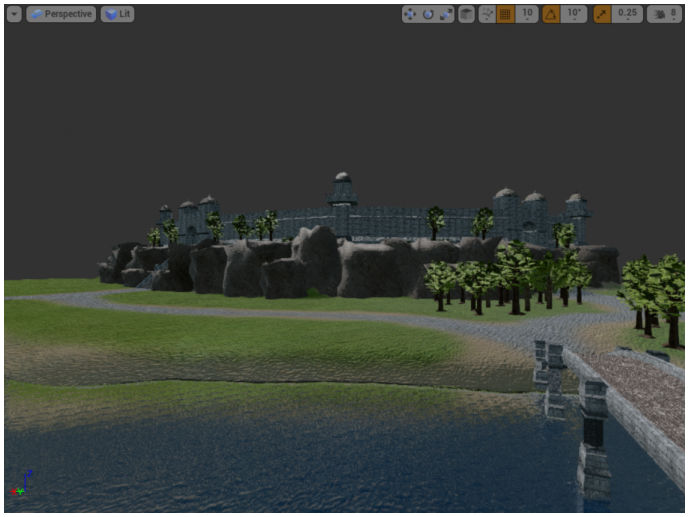

Figure 4.1: The player's fortress.

4.1.5 The Resource Points

Spread across the battlefield are several resource points that can be captured by any of the players in order to increase the amount of resources gained over time. Strategically choosing which resource points to capture will ensure a better income of resources, giving the player the possibility to spawn more troops and overpower the enemy.

Figure 4.2: Resource point with a capture sphere. When a trooper enters a capture sphere, the process of taking over a resource point or fortress begins.

4.1.6 Artificial Intelligence

There are at this point two major AI parts implemented and running, a pathfinding AI and some different versions of a decision making AI. The pathfinding functionality in Unreal Engine 4 was used as the first AI. This made it possible to focus more on the second major AI, the decision making AI.

Each type of unit shares the same core functionality such as choosing a captain and moving to their flag. In addition each character uses their own functions for attacking and in the case of the musketeers, finding and choosing cover. These functions are given a priority, a condition to be called and are placed in the behaviour tree. When, for example, evaluating cover, the musketeers take into consideration the position of all seen enemy soldiers as well as the distance to the cover. The position of the enemies is used to evaluate which cover is the most optimal. If no cover is in a specific range, the condition to find cover is not met and the function will not be called.

4.2 Virtual Reality

To create a sense of immersion it was decided to put the player within a virtual room. This room would contain a table where the game field is set up as a board game for the player akin to a game of chess. This decision was made partly to prevent cybersickness, since a lot of times games can be quite chaotic and this might be a reason for the emerging of cybersickness and also to provide a 1:1 mapped method of moving around the game world as needed.

As of now we have two alternatives for the IPD issue first introduced in the method section (chapter 3). The first one is to keep the IPD at a precalculated value, either the default or one that is significantly lower. The other option is to change it dynamically with the method described in section 3.6.

The Leap Motion hands work well and as intended in VR after the scaling issue was solved as described eariler in section 3.6.

When it comes to cybersickness, which was one of the main problems to be addressed in this project, it was concluded that in situations where the player only moves 1:1 with the movement of the head, there was little to no cybersickness for any of our testers. One of the only situations in which the application makes our testers sick is when the frame rate drops to below about 60 fps. In this case, the screen stutters forward, which causes a very high strain on the eyes, and might cause heavy cybersickness. This is, however, not a problem as long as you run the application on hardware fit for it (mainly with a sufficient GPU). It was also discovered that certain guidelines existed in order to achieve better frame rate. For instance, dynamic shadows shouldn't be used at all and static shadows or light maps should be used instead[20].

Another issue had to do with the dynamic change of IPD which made some testers experience cybersickness. But since this was implemented at the end of the project more testing will have to be made in order to decide whether to remove this feature or keep it as an option. The reason it might cause cybersickness is due to the fact that a decrease in IPD means shrinking the player's head down while the movement still relates to a larger scaled player. [21]

4.3 Input

In the end, after dealing with all the issues described earlier with VR and Leap Motion, there are now custom made hands in the form of medieval looking gauntlets in correct size.

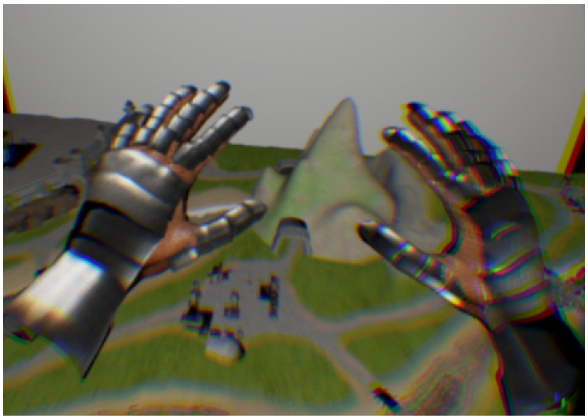

Figure 4.3: The custom made hands.

There is a basic GUI attached to the left hand that gets shown when the user faces the left palm towards themselves. Right now the GUI only handles squad production which means the player can easily queue up a series of squads that shall be produced if the player has enough gold and then a progress bar will start loading that indicates how long it will take to produce the selected squad. The GUI will be expanded greatly in the future and a general system for interactive buttons and progress bars has been implemented which will make expanding it easier.

Figure 4.4: The GUI used for production of squads. The icons below the progress bar represent the different squad types. The icon to the right of the progress bar is the current squad being in production. The icons above are the ones in the production queue.

A general system for interacting with objects has been implemented. For now it is only used for the flag and the target ball that is connected to the mortars, but it will be easy to expand the system for more interactable objects.

For now the system of interaction supports grabbing objects with pinch events, moving them wherever the player wants and releasing them. On release they will fall down with a specified speed until they land on the visual indication point on the map.

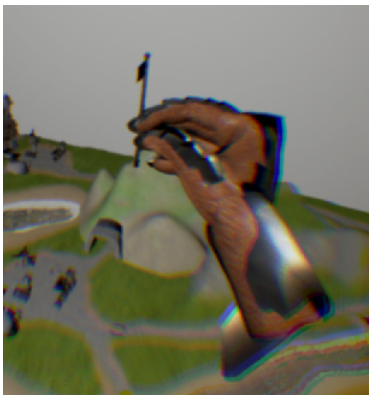

Figure 4.5: Here is a picture of the player grabbing and holding a flag that is used for commanding the troops. Directly below the flag is the indicator showing where the flag will land if released.

5

Discussion

5.1 Input

When deciding what kind of input would be best suited for an RTS game in Virtual Reality, there were many options. The first option considered was a regular gamepad like the Xbox One controller, however it was quickly decided that this option would be a pretty boring and not so unique solution to the input problem. A more suitable solution for VR was needed and the thoughts started heading towards some sort of motion controller in the likes of the upcoming Oculus Touch. Since the Oculus Touch wouldn't be launched until the autumn of 2016 we had to look towards alternatives and quickly found that one of the more popular motion controllers was the Razer Hydra. However the Razer Hydra was too expensive and it appeared not to be as precise as we had hoped. In the end we decided to go with a solution that is both interesting and very immersive, the Leap Motion controller. The Leap Motion is an IR camera that is attached to a head mounted display and then tracks the player's hands to create a virtual representation of them inside the virtual environment. This gives the player a very immersive feeling as they can see their hands and interact more naturally with the virtual world. For our game this felt like the most optimal and unique solution.

Another option that would have been great was the HTC Vive, however due to the late release date it was not an alternative. We will have support for it as soon as we get our hands on the hardware. Since HTC Vive is roomscale VR, which means that a larger tracking space exists for the HMD due to an additional tracking camera, it would be a perfect match for our game.

There has been problems with the Leap Motion controller. The initial creation of the models made them too small making it impossible for the troopers to navigate the map. Scaling them up resulted in the hands being too small in comparison to the rest of the models and the map objects. When scaling up the hands we encountered a problem with the handling which was discussed in section 3.5. There was a rubber band effect where the hands would not correctly follow the movement of the Head Mounted Display. The hands also twisted and turned and locked into awkward positions making it impossible to control the game. In order to solve this we had to modify the plugin software for the Leap Motion controller and scale up

the hand models in Blender. This task proved arduous and consumed a vast amount of time, but in the end we succeeded in making the hands the right size.

Using the Leap Motion controller with Unreal Engine 4 proved to be not as straightforward as we first thought simply due to the nature of the game we were making. Since no one before us seemed to have done anything similar to what we wanted to achieve, it was inevitable that issues would arise.

An issue which has not been resolved with the Leap Motion is the decline in tracking when moving hands into non-optimal angles of the camera. It is, however, expected to be improved with future software improvements from the manufacturer.

5.1.1 Leap Motion User Interface

When implementing the Leap Motion GUI there were some guidelines that were followed. As mentioned in [17] on page 23 it is really important to make buttons appear interactive and give proper response when the user interacts with them in order to achieve a sense of feedback. They should also be in proper sizes and have a certain distance from each other so that the user does not accidentally press more buttons than intended. This was all taken into consideration when the implementation started. Something that was not added during the time of the project was sound response when pressing buttons but this will be implemented in the near future as it is of great importance when it comes to giving proper response.

Another thing that has been discussed and that will most likely be implemented in the future is to enable the player to disconnect the GUI from the left hand and place it wherever the player wants it. This implementation was inspired by a Youtube video. [18] It is a very flexible system that is completely optional for the player but can provide a greater sense of interaction.

Before it was decided to have the GUI attached to the hand other options were considered. For instance a control panel in front of the table where the player could handle all the necessary interactions for playing the game. It was also talked about placing the GUI for the squad production above the player's fortress. This will, however, be optional for the player if we choose to go with the detachable GUI method.

The reason for not choosing the control panel approach was because having a GUI attached to the player's hand felt like a more compact solution. It also becomes more dynamic with the player being able to easily access the GUI from anywhere in the virtual room and always from a close range.

One issue that is well known is the so called Swayze effect which has to do with the player's ability to go through obstacles in the virtual world. When getting hands in the virtual environment, questions arise regarding how this effect should be handled with regards to user interfaces. When it comes to buttons one could implement

them in such a way that the player cannot press through them and the buttons will follow along with the finger pressing them.

However according to Mike Alger [19] most people do not have an issue with being able to move their hands through virtual objects. He also made buttons with a water like color since it might make people associate the virtual button with water which might ease the experience of being able to press through objects in the virtual environment with hands.

In our game the problem of the Swayze effect is partly solved with the virtual table being 1:1 mapped to a real table but for the user interfaces we allow the player to press through buttons and other virtual objects that are not mapped to real world objects.

5.2 Virtual Reality

An issue that was discovered quite early in the project, but was paid more attention to quite late was the problem with looking closely at things. Normally when looking at things closely, many people cross their eyes to focus. This becomes an issue in VR, since the angle of view does not cover that far inwards, which in turn makes it impossible to focus on things closer than about 2dm with both eyes at the same time. This created a very annoying effect for some people where it felt like there was an error with the hardware. However, it was discovered that some people had no issue with this. Si Lumb, BBC Research & Development [16], suggests that testing has shown that older people have lost the reflex of crossing the eyes, for whom it is not an issue. When talking to Si Lumb he says:

"In current VR, focus is always at infinity, so young eyes try to converge on close objects without realising they are in fact an infinite distance away, just represented in stereo. Older eyes can't converge as well, so the effect is less pronounced."

As this technology is new, there is not much research on the topic. A thorough study would need to be conducted to get to the bottom of this. This will be necessary to be able to design around the problem in our application, but has not been within the timeframe of the project. There are however some options to a workaround for this problem. The way we handle it right now is with dynamic change of IPD depending on how close the player gets to objects.

However some people that have tried it experienced a sense of cybersickness while other did not so further testing will have to be made with this method before a decision is made to keep it. But for now the option of dynamic IPD will exist. Other ways that this might be solved is by letting the IPD be at default and then add a magnifying glass that the player can grab somewhere in the virtual room and then use to be able to look closely at objects. Another option we have been discussing is that the player should be able to throw some kind of sphere or character

at the place on the map they want to look closely at and then they will be teleported there and shrunk down to the appropriate size.

It could also be a camera that the player can place on the map and which starts streaming the zoomed in environment to the big screen in front of the table. This has the advantage of not making the player move too much and accidentally smash their HMD in the table. We will implement all these suggestions in the future and make a lot of user tests in order to decide what might be best suited for our game. It might be appropriate to have all solutions to the problem available as options in the game and then the player gets to decide which suits them best.

Something else that has to be properly done is a way of mapping our virtual table to a real table more generally since we cannot assume that all players have the same tables. This can be done with the HTC Vive fairly easily where the user measures the table to be used with their motion controllers. This might be harder with Oculus though since it only comes with one tracking camera and the Oculus Touch motion controllers have not arrived yet. A way to solve this is to let player input the dimensions of the table they aim to play on.

One of the key focuses of this project was to reduce the level of cybersickness for users and one of the ways in which we did that was to give the player a familiar environment inside a room by a table. This idea of a room and table would also open up new possibilities when it came to the UI. In a game in virtual reality it is a difficult task to display an informative UI, without the UI being in the way. A minimap or a scoreboard will take up a lot of space and, since the player now is inside the game instead of just being an observer, these can easily be distracting.

Many of the UI features can now instead be placed on the walls and in other spaces around the room and the player can just turn his head to access a specific feature. Utilising the Leap Motion controller to show information was also an option in which the player would be able to have the information displayed in the palm of one of the hands. The player would simply turn their hand, open their palm and the information would be displayed.

While we do not have physical objects to interact with (i.e. AR cards that would function as flags), the idea is to have a physical table in front of the player that is approximately the same height and size as the virtual table in the game. This gives the player the notion of moving and interacting with a real table in the game, and to have real life objects in the game gives elements of MR. A good question might be why we chose MR instead of AR. AR would probably have provided a better experience, the downside to AR is that it is currently very expensive and not commercially available. To make a game for AR today would mean almost no one got to play it since not many have the gear required.

5.3 Other Possibilities

An early idea was to make use of Microsoft Kinect cameras, capable of sensing depth within an image, in order to bring the player and the encompassing area into the game. The setup would require at least three cameras placed around the player to scan in the player and the table. With this method the players could walk around the physical table and interact with the playing field. In this example, controlling the flags would be done by using special cards that the cameras can scan and interpret as flags in the game. For a more in depth explanation on how room scanning works, although without real time rendering, is to look at a project by researchers at Chalmers where depth streams are used to scan the player into a virtual environment[22].

6

Conclusion

Making an RTS game for VR has been an interesting and challenging task. Many issues arose along the way that took a lot of effort to solve but in the end we got a working product. One of the most challenging problems had to do with our choice of input, Leap Motion. Since no one before us had made an RTS game in VR with Leap Motion as input, the problems got much harder to solve. In the end a solution was found and the Leap Motion hands work very good as a source of input in the game.

The approach taken to solve the issue of user interfaces in VR was to attach a GUI to the left Leap Motion hand and to place interactable banners in the virtual room. A GUI system had to be developed for this which will be expanded in the future.

When it comes to the issues described in our report about IPD, which was a difficult problem to tackle, it has not fully been solved. A reason for this is that the current method being used, where the IPD gets updated dynamically, has various side effects from person to person. As discussed in the previous section a lot of user testing will have to be done in order to decide whether this solution is the best or if one of the other suggestions made in the discussion section will be better suited.

As for the problem of cybersickness we cannot say that no user will ever experience it from our game but a lot of effort has been taken to avoid it as much as possible. Few of the people who have tried our game so far have complained about feeling nauseous. The few who did experience nausea have had a history of motion sickness, for example feeling sick from traveling by car. In our game we tried to avoid and minimize cybersickness by mapping a real table to the virtual table and placing the player in a room that feels familiar and real. Keeping a high frame rate was also important since it was discovered pretty quickly that a low frame rate caused cybersickness for almost all users.

7

Future Plans

At one point in the middle of the project, there was a realization that there might be a commercial value in the product. Since this generation of VR is very new, there are very few applications for the hardware. After some initial market research, it was established that our product would stand decently against current competitors, and that the project was fairly alone in the specific genre of gameplay. Because of this, it was decided to explore the potential commercialisation of the project. Through gaining business contacts through many different sources (closed Slack channels, Chalmers Ventures, Gothia Innovation), we have managed to build a network of information and channels. This gave a lot of very needed information on business design and further progress. A business plan has been formed, and application to join the incubator Gothia Innovations in Gothenburg has been discussed, so the odds of the game becoming commercially available are fairly high.

The game is in a very rough state, where most things are working on a very basic level, but the game has uncompleted features as well as a lot of bugs. The next stage of the development would have to first include completion of features to create a working beta version. After that a big testing phase would have to begin where the game is polished and everything tested thoroughly to prepare for a potential release.

Bibliography

[1] LaViola Jr. J J, (2000), "A Discussion of Cybersickness in Virtual Environments". In: *ACM SIGCHI Bulletin: Volume 32 Issue 1, Jan. 2000* pp. 47-56 [Online] DOI: http://delivery.acm.org/10.1145/340000/333344/ p47-la$_v$*iola.pdf?ip* = 129.16.184.203*id* = 333344*acc* = $ACTIVE\%20SERVICEkey = 74F7687761D7AE37.3C5D6C4574200C81.4D4702B0C3E38B35.4D4702B0C3E38B35CFID = 616119686CFTOKEN = 45645855\&\_acm\_\_\_ = 1463407045_6abb076cbd95380ccc4c4ec174cf35a3URLTOKEN$.

[2] D. Allen, (2015), *Ten Do's and Dont's to improve Comfort in VR.* Available: http://www.blockinterval.com/project-updates/2015/10/16/ten-ways-to-improve-comfort-in-vr.

[3] M. Buro, D. Churchill (2012), "Real-Time Strategy Game Competitions". In: *AI MAGAZINE* pp. 106-08. ISSN 0738-4602 DOI: http://www.aaai.org/ojs/index.php/aimagazine/article/download/2419/2317.

[4] Blizzard Entertainment, INC. (2016) *What is StarCraft II?* Available: http://eu.battle.net/sc2/en/game/guide/whats-sc2.

[5] L. Valente, E. Clua, A. Ribeiro, S. Bruno Feijó (2015),"Live-action Virtual Reality Games", ISSN 0103-9741. DOI: https://arxiv.org/ftp/arxiv/papers/1601/1601.01645.pdf

[6] N. Summers, (2015) *I played 'Minecraft' with Microsoft's HoloLens.* Available: http://www.engadget.com/2015/07/08/minecraft-hololens-minecon/

[7] Microsoft Sweden (2015) *win10-HoloLens-Minecraft* Available: https://www.flickr.com/photos/microsoftsweden/15716942894/

[8] S. Hayden, (2016), *7 Ways to Move Users Around in VR Without Making Them Sick* Available: http://www.roadtovr.com/7-ways-move-users-around-vr-without-making-sick/

[9] E. Raymond and R. Landley, (2004), "The first GUIs, History: A Brief History of User Interfaces". In *The Art of Unix Usability*, Catb.org, 2016. [Online]. Available: http://www.catb.org/esr/writings/taouu/html/index.html [Accessed: 2016]

[10] A. Balabanian, (2016), *Cause and Effect-VR's Essential Interaction*, Available: https://medium.com/@WizardofAz/cause-effect-vr-s-essential-interaction-efff0471b470.7ix2encm2

[11] M. Burdette, (2015), *The Swayze Effect*
Available: https://storystudio.oculus.com/en-us/blog/the-swayze-effect/

[12] Leap Motion Documentation, (2016), "API Overview" [Online].
Available: https://developer.leapmotion.com/
documentation/cpp/devguide/Leap$_O$*verview.html*

[13] Epic Games, (2016), "Blueprints Technical Guide"
Available: https://docs.unrealengine.com/latest/INT/
Engine/Blueprints/TechnicalGuide/index.html.

[14] M.Weiser and J.Kaniewski (Getnamo), (2014), "An event-driven Leap Motion plugin for the Unreal Engine 4". Available: https://github.com/getnamo/leap-ue4/blob/master/Plugins/LeapMotion/Source/LeapMotion/Private/LeapInterfaceUtility.cpp

[15] Epic Games, (2016), *FBX Import Options Reference* Available: https://docs.unrealengine.com/latest/INT/Engine/Content/FBX/ImportOptions/

[16] S. Lumb, (2016), BBC Research & Development, [Personal Communication]

[17] Leap Motion, Inc. (2015), *VR Best Practices Guidelines* [Online] Version 1.2
DOI: https://developer.leapmotion.com/
assets/Leap%20Motion%20VR%20Best%20Practices%20Guidelines.pdf

[18] HOCgaming, (2016), Sword Art Online GUI - V2! [VR Dev Log - 05], Available: https://www.youtube.com/watch?v=d70Sc0IzKm0

[19] M. Alger, (2015), *VR Interface Design Pre-Visualisation Methods*, Availabe: https://www.youtube.com/watch?v=id86HeV-Vb8

[20] Epic Games, (2016), *Virtual Reality Best Practices* Available: https://docs.unrealengine.com/latest/INT/Platforms/VR/ContentSetup/

[21] Epic Games, (2016), *Virtual Reality Best Practices* [Online]
Available: https://docs.unrealengine.com/latest/INT/Platforms/VR/ContentSetup/

[22] V. Kämpe, S. Rasmuson, M. Billeter, E. Sintorn and U. Assarsson (2016). *Exploiting Coherence in Time-Varying Voxel Data* [Online]
Available: https://www.youtube.com/watch?v=ptmHag3XwXY&feature=youtu.be

[23] Fantastic Contraption, Northway games, (2016) Available: http://fantasticcontraption.com/

[24] Tilt Brush , Google, (2016) Available: http://www.tiltbrush.com/

[25] LeapMotion Blog , (2016) Available: http://blog.leapmotion.com/vr-interface-design-future-hybrid-reality/

[26] MotherBoard's forecast of sales, (2016) Available: http://motherboard.vice.com/read/how-much-will-oculus-vive-vr-sell

A

Appendix 1

A.1 Individual Contributions

A.1.1 Jimmy Malmer

Responsibilities:
Project Leader. Responsible for project deadlines, planning etc. Involved in almost all large decisions concerning design choices, features, path of the project etc. Scrubmaster-ish role, helping where needed. Developer of large parts of the AI. Main responsible for the marketing.

Problem solving, synopsis, analysis:
Been trying to solve issues where they arise, acting sounding board and giving feedback regarding ideas. Been largely involved in the general VR problems with IPD etc, and discussing the issues outwards to gain information. Had the initial idea of an RTS, and had a big role in the building of the project planning.

Report sections responsibility:
Virtual Reality (Background & Discussion)
Development Process
Future Plans
(Have also done additions in many other parts of the report where needed)

A.1.2 Anders Eriksson

Responsibilities:
Mainly responsible for Leap Motion and VR integration in the project and dealing with all the issues that came with it. Also helped out with various problems that other members encountered.

Problem solving, synopsis, analysis:
Struggled with a lot of issues regarding VR and Leap Motion but managed to get it

working after a lot of trial and error and discussions in the unreal forums. We have also worked together a lot in the group to solve problems that arose by discussions and laying out the problem on a whiteboard.

Report sections responsibility:
Wrote everything to do with input and also a lot about VR and the conclusion.

A.1.3 Kevin Björklund

Responsibilities:
Primarily responsible for still-graphics, ranging from icons, to textures and environment assets such as rocks and walls. Painted the game titelpicture and the poster. Heavily influeced the planning of the project, decisionmaking within the project and shaping the gameplay. Inlearning was made as needed, looking up alot of things regarding graphics.

Problem solving, synopsis, analysis:
Several problems arose around the graphics that was solved. Been around to assist solving questions from everyone and tried to make myself an asset.

Report sections responsibility:
Been writing parts of the project report. Mainly the parts about UI and GUI as well as the problem definition and the abstract.

A.1.4 Christian Roos

Responsibilities:
Mainly focused on the network part of the project and also been helping out with different parts of the developing process. Made the melee unit and made all the other units multiplayer compatible. Mostly trial and error, reading forum posts and watching tutorial videos. A lot of work with blueprints in Unreal Engine 4.

Problem solving, synopsis, analysis:
We have mostly worked together when major problems have been discovered. When minor problems have come our way the forum posts has been the number one source of help.

Report sections responsibility:
Been writing a part of the introduction, writing the RTS part in the background as well as the AI part in both background and results. Been helping out with some of the images taken for the report.

A.1.5 Daniel Olsson

Responsibilities:
Responsible to make sure the animated models were created to be used in testing AI, and as an end-product. The animated characters were: Artillerist for Mortar, Soldier and Musketeer. Also were responsible for the death animation of those, using Ragdoll physics. Created the Particle effects used in combination with the units, gunfire, blood spray and mortar fire explosions. Created the animated storybook which were used for navigation.

Problem solving, synopsis, analysis:
Has been focused on trying to get the units to work as fast as possible so that the AI-guys could test and evaluate their AI with a moving character. Attended all meetings and listened carefully for other peoples problems, and considered if I could help. Has tried to help with any problem arisen with Blender which I've used alot. Coded the animation transitions and events for the soldiers, and the change of physics for the soldiers.

Report sections responsibility:
Wrote the modelling section in the method section, though half of it was later removed. Helped transferring the method section to LateX, found a couple of references. Added picture for Mortar, and Units.

A.1.6 Richard Wecke

Responsibilities:
Assisting project management, involvement in creative design as well as project planning and gameplay elements. Was also a large part of the development of the artificial intelligence (mainly responsible for the Mortar Launcher, Artillerists and partly responsible for the Musketeers and other more general features of the AI). Responsible for the structure of the Final Report. Composed the Main Theme of the game.

Problem solving, synopsis, analysis:
Have been struggling and solving problems, together with other group members, regarding the AI as they arose. Making use of solutions from the Unreal Forums as well as the documentation of the Unreal Engine have been other ways of solving problems with the AI. Have also taken part in discussions of solutions to other problems within the project.

Report sections responsibility:
Responsible for the structure of the final report. Also wrote about the Game Design and partly about the Unreal Engine as well as cleaning up, or filling out, some text when necessary. Mainly responsible for bringing the final text into LaTeX.