Western Norway University of Applied Sciences

Faculty of engineering and science

Department of Computer Science, Electrical Engineering and Mathematical Sciences

BACHELOR'S THESIS

Visual Stimulation for fMRI Scanning Using VR Technology

Arja Sivapiragasam

Enah Lanto

Department of Computer Science/Faculty of Engineering and Science/Computing & Information Technology BSc

Carsten Helgesen

2nd of June 2020

Western Norway University of Applied Sciences

Faculty of engineering and science

Department of Computer Science, Electrical Engineering and Mathematical Sciences

TITTELSIDE FOR HOVEDPROSJEKT

Rapportens tittel: Visuell stimuli for fMRI skanning ved bruk av VR teknologi Visual Stimulation for fMRI Scanning Using VR Technology	*Dato:* *2.juni 2020*
Forfatter(e): Arja Sivapiragasam Enah Lanto	*Antall sider u/vedlegg:* *43* *Antall sider vedlegg:* *3*
Studieretning: Dataingeniør og Informasjonsteknologi	*Antall disketter/CD-er:*
Kontaktperson ved studieretning: Carsten Helgesen	*Gradering:* (Velg: Ingen eller Begrenset til ddmmåå)
Merknader:	

Oppdragsgiver: NordicNeuroLab	*Oppdragsgivers referanse:*
Oppdragsgivers kontaktperson: Stian Scisly Sagevik	*Telefon:*

Sammendrag:

Bacheloroppgaven går ut på å opprette en VR programvare for hardware utviklet av NordicNeuroLab. Programvaren skal vise potensialet til systemet til den utviklede hardware. Programvaren skal fungere som en visuell stimuli for å aktivere hjerneaktivitet under fMRI skanning.

This bachelor thesis focuses on creating a VR application for the hardware manufactured by NordicNeuroLab. The application will show the potential of the developed hardware. This application is utilised as a visual stimuli to activate brain activity during fMRI scanning. The application can be used as a template for further development.

Stikkord:

fMRI	Visual stimulation	Virtual Reality

Høgskulen på Vestlandet, Fakultet for ingeniør- og naturvitskap

Postadresse: Postboks 7030, 5020 BERGEN Besøksadresse: Inndalsveien 28, Bergen

Tlf. 55 58 75 00 Fax 55 58 77 90 E-post: post@hvl.no Hjemmeside: http://www.hvl.no

PREFACE

Our sincerest thank you to Stian, Øyvind and everyone at NordicNeuroLab for giving us insight into the work you do and trusting us with this project.

We also want to give a special thanks to our supervisor Carsten, for guiding and encouraging us through this process.

And lastly, we have to mention our family for always being supportive and helping us stay motivated while completing our final thesis.

TABLE OF CONTENT

1. INTRODUCTION

1.1. Motivation and goal

Functional Magnetic Resonance Imaging, or fMRI, is a non-invasive technique to study brain activity. This technique can be useful in the diagnosis and monitoring of various neurological disorders such as stroke, brain tumors, pre-surgical planning and can also provide valuable information when researching brain connectivity (Glover, 2011). Visual Systems HD is a solution developed by the company NordicNeuroLab (NNL) to present visual stimuli during MRI scanning through special video goggles. The only visual stimulation provided for these fMRI goggles today are simple 2D images and text-based tasks for the test subject to solve while undergoing an fMRI scan.

The goal for this project is to create a 3D/VR program which can be integrated with the already existing solution Visual Systems HD. By creating a 3D/VR environment which mimics real-life interactions with objects and their surroundings, the test subject can become more immersed in tasks, which will provide more realistic and accurate results and findings (Bohil et al., 2011).

This project is conducted in regards to a bachelor thesis at the Western Norway University of Applied Sciences. The project was assigned to us by NordicNeuroLab (NNL) who aims to develop a VR environment that can be integrated with their fMRI goggles, Visual System HD, to provide visual stimulation when performing fMRI scanning on a test subject or patient. The project will hopefully offer great insight into the possibility of optimising digital solutions for both research and clinical application within the medical field.

1.2. Limitations

Throughout the duration of this project our work has been significantly affected by the COVID-19 pandemic. Because the NNL offices were closed due to NIPH's (Norwegian Institute of Public

Health) health regulations and a nationwide lockdown, it was not possible to test the VR application on the intended hardware until the final stages of the project when the offices reopened. This may have compromised the efficiency and accuracy of our work, and introduced a new level of uncertainty when working towards the main goal of our project, which was to be able to have a functioning application specific to the hardware developed by NordicNeuroLab.

Another limitation that occurred due to COVID-19, was the inability to perform user testing on real test subjects, which could be useful for adapting the application based on user experiences and feedback. When developing a VR experience with the player in mind it is necessary to consider their overall experience as well as complications such as virtual reality sickness and eye strain. These aspects were not heavily emphasised during this project based on recommendations from the project owner, however the game idea was developed based on well-known visual stimulation paradigms after consulting with a professional.

VR technology is a newer field of research and the learning resources for creating a VR game are mainly limited to commercial VR headsets and are not always applicable when developing an application for the fMRI goggles. The fMRI goggles do not contain a head-tracker like commercial VR headsets, because the head of the human subject must stay completely still during an fMRI scan.

1.3. Resources

Our personal computers will be used over the duration of the project. The personal computers will have a game engine installed to develop the VR environment. We need fMRI goggles to display the output of the VR environment, and its controllers to get the input for navigation.

The writing and planning part of the project will be assisted by our internal advisor Carsten Helgesen. Academic literature will be provided by the library's online resources as well as other online learning resources and software relevant documentation.

1.4. Organisation of the report

Chapter 1, Introduction, presents the motivation and goal, the limitations and the resources used in the project.

Chapter 2, Project Description, discusses who the project owner is and the significance of the project in regard to them. It also presents how the project is built on previous work, as well as the initial requirements and solutions.

Chapter 3, Project Design, discusses the possible approaches considered for this project. A detailed description of the game is presented. The project development method including the organisation and planning of the project, risk management and plans for evaluation are also presented.

Chapter 4, Detailed Design, discusses how the requirements are solved.

Chapter 5, Evaluation, discusses the details of the methods for evaluation and the results.

Chapter 6, Results and Discussions, presents the results of the project and the technical limitations that have affected the results.

Chapter 7, Conclusion and Further Work, presents a summary of the project goals. Details of useful applications of the project both within the same field and others are discussed.

Chapter 8, References, presents the reference to all sources of information used in the project.

Chapter 9, Appendix, presents the detailed risk assessment, the Gantt-diagram and user manual.

2. PROJECT DESCRIPTION

2.1. Project owner

The owner of this project, NordicNeuroLab, is a company based in Bergen that develops, manufactures and markets a complete fMRI system. This system includes both hardware and software components.

The VR/3D aspect of the project will help NNL promote their product Visual Systems HD. In recent years, fMRI has been used for various clinical applications such as pre-surgical studies, diagnosis and treatment. As fMRI starts to become a standard method, an increased demand in the clinical market for fMRI solutions is plausible. In the long run, the project will also help in giving insight into the possibilities within research and health-oriented software development applications.

2.2. Visual Systems HD

As mentioned above, this project is to be integrated into an existing solution by NNL called Visual System HD. Visual System HD is a solution for presenting visual stimuli during an fMRI procedure. Its hardware parts consist of the aforementioned fMRI goggles, see figure 1, which displays the visual stimuli presentation, as well as the ResponseGrip controllers. The hardware also has the functionality to record eye movements during fMRI scanning.

The visual stimuli presentation, called NordicAktiva, is the software part of the pre-existing solution. This can be either an image, video or a text-based task, see figures 3 and 4. During the procedure a technician presents the test subject with the visual tasks, and the test subject responds through a user feedback system by using the ResponseGrip, see figure 2.

Figure 1: fMRI goggles. The fMRI goggles created by NNL as a part of the Visual Systems HD solution, NordicNeuroLab, 2020. Available from: https://nordicneurolab.com/nordic-fmri-solution/

Figure 2: ResponseGrip. The ResponseGrip has two buttons on each controller, NordicNeuroLab, 2020. Available from: https://nordicneurolab.com/fmri-solution-v

The project will be integrated with the fMRI goggles serving as the screen output of the VR environment, and with the ResponseGrip as the means of input for navigation. The project is not intended to directly build on NordicAktiva. Rather, it is a different approach for presenting visual stimuli. NordicAktiva will serve as a blueprint for the type of visual stimuli which is to be developed.

Figure 3: NordicAktiva. A screenshot of a demonstration of NordicAktiva, an existing visual stimuli presentation which can be used with the fMRI goggles, NordicNeuroLab, 2020. Available from: https://nordicneurolab.com/nordicaktiva/

Figure 4: NordicAktiva. A screenshot of a demonstration of NordicAktiva, NordicNeuroLab, 2020. Available from: https://nordicneurolab.com/nordicaktiva/

2.3. Initial requirements specification

The initial requirements given by the project owner was for the product to include:

- Dual output, meaning the display output is to be rendered to both the left and right eye, which is essential for a 3D experience
- An environment where a player can navigate or play in, the environment can be:
    - Google earth
    - City
    - Store

    - Nature
    - Flying

- A user can navigate using ResponseGrips as keyboard input: A, B, C, D
- If there is inactivity in a given amount of time, the program switches to demo mode in which the program takes control of the player. Once an activity from the user is registered, the program switches from demo mode to an interactive mode, meaning the user is now the one navigating the player.
- A spectator is able to observe what is shown in the fMRI goggles on a PC screen
- An executable program is compiled that can be executed on a Surface laptop with docking, or a compact desktop computer
- Maximum 5 minutes setup-time
- Setup should not require expert PC or programming skills

Some features that were regarded as "Nice to have" by the project owner:

- The footage of the EyeTracking is shown on the PC screen
- Show eye-tracking footage on a monitor within the 3D paradigm viewer
- Show the eye-tracking footage in the fMRI goggles
- Show the gaze position as overlay on the PC screen as a cross, possibly as a heat map

2.4. Initial solution idea

Jerald (2016, p. 4) has stated that "virtual reality is defined to be a computer-generated digital environment that can be experienced and interacted with as if that environment were real". According to Parisi (2015, p. 7), there are four technologies needed to create virtual reality:

- Stereoscopic displays
- Motion tracking hardware
- Input devices

- Desktop and mobile platforms

Parisi has also stated that Oculus Rift, Samsung Gear VR and Google cardboard are some of the go-to choices for VR hardware, and that Unity3D and Unreal are the preferred tools for native VR development.

In this project the *stereoscopic display* is the fMRI goggles. The ResponseGrip hardware is used as an *input device*, while the computer and Unity game will serve as a *platform*. The eye tracking functionalities will replace the head tracking that comes with a head-mounted device, due to the restriction of head movement during fMRI-scanning.

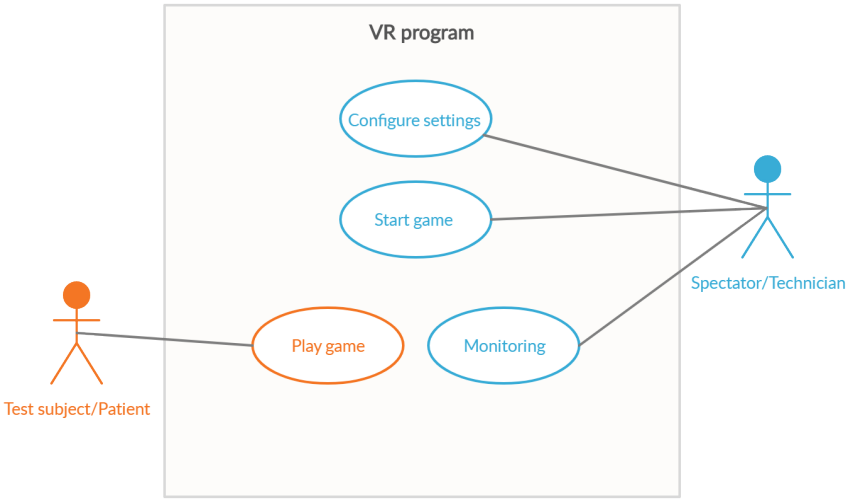

Figure 5: UML Use Case diagram. A use case diagram of the product. The orange stick figure represents the test subject while the blue figure represents the spectator or technician.

A use-case diagram, see figure 5, was created to better understand the interactions between the users and the product. An actor within a use-case diagram is a person who interacts with the system, which in this case is the spectator, who will also be referred to as the technician as well as

the test subject, also referred to as the patient within a clinical context. The user journey starts with the technician executing the software and configuring the settings, or doing both simultaneously through the command line through a Windows operating system. The technician will then start the game and is then able to monitor what is seen in the fMRI goggles on a preview screen. The test subject or patient will be able to play the game as soon as the technician starts the game, and this is the only use case in which they interact with the program. The technician is then able to monitor the game as a spectator throughout the duration of the game (Amber, 2003).

To create the VR environment we can choose between the two aforementioned game engines. The environment should simulate a natural environment such as a garden or park. There will be a gamification element where the player has to click and collect objects in order to progress in the game.

In order for the Visual System HD hardware to be supported by the game engine the development needs to be generic. By using OpenVR the system should be able to be utilised by any VR headset and controllers, and there is an option for creating a hardware specific driver. The hardware created by NNL can be connected through a USB-port and will then be recognised by the computer as a keyboard. It should therefore be possible to develop the game by using A, B, C, D inputs.

Since there are two simultaneous ways to use the product, the plan is to create some sort of on-boarding for the technician and a tutorial for the player to familiarise themselves with the game before starting.

When running the game through the fMRI goggles, there should also be the option of displaying the game on a monitor during playthrough. We will look at the possibility of placing two cameras within the scene to achieve this.

To implement eye-tracking, existing libraries such as OpenCV can be used to handle data in real-time.

Any game engine gives an option to compile the game as an executable file and should run on a Windows computer with no problem. It should then be easy to set up the game and there will be a user manual containing simple instructions.

3. PROJECT DESIGN

3.1. Possible approaches

This subchapter presents the themes wherein multiple approaches are possible. The first theme presents the possible game engines to use. The second theme discusses the possible game type that can be developed. The last theme discusses the two possible frameworks for the development of the product. All three themes conclude with the chosen approach.

3.1.1. Choosing the game engine

As mentioned, there are two game engines to choose from, namely Unity and Unreal. Upon recommendation from the project owner, Unity is chosen as the game engine. We have also used Unity for the purpose of learning game development during the pre-planning phase in preparation for this project.

3.1.2. Unity3D vs OpenVR

There was an uncertainty on the type of foundation the product would build on due to having custom hardware. The first possible approach could be using OpenVR, which will be discussed further. The second was building the application as a 3D program and adapting its features in order to create a VR program.

While doing research about VR application development on custom hardware, using OpenVR has become a sole option to approaching the project. OpenVR is an API that allows VR applications access to custom VR hardware. It has two layers, the OpenVR driver and the OpenVR application. The pipeline is built in a way where the OpenVR application communicates with SteamVR, then SteamVR communicates with the OpenVR driver. The OpenVR driver is a software any hardware manufacturer can develop for its system to be introduced to the SteamVR system. This makes any OpenVR application compatible with this hardware. In order for the application developed in Unity to run on the hardware, we have to develop the OpenVR driver and build the VR application with the OpenVR SDK.

The specifications of the hardware, however, have given us the option of developing a normal 3D game in Unity. As mentioned in chapter 2, the setup of the NNL hardware is designed in a way that the fMRI glasses are two monitors and the ResponseGrips work as a keyboard with A, B, C, D as the key inputs. These are connected to a Surface laptop with a docking station. Unity has the capability of detecting all active monitors and input devices connected to the computer. Thus, a Surface laptop, with the NNL hardware connected to it, can execute the executable file Unity compiles.

The chosen approach for this theme is developing on Unity3D. Not having to develop a driver gives more time for development of the application itself. We can focus fully on the requested functionalities rather than having divided attention on developing both a driver and an application.

3.1.3. Game type: find the object vs spot the difference

Jerald (2016, p. 45) has mentioned three concepts that can make users feel like they are somewhere else. These are immersion, presence and reality trade-off. Motion sickness is also something we have to consider and make sure the user does not experience. Among the factors in achieving immersion is the plot of the game. We have come up with two alternative game types

that are aligned with the project owner's requirements and the concepts of virtual reality. The first alternative is a "find the object" type of game where a picture of an object is shown on the screen and the goal of the player is to find this object. The other alternative is a "spot the difference" type of game where the player observes a room filled with different objects. The room rotates after a few seconds, changing the player's perspective of the room. The player then has to decide which objects have changed their placement in the room.

The chosen approach is to develop a "find the object" type of game. The choice was made in consideration of the following: the concepts that determine a virtual reality experience, the time we have for the development of both the game and the application features, and the limitations of the hardware setup. The trade-off in choosing this alternative is the player becomes more exposed to motion sickness. This is due to the player having the possibility to navigate around the environment which can cause a mismatch between what is seen and what is physically felt (Jerald, 2016, p. 165). A conundrum that came up was how the player was going to select an object if the four buttons were for moving up/forward, down/backward, left and right. Therefore, the first alternative became the more feasible option in this early stage of the applications where the only means for input is input from the ResponseGrips.

3.2. Game design

The planning of the game design is based on the basic theories of game designing presented by Menard & Wagstaff (2015, pp. 31-43). The following terminologies are presented in this subchapter: core and features. To conclude the subchapter, a detailed explanation of the gameflow is presented.

3.2.1. Core

As Menard & Wagstaff (2015, p. 32) have defined, "The core is basically one defining and unifying statement that describes your game. You can think of the core as being like a thesis

statement for a research paper". The core of this game is an immersive memorisation game intended to trigger brain activity. The sole character of the game is the player. A city is going to be the environment of the game wherein the player can navigate to accomplish an objective.

3.2.2. Features

"If the core is like the thesis statement of a research paper, the feature set is the collection of main ideas from each supporting paragraph. Each element in the feature set describes a main part of the play in your game, and each should definitely relate back to the core idea", Menard & Wagstaff (2015, p. 33) have explained. The game consists of the following features:

- Virtual city environment, see figure 6.
- First person perspective
- Memorisation is activated by showing an image of the objective for only a few seconds

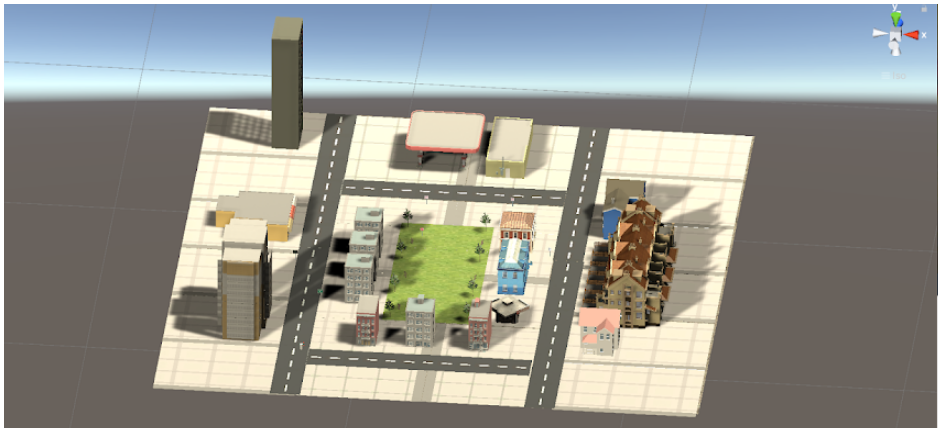

Figure 6 : An aerial view of the city environment.

The first two features support the core of the game by offering a virtual platform that can contribute to the player feeling more immersed in the experience. The last feature contributes to the core of the game by serving as the visual stimuli that can trigger brain activity. Since there are

multiple traffic signs in the city, it can provide a challenge for the player to determine the correct traffic sign.

3.2.3. Game mechanics

Menard & Wagstaff (2015, p. 32) have stated, "The most basic building block of the game is a mechanic, a rule or description that governs a single, specific aspect of play". This subchapter presents the flow of the game and an explanation of how the player can interact with the game.

The game consists of nine objectives. The general objective is to find a traffic sign that is shown to the player. After this sign is found, the player receives a new traffic sign to find. The player is able to navigate within the game environment through the ResponseGrip controllers. There are two buttons on each controller and the user is able to decide which button corresponds to which direction. The player can move forward, backwards and to the left and right. During the final stages of the game, an option for rotational left and right movement was implemented.

An example of a game session is as follows:
The player is placed in a city in the beginning of the game. As figure 7 shows, the objective is an Exit Left sign. After the image of the objective is shown for five seconds, the player can start looking around the city to find it. The player has to move close to the traffic sign in order to make the selection.

Figure 7: A screenshot of the compressed left and right display when an image of the traffic sign is shown.

A green check mark is shown, if the player has selected the Exit Left sign, see figure 8. Selecting the correct traffic sign increments the score of the player and the objective is completed. If the player has selected a different sign, a red cross is shown, see figure 9.

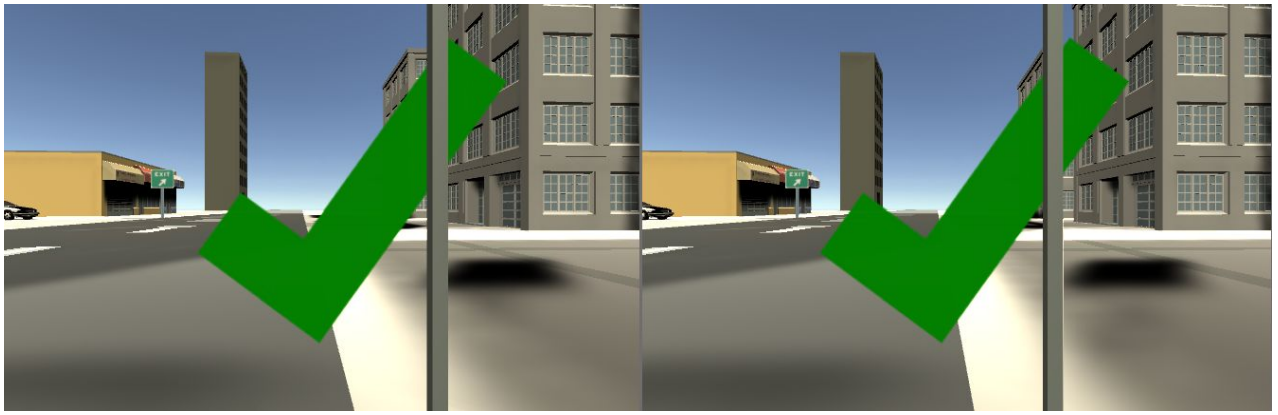

Figure 8: A screenshot of the compressed left and right display when the player finds the correct traffic sign.

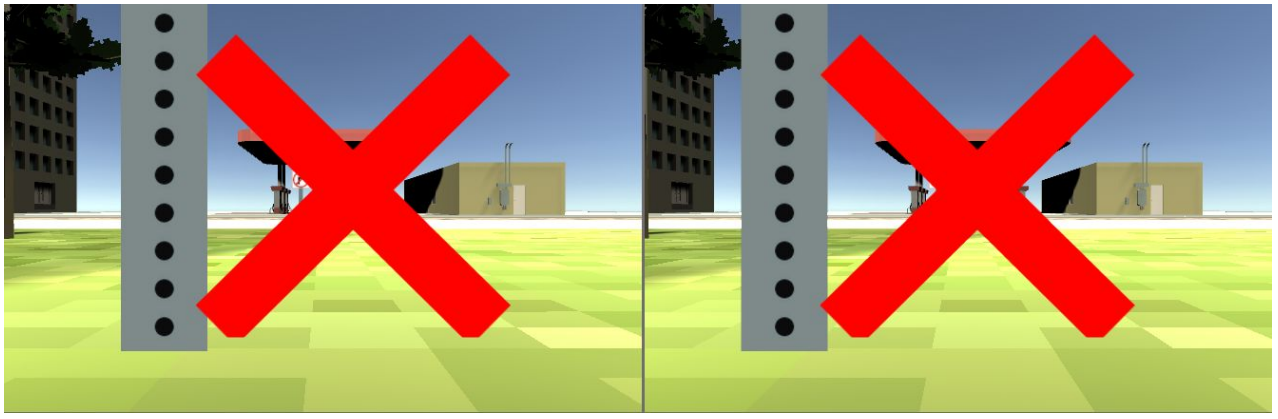

Figure 9: A screenshot of the compressed left and right display when the player selects the wrong traffic sign.

After the green check mark is shown, a new traffic sign, the Exit Right sign, is shown on the screen, see figure 10. Exit Right sign is now the objective. Figure 10 shows also that the score is

incremented after finding the Exit Left sign. The process mentioned is repeated until all nine objectives are completed.

Figure 10: A screenshot of the display rendered on the right eye of the player after finding the Exit Left sign.

3.3. Project development method

3.3.1. Development method

The project will be developed as an agile process. An agile method favors and prioritises individuals and interactions over the technical aspects of the development process and encourages collaborating *with* the customer instead of working *for* the customer.

The key to successfully mastering the agile method of development is to respond to change by working through the steps of negotiating the requirements, designing the software architecture, implementing the software, evaluating and maintaining and repeating them instead of following them in a top-down fashion (waterfall model) where one step must be completed in order to proceed to the next step (Measey, 2015).

The project plan includes six iterations, each requiring some sort of testing at the end of every iteration. By creating a Kanban-board, a board created to visualise the project management, it is possible to follow the continuous development and adapt the workflow and tasks accordingly.

Weekly meetings are held with the project owner through Teams to get feedback on our work and discuss the tasks for the next iteration. Each meeting would start with follow up questions from the previous meeting.

3.3.2. Project Plan

See GANTT-diagram in Appendix (9.2).

The main phase of the project is split into six different iterations, starting with creating the VR environment and placing the assets in the game. Integration testing should be performed by running it on a simulator and on the hardware.

The second iteration of the project is providing functionality to the game. In order for the 3D objects and other assets of the game to interact with each other, scripts must be written in the programming language C#. This requires both unit testing and integration testing to complete.

The third iteration will focus on integrating the Visual System HD controllers by initialising them through setup and scripting if necessary. Finalising this step requires integration testing on hardware.

The focus of the fourth iteration is stabilising the application by making sure it is fully optimised before launching it. This iteration will also include a timeframe to research and possibly implement optical distortion correction filters.

Iteration five is to make sure the application is easy for the customer to install and run. This can be achieved by performing user testing and writing a simple user manual if it is required.

The last iteration is reserved for eye-tracking if the rest of the iterations are completed on time.

3.4. Evaluation method

Evaluation was performed by running the game in the Unity game view simulator as well as testing on the hardware. Testing on hardware was both performed by the group members and the project owner, with the latter also being used as a method for user testing.

3.5. Risk management

To define the risks that might occur during the completion of the project, a risk matrix was used. A scale of 1-5 was used to evaluate the likelihood of the risk happening and the severity it had in affecting the completion of the project. If a risk does happen, the group had formulated measures to ensure the completion of the project.

Most of the assessed risks are based on the fact that we have little experience in game and VR development. Therefore a general preventive measure for the risk of not completing the project due to inadequate skills, was to learn Unity before the development phase. Integration of the application to the hardware was a crucial part in the evaluation of the success of the project. Therefore having limited access to the hardware was one of the biggest risks. To manage this risk, the first iteration of the development was dedicated solely to building the foundation of the application that was immediately integrated to the hardware. For more details about the risk management, see Risk list in Appendix (9.1).

4. DETAILED DESIGN

This chapter explains the details of how the requirements are solved. The architecture of the application is explained in the first subsection. The remaining subsections explain how the iterations of the project have been approached. The approach to creating this VR program using Unity is not the same as creating a VR program for a known VR hardware. This is why a subsection is dedicated to explain in detail how the foundation of the VR environment is established. It goes through how the framework of Unity 3D has been utilised to get the product to execute as a VR program on customised hardware. The next subsection talks about the implementation of the game functionality. Another subsection details handling customised input parameters. The last two subsections are about solving the preview screen requirement and optical distortion correction.

4.1. Architecture

The early process of developing the game required mapping the functionalities based on software architecture concepts. Sketches and descriptions were adapted throughout the course of the project and the final version was created, as shown in fig. 11.

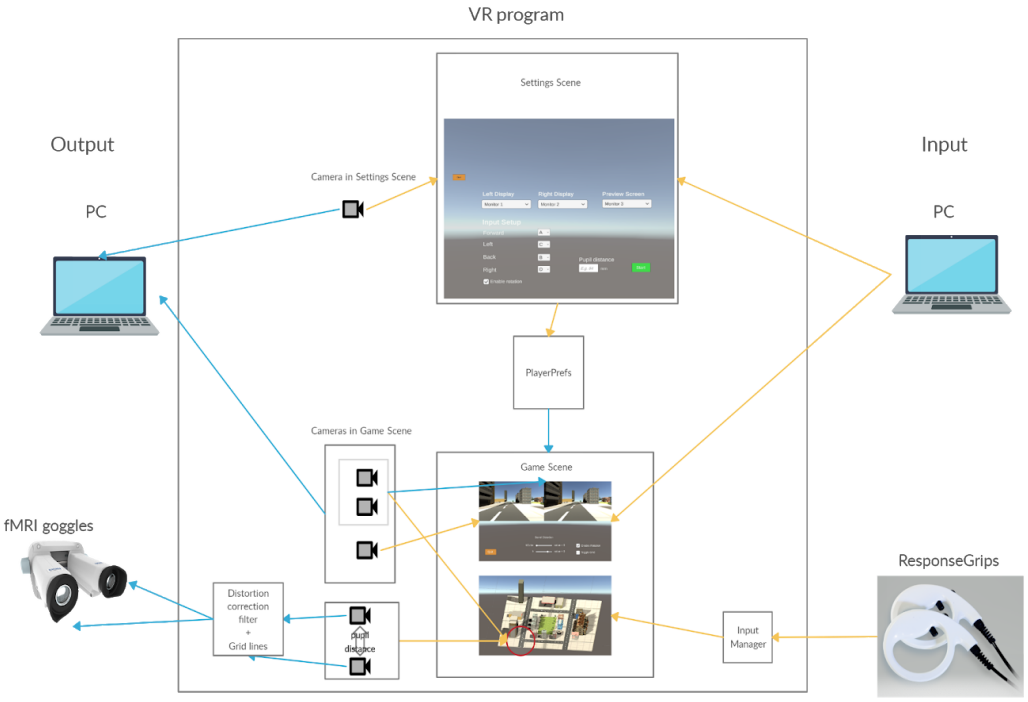

Figure 11: Architecture of the entire system. A visualisation of the interactions between the different components of the system.

Figure 11 shows a full overview of the entire system consisting of both software and hardware. This illustration is more detailed for the purpose of assisting other Unity developers, however the main concepts are explained here, and further explanations are presented in the next subchapter. The box in the middle represents the entire game or VR program created in Unity. To the left of this box is the computer running the program and this is the output monitor for the Settings scene displayed when the program is first executed. The technician can state their player preferences through the user interface on the Settings scene through computer keyboard input. After the technician clicks the “Start” button, the Game scene is loaded.

The Game scene is displayed for the test subject in the left and right displays of the fMRI goggles. The two cameras at the bottom of the diagram point towards the part of the game environment shown on the output displays. The left camera for the left eye display has a slight offset based on

the pupil distance between the eyes. A distortion filter is also applied to these two cameras as they are rendered to the fMRI goggle displays. In order to navigate within this scene, the test subject gives input through the ResponseGrip controllers.

There are also three cameras pointing towards the preview screen on the diagram. Two of these are copies of the left and right eye cameras for the technician to monitor what the test subject is viewing. The final camera points at a UI canvas for the technician to be able to control in-game settings such as the amount of barrel distortion or quitting the game.

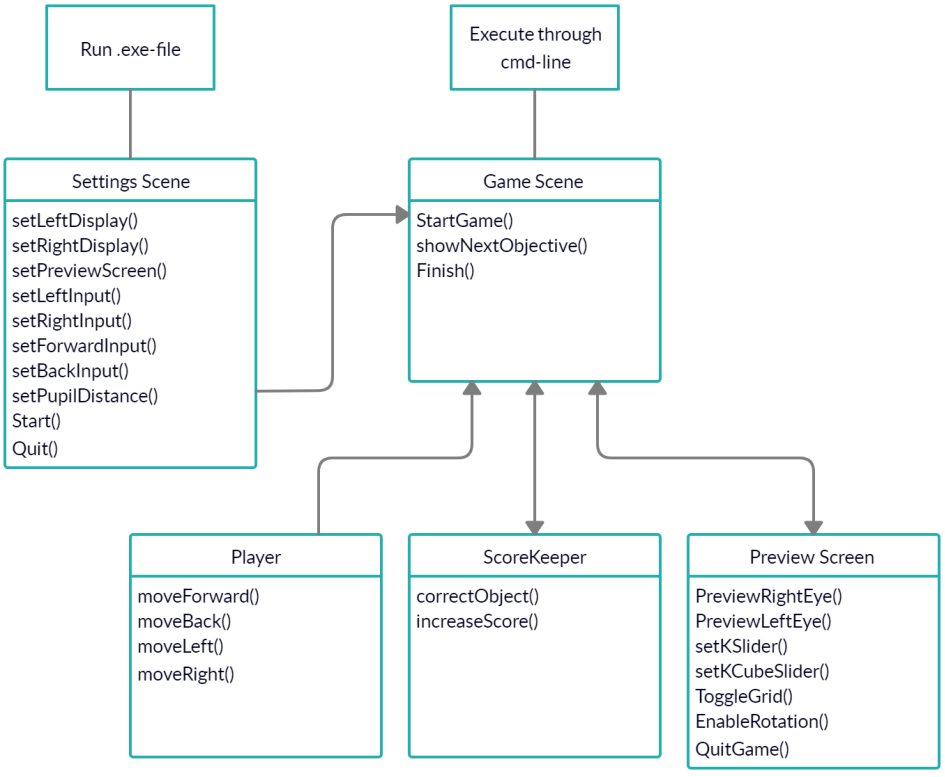

Figure 12: A simplified version of a UML diagram.

A UML diagram, as shown in fig. 12, is created to present the basic ideas behind the game structure which is the back-end of the program code. There are two options for executing the

game, either through the command line or by running the .exe file, as shown at the top of the diagram. Each box is titled with the name of a class. The main classes are the Settings and Game scene. The Game scene includes two subclasses, titled Player and ScoreKeeper, which give the player the ability to move, find the objectives and increase the score. The third subclass gives the technician the ability to control the game settings. The text inside the boxes represent the operations that are to be implemented within the various classes.

4.2. Creating the environment in Unity

The game is created as a 3D project in Unity 2019.3 and built by choosing Windows as the target platform from the PC, Mac & Linux Standalone build settings. The project consists of two main scenes: the Settings scene and the Game scene. A game created in Unity is separated into individual files called “scenes”. Each scene holds objects of the game such as UI components, cameras, 3D assets and other components that help build the game. All the components are placed in the scenes through the drag-and-drop functionality or by creating them through the Unity user interface.

4.2.1. Settings scene

The user experience of this game starts with the Settings scene unless the game is executed through the command-line with the help of the user manual. This is the main user interface for the game where the technician can select which monitor displays which image between the left eye, right eye, and the preview screen. The technician is also able to set the keyboard or controller inputs as well as the pupil distance of the test subject.

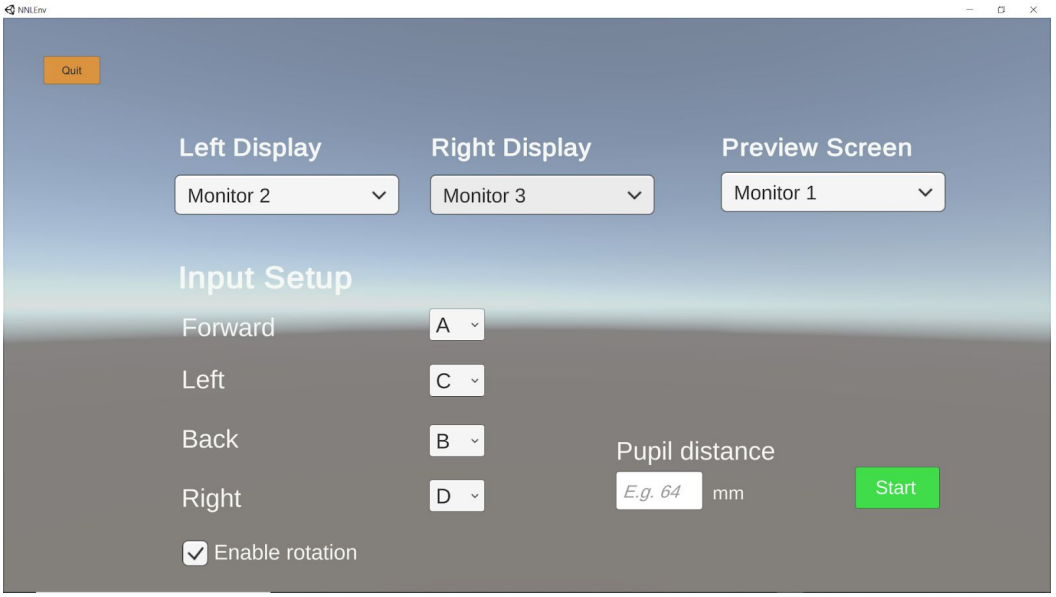

Figure 13: The Settings scene for stating player preferences before starting the game.

The technician provides input through Unity's default UI components with a drop-down menu consisting of strings for the displays and controller inputs and by entering the pupil distance in millimeters as an integer through an input box. These user inputs are stored as player preferences using the PlayerPrefs functionality through a script.

If the user leaves the input fields empty the default values are set to A for forward movement, B for backward movement, C to rotate or move to the left and D for right-side movements. The pupil distance is set to 64 mm based on the average distance between the left and right pupil on the human body (US Army- Medical Department, n.d.). The preview screen is set to show on Monitor 1 by default while the left and right eye displays are set to the second and third monitor.

The user can also set their preferred values by executing the program through the command line. To achieve this, a script handling command line arguments is added to an empty scene. This prevents the Settings scene from being briefly executed, thus providing a better flow of running the intended next scene. This functionality is achieved by using C#'s Environment class which is

under the System namespace. The Environment class has a method that returns a string array containing the command line arguments for a process. The command line arguments are parsed and converted to the correct data type in order to be saved in PlayerPrefs. The user is able to state their preferences based on the custom parameters, see the User manual in the Appendix (9.3) for more details.

4.2.2. Designing the Game scene

Building on the results from the first iteration, a city environment was created for the game. Prefabs, materials and textures found available for free from the Unity Asset Store are utilised to shape the environment. All the assets are scaled intuitively based on system testing to ensure a more realistic experience. A 3D plane is placed as the foundation and road prefabs are placed on top of it. A 3D capsule is used as the body of the player to mimic the stance of a person.

Figure 14: Another aerial view of the city environment.

The landscape of the city is divided into blocks separated by four main roads. Each block in the city is populated with several 3D buildings such as apartment complexes, gas stations, trees and cars. In the middle of the city is a park with a fountain in the middle and benches surrounding it. The grass is created by using premade textures. All the assets placed around the city are attached with a collider so the player does not walk through the objects. This functionality is explained further in the Game Functionality subsection (4.3).

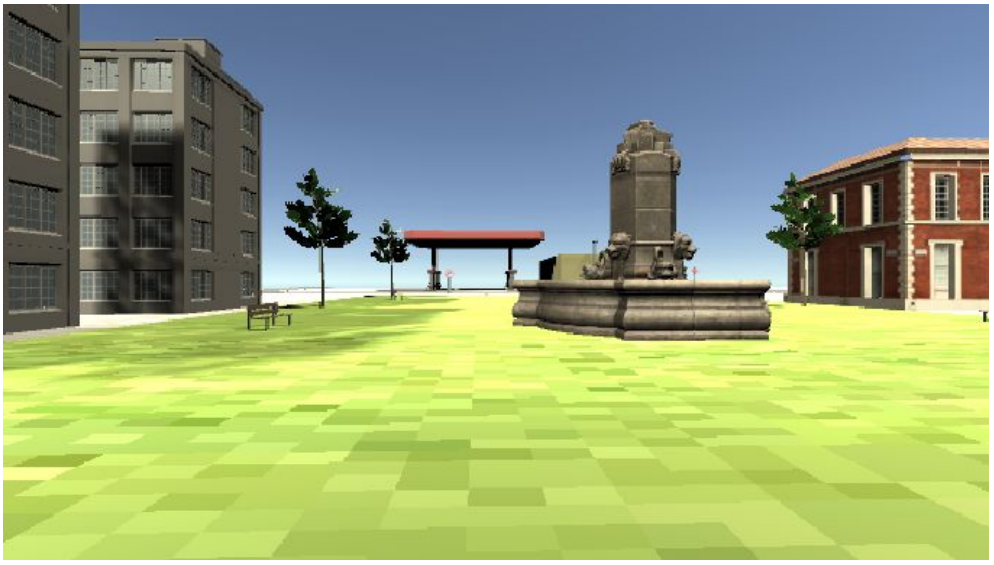

Figure 15: An in-game view of the park with a fountain in the middle showcasing the 3D assets.

When a camera is placed within a scene, it is automatically rendered inside a skybox creating the appearance of a sky around the city environment. Placing directional light in the sky adds to the realism of the environment by casting shadows onto the game objects and their surrounding. The skybox updates the direction of the shadows as the player rotates.

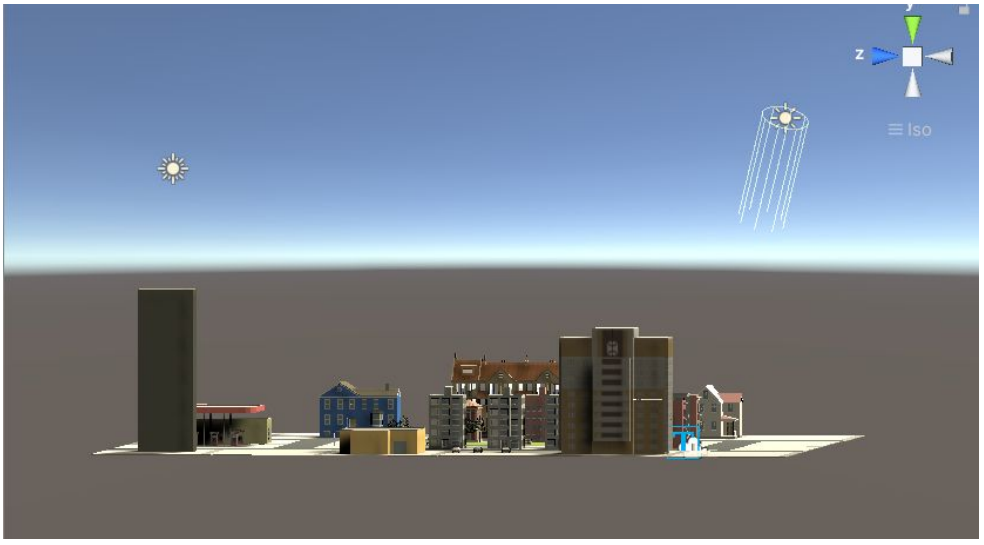

Figure 16: A view of the game environment. The directional light is placed in the sky on the left side of the city, casting shadows on the 3D objects and their surroundings.

4.2.3. Dual output

The dual screen output was created through Unity's multiple camera feature. Implementing this feature was achieved by placing two cameras from the same viewpoint in the environment. These cameras are then assigned to each their display and activated by a script for multi-display support from Unity's documentation. The target displays are also assigned through player preferences in the same script.

```
public class ActivateAllDisplays : MonoBehaviour
{
      void Start ()
      {
    for (int i = 1; i < Display.displays.Length; i++)
        {
            Display.displays[i].Activate();
        }
      }
}
```

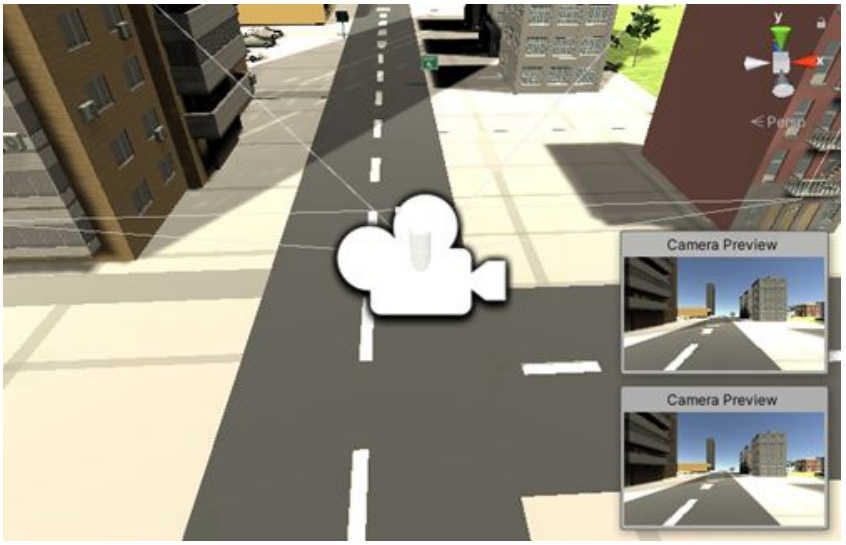

Figure 17: A scene preview in Unity showing both camera objects placed on top of each other and a preview of each camera view.

4.2.4. Achieving the "3D effect"

The stereoscopic displays that come with most commercial VR devices display a separate image for each eye where one image is slightly offset from the other to simulate parallax, which is the difference in how objects in front of us are viewed through each eye (Parisi, 2015). This was solved by creating a functionality which uses the pupil distance specified though player preferences, converting it to Unity measurements, and updating the position of the right eye camera when the game scene is loaded. The benefit of specifying the pupil distance through user input instead of using a default value, is to increase the accuracy of how the dual images are viewed as the fMRI goggles are adjusted according to the eye position of the player.

4.3. Game Functionality

Due to the setup of the dual output functionality, displaying images on the screen is achieved by using sprites, and not as UI elements. The images shown on the goggles, both images of the traffic

sign, and images of the green check mark and red cross, have a sprite texture and are stored in a game object. The game object containing the sprite gets activated and gets deactivated in accordance to the state of the program. To regulate the program's behavior, a finite state machine, see figure 18, is created. The flow of the visuals are also controlled by this state machine. The images are only shown either during the transition from "Show Objective" state to "Start finding" state, or during the transition from "Selected a traffic sign" state and the next state which is dependent on the validation the program has done.

To show the score, a TextMesh Pro game object is used because it is presented as a normal UI text but behaves in the similar way as a game object. TextMesh Pro does not need a canvas to be rendered, which is the best solution to display text to the goggles in the dual output setup of the application.

Colliders are set to all the objects in the city. A collider defines the object's shape and the possibility of interaction with the player. To differentiate the interaction between the traffic signs and the player from the interaction with the other objects, triggers are set to the traffic signs. These triggers are set to give feedback to the player. To avoid the player from falling off the plane, colliders are used as boundaries. A class in Unity Engine called Character Controller is set as a component of the player. This class inherits from the collider class and is in the shape of a capsule. Thus, the 3D capsule that represents the player gains the ability to interact with other colliders.

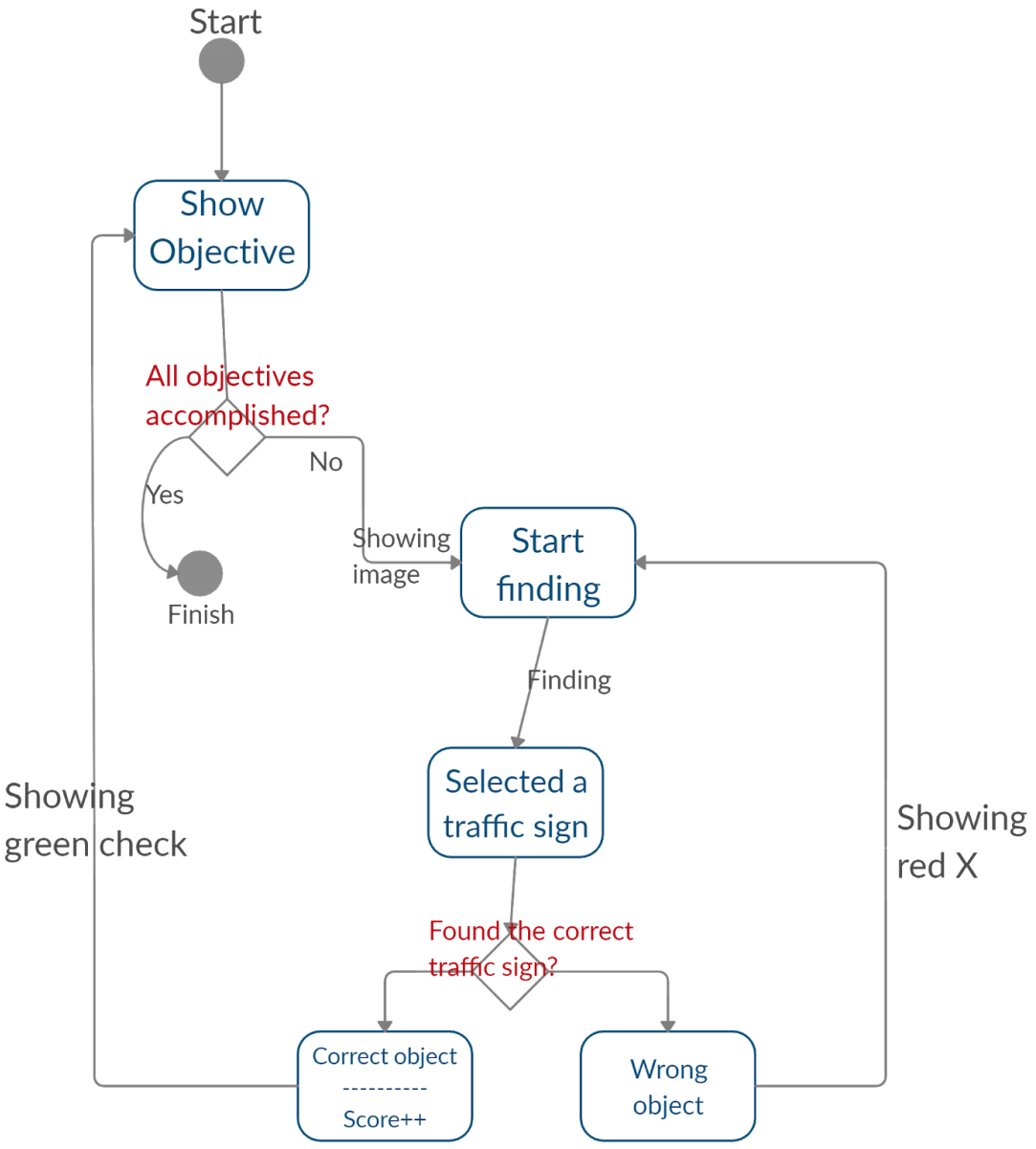

Figure 18 : Game system's finite state machine

To avoid motion sickness, the acceleration speed of the player is set to a constant low value. This minimises the mismatch between the virtual motion and what the human organs sense. The player's score is presented on the screen and acts as a rest frame. Jerald (2016, p. 167) explains that the "rest frame is a part of the scene that a viewer considers stationary and judges other motions relative to".

4.4. Input controller

Since the inputs from the four buttons on the ResponseGrips behave in the same manner as normal keyboard inputs of the keys A, B, C and D, the inputs are handled through scripting in accordance to the variables that are saved in the player preferences. This means if a key input is detected and is equal to one of the keys saved for navigation, the player will move in the corresponding direction. It is default that the player is allowed to rotate using the input buttons when moving sideways. The tehnician is however allowed to toggle the rotation option on the Settings screen before the game starts. If the ResponseGrip controllers are not available, they can easily be replaced with a regular keyboard.

4.5. Preview screen

The preview screen is a screen where a technician or spectator can monitor the player's movement within the game as well as being able to control the settings for the barrel distortion. This third screen is created by placing a third "Preview Screen" camera pointing towards the UI canvas for these settings and the two displays showing the preview in the fMRI goggles.

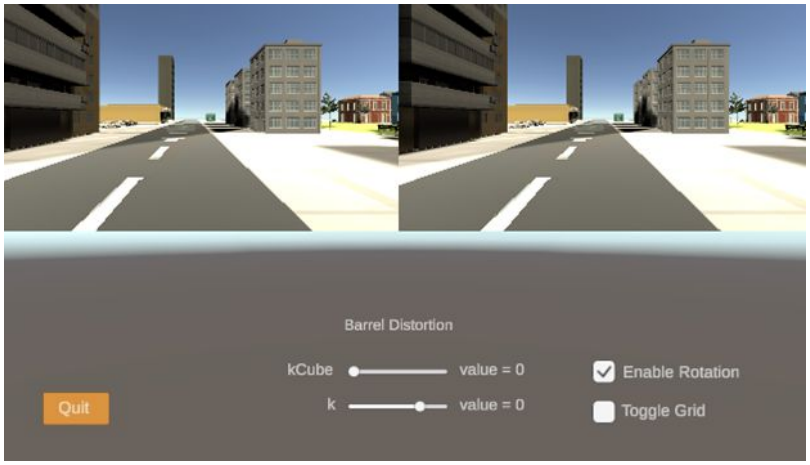

Figure 19: A screenshot of the preview screen

In order to display the left and right eye display side by side, the viewport rect coordinates are changed to fit the screen. The x and y coordinates range from 0 to 1 with 1 representing the full length of the screen. The left camera starts position x = 0, y = 0.5 and the right camera at x = 0.5, y = 0.5. The width and height are both set to 0.5 to cover 1/4 of the screen.

Figure 20: The viewport rect settings option in the Unity Inspector view.

To show what is viewed on the two displays in the fMRI goggles, two cameras were positioned on top of the main cameras for the left and right eye display. The depths for these cameras are then assigned the numbers 1, 2 and 3 in the order of which should be rendered first. This is to make sure the two game previews are rendered on top of the full preview screen.

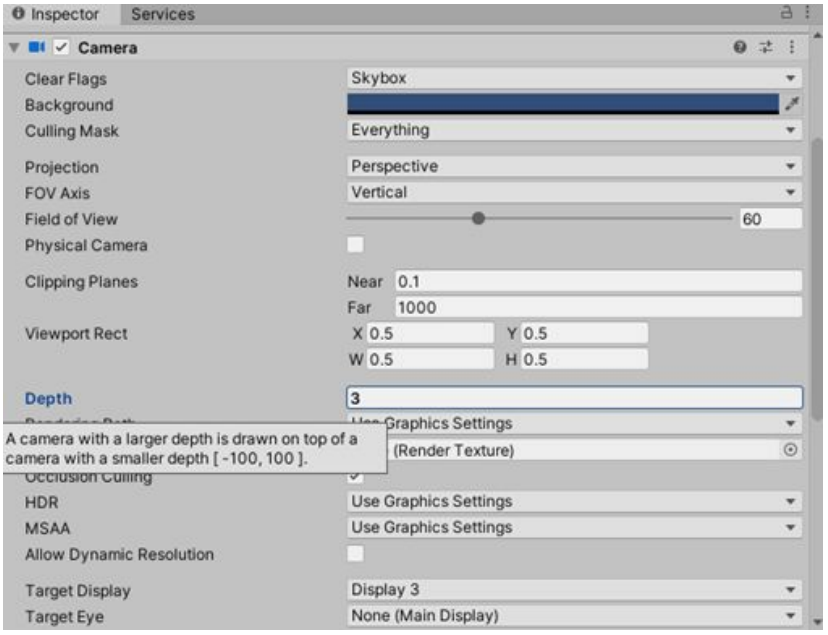

Figure 21: The Camera settings in the Unity Inspector view.

4.6. Optical distortion

4.6.1. Solving pincushion distortion

Optical pincushion distortion occurs due to the convex shape of the human eye when looking at the displays through the fMRI goggles. This causes the image to appear like it is bending inwards to the center of the image. This optical artefact can be corrected digitally by applying barrel distortion, which creates the opposite effect where the image magnification is higher at the center of the image.

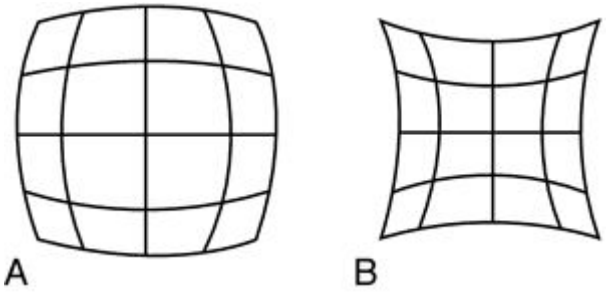

Figure 22: An example of barrel distortion (A) and pincushion distortion (B). Available from: https://medical-dictionary.thefreedictionary.com/barrel-shaped+distortion

The next figure (fig. 23) was provided by NNL and explains the issue of the pincushion distortion which occurs in the two OLED displays of the fMRI goggles.

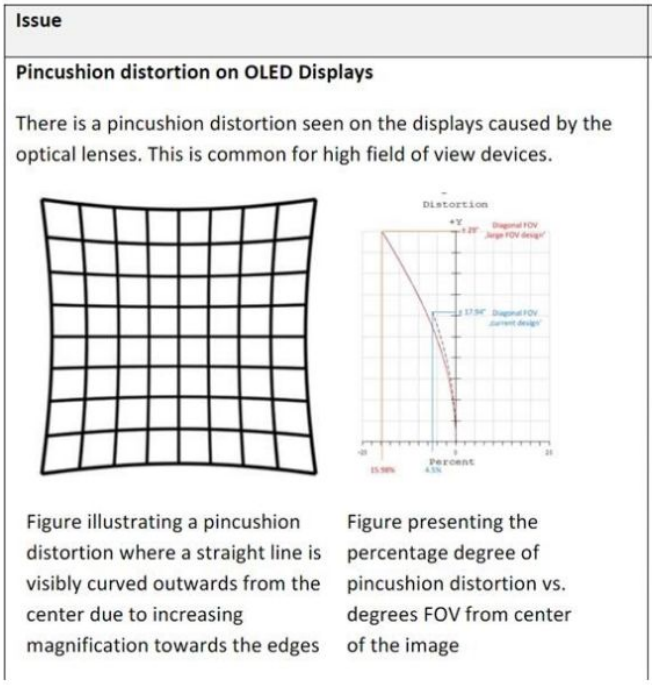

Figure 23: An explanation of the distortion observed in the fMRI goggle displays by NNL. Source: NNL.

4.6.2. Barrel distortion shader

A shader is a small script which can be applied to an object within the scene. It modifies the object by applying mathematical calculations to graphics data such as pixels, vertices or textures. Shaders are the most common method of adding different lighting and special effects to an object or scene.

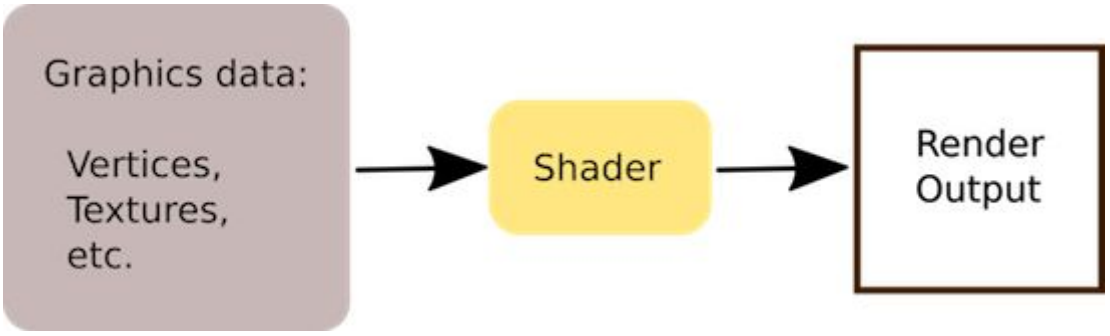

Figure 24: An illustration of how shaders work in Unity by using graphics data as input and rendering the output. Available from https://www.raywenderlich.com/5671826-introduction-to-shaders-in-unity

A shader was applied to the two cameras for the left and right eye in the fMRI goggle displays using a post-processing script. The script was based on an algorithm for barrel distortion using the vertices as graphical data and modifying their position and changing the outer pixels to a black colour. After the shader is applied, the image will appear distorted on a monitor as seen in fig. 25, however, inside the fMRI goggles it will be seen correctly through the human eye.

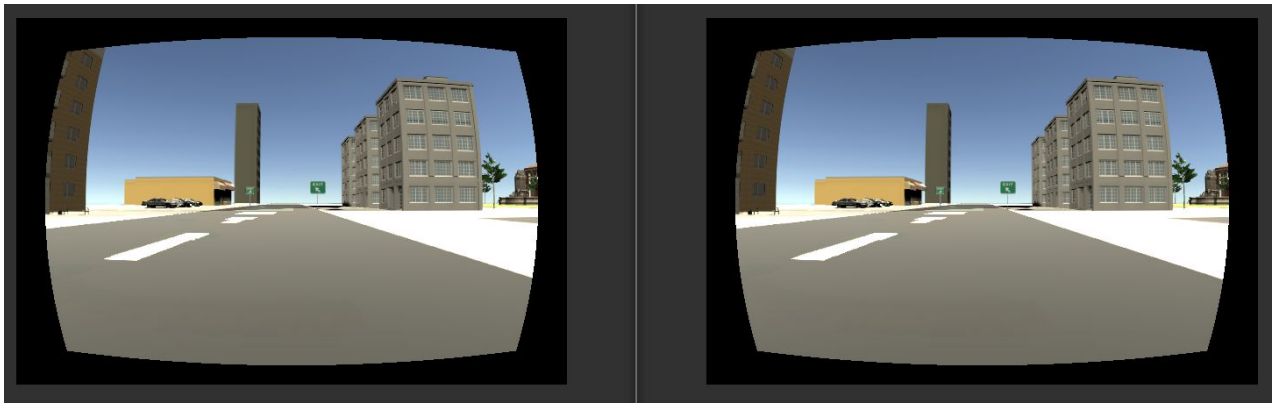

Figure 25: The display of the left and right eye after barrel distortion is applied.

4.6.3. Determining distortion amount

In order to determine how much distortion should be applied to correct what is seen inside the fMRI goggles, sliders were created for the preview screen to adjust the level of correction. This can also work as a user interface for further testing and was therefore kept on the preview screen. After user testing the range of the sliders were lowered twice to ensure greater accuracy.

A grid was also applied in front of the camera as a reference to observe how the image was changing. A toggle function to remove or apply the grid was then implemented using a UI toggle component and scripting, based on the project owners request.

5. EVALUATION

5.1. Evaluation method

The V-model, as shown in fig. 26, describes a method of testing which would be performed concurrently throughout the entire project lifecycle.

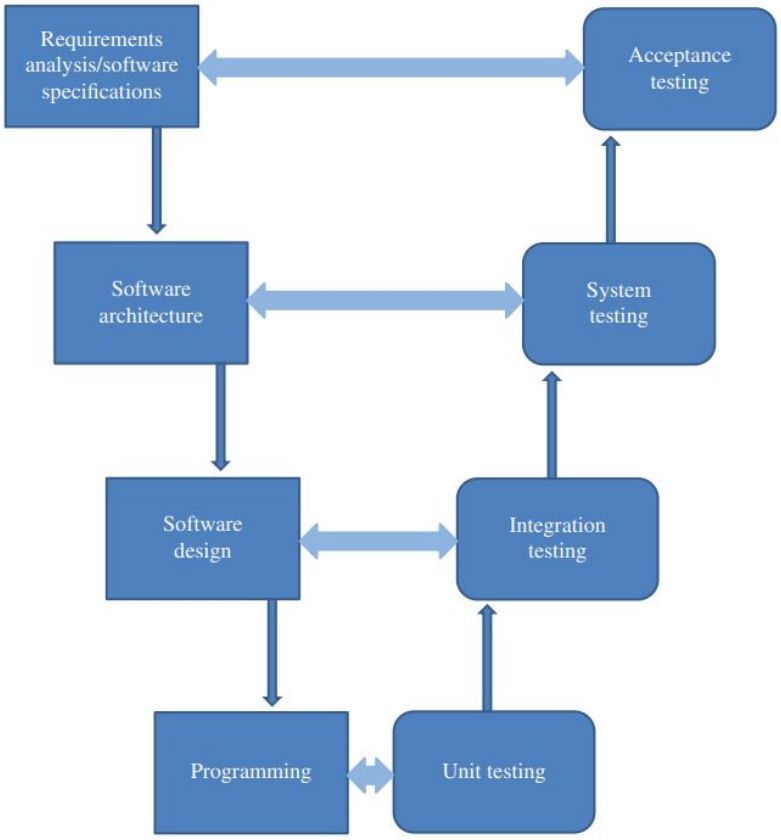

Figure 26: A diagram of the V-model of software testing. Mili & Fairouz (2015, Fig. 34, p. 33)

The goal of acceptance testing is to ensure that the product being developed meets the project owner's requests and expectations. This is the final and most important stage of testing and requires previous system testing. System testing is performed by the developers (in this case our project group) and this step entails evaluating the system as a whole by running the software on the intended hardware to ensure that it gives the expected output. While unit and integration testing was performed after every significant new addition or change to the program, system and

user testing on hardware was limited to being tested once a week or after the end of each iteration (Mili & Fairouz, 2015).

5.1.1. Unit and Integration testing with Game View

Observation using the game view became a key tool for evaluation. Unity (2020) described the game view as representative of the final, published game. This became a variant of manual assertion testing where the program was given different types of input while we observed if the output was as expected.

After a feature was added, it was tested by running the game in the game view. This provided quick and reliable results as to whether the newly implemented functionality was working correctly or not. If there was an error detected, changes and adjustments would be made following another round of testing. These steps were repeated until the results were viable. Testing using the game view was continuously used during the development of the application

5.1.2. System and User testing on hardware

After implementing a new and usually more extensive feature the game would be built and tested on a Windows operating system. Due to limitations mentioned previously, the testing was mostly performed by using the laptop monitor and an external monitor to emulate the two displays in the fMRI goggles. After each iteration the project owner would also perform user testing and report back. Each new version of the file included a description of the newly added features as well a list of post-testing questions. During the final stage of the project, the company offices had opened back up, giving the group members the possibility to test themselves. The application was tested several times on the fMRI goggles and ResponseGrips and a Surface laptop.

The final version of the project was tested by several users from the company as well as the project group. The observations were then discussed and some additional requests for adjustments and new features were received.

5.2. Evaluation results

The most important observations made during testing were that the dual screen output had scaling problems wherein the size of the player was not realistic in relation to its surroundings, as well as the range of the optical distortion filter slider not being sized correctly which affected the environment so much to the point where the surroundings were unidentifiable. During user testing the subjects also experienced some difficulty when controlling the sliders on a touch screen. The testing of the hardware resulted in an additional request from the project owner to have an option for the player to rotate horizontally, as opposed to walking sideways.

There was an uncertainty to whether specifying the distance between the eyes gave the 3D effect that was expected. Another uncertainty was whether the distance between the cameras on the game corresponded to the real life distance. In-game, there were also instances where the player got stuck inside a collider.

6. RESULTS AND DISCUSSIONS

6.1. Results

After the product was tested on the hardware, several adjustments had to be made. Most of the “must-have” initial requirements were implemented. The must-have requirement that was not implemented was the demo mode. A reassessment of the prioritisation of the requirements together with the project owner, resulted in the demo mode becoming a “nice-to-have” requirement.

The result of the project is a VR application where a test subject can play a game using the fMRI goggles and ResponseGrips while a technician/spectator supervises and adjusts the optical distortion on a preview screen if needed. During the application startup process, the technician can also change which button the player pushes to navigate to which direction, and toggle whether the player can rotate left and right instead of walking sideways. One of the advantages of this application is that the game itself can be developed by anyone because the functionalities such as the preview screen, distortion correction filter and custom input are almost completely separated from the game. The majority of the functionalities are built up by scripts that are attached to a game object to make use of their functions.

The system was tested not only by the group members but also a number of employees at NordicNeuroLabs. It was positively received by the project owner. We also made some last adjustments the project owner had requested and the necessary improvements based on what was observed during evaluation.

6.2. Technical limitations

During the development stage there were technical limitations that affected the quality of the result. The number of possibilities for visual stimuli are countless, however user input was limited without having implemented the eye-tracking functionalities. This led to limited game functionality such as the player not being able to experience realistic viewing angles and the player could not actively aim toward and select an object. This could lead to the player being less immersed in the experience.

The premade assets used in creating the environment were from multiple manufacturers. Scaling these assets proved to be challenging to be able to present them as one whole unit. The group members had to use the similar prefabs multiple times.

The ResponseGrips were designed to only take one input at a time. If a user held a button down, some graphics lag could be experienced due to the latency. This could potentially lead to motion sickness and cause the game to be less immersive and enjoyable.

Perhaps the biggest limitation was that an SDK (Software Development Kit) for a custom hardware was not available. Everything was implemented from scratch, which significantly limited the fundamental VR functionalities and design of the game. The project's progress was affected because a lot of time was spent building the foundation of the application. The goals of implementing eye-tracking were not reached, which was a feature that would have given the application more functionalities.

7. CONCLUSIONS AND FURTHER WORK

7.1. Summary of project goals

We were able to create an immersive VR experience for the fMRI goggles which will showcase the capabilities of the Visual Systems HD. The game provides visual stimulation to the player as well as brain exercise through the quest of memorising, recognising and localising objectives. The VR effect was created through implementing the offset between the two screens displaying a 3D environment and barrel distortion was able to correct the optical artifacts that occured in the fMRI goggles.

The "must-have" requirements were achieved, and whenever the project owner came with new suggestions and advice we were able to adapt our workflow. The "nice-to-have" requirements regarding eye-tracking and a demo-mode were not completed due to time limitations. However, the results will achieve their purpose of providing a basis for further research and development of fMRI technology at NordicNeuroLab.

7.2. Other applications

Other game developers can also find the project helpful as a paradigms for their own applications. Since the project is not built on any SDKs, developers who are interested in creating a VR application for custom hardware can use the project as a reference. Rehabilitation therapy and any field that requires memory recall training may also benefit from this template when creating VR environments.

7.3. Ideas for future development

There are many possibilities for further development of this game. The realism of the immersive environment can be greatly enhanced by creating more intricate 3D models and introducing more life-like movements and details. The complexity of the game can be increased by implementing more levels, objectives and even obstacles. If the eye-tracking is eventually integrated the player will have an improved viewing experience and the results will be more readily available for any user or technician. In terms of the gaming experience, if motion trackers are attached to the ResponseGrip controllers as well, Visual Systems HD could work similarly to a commercial VR system.

8. REFERENCES

Literature:

Ambler, S. (2003) *The Elements Of UML*. Cambridge [U.K.]: Cambridge University Press.

Bohil, Corey & Alicea, Bradly & Biocca, Frank. (2011) *Virtual reality in neuroscience research and therapy*. Nature reviews. Neuroscience. 12. 752-62. 10.1038/nrn3122.

Drap, P. & Lefèvre, J. (2016) *An Exact Formula for Calculating Inverse Radial Lens Distortions*. Marseille: Aix-Marseille Université, CNRS, ENSAM, Université De Toulon, LSIS UMR 7296, Domaine Universitaire de Saint-Jérôme, Bâtiment Polytech, Avenue Escadrille Normandie-Niemen

Glover, G. H. (2011) *Overview of Functional Magnetic Resonance Imaging*. Neurosurg Clin N Am., 22 April, pp. 133-139.

Jerald, J. (2016) *The VR Book: Human-Centered Designed for Virtual Reality*. University of Waterloo: Morgan & Claypool Publishers

Measey, P. (2015) *Agile Foundations : Principles, practices and frameworks*. Swindon: BCS Learning & Development Limited

Menard, M. & Wagstaff, B. (2015) *Game Development with Unity®, Second Edition*. Boston: Cengage Learning PTR

Mili, A. & Tchier, F. (2015) *Software Testing: Concepts and Operations*. New Jersey: John Wiley & Sons, Inc.

Parisi, T. (2015) *Learning Virtual Reality*. 1st release. Sebastopol: O'Reilly Media, Inc.

US Army- Medical Department (u.d.) *Notes on Eye, Ear, Nose, and Throat in Aviation Medicine*. 2nd Edition. US Army

Documentations:

Microsoft (2020). *.NET Documentation: Environment Class*. [Online]. [Accessed 26.05.2020]. Available from: https://docs.microsoft.com/en-us/dotnet/api/system.environment?view=netcore-3.1

OpenVR (2020). *API-Documentation*. [Online]. [Accessed 03.04.2020]. Available from: https://github.com/ValveSoftware/openvr/wiki/API-Documentation

Unity (2020). *Unity Scripting Reference*. [Online]. [Accessed 18.03.2020]. Available from: https://docs.unity3d.com/ScriptReference/

Unity (2020). *Unity User Manual (2019.3)*. [Online]. [Accessed 18.03.2020]. Available from: https://docs.unity3d.com/Manual/index.html

Images:

Medical Dictionary (2020). *Barrel distortion, pincushion distortion*. [Figure]. [Accessed 20.05.2020]. Available from: https://medical-dictionary.thefreedictionary.com/barrel-shaped+distortion

Mili, A. & Tchier, F. (2015). *Software Testing: Concepts and Operations*. Hoboken, New Jersey.

NordicNeuroLabs (2020). *fMRI goggles*. [Photograph]. [Accessed 31.03.2020]. Available from: https://nordicneurolab.com/nordic-fmri-solution/

NordicNeuroLabs (2020). *ResponseGrip*. [Photograph]. [Accessed 02.04.2020]. Available from: https://nordicneurolab.com/fmri-solution-vshd/

NordicNeuroLabs (2020). *A screenshot of a demonstration of NordicAktiva*. [Photograph]. [Accessed 31.03.2020]. Available from: https://nordicneurolab.com/nordicaktiva/

Ray Wenderlich (2020). *An illustration of how shaders work in Unity.* [Figure]. [Accessed 27.05.2020]. Available from: https://www.raywenderlich.com/5671826-introduction-to-shaders-in-unity

9. APPENDIX

9.1. Risk list

Name	Description	Severity	Probability	Management approach
Integration risk	The VR environment will fail to integrated to the hardware	5	3	Test the environment right away in the hardware.
Project complexity	The requirements are too complex to be solved	5	4	Close communication with advisors, consistent work
Resource risk	Resource needed are insufficient/inaccessible	5	3	Find alternative approaches
Skills risk	Needed skills are insufficient	4	3	Incremental development, one thing at a time
Loss of support	Initial additional support from the project owner cannot be given due to unforeseeable circumstances	4	4	Find/replace the support from someone/somewhere else
Time constraint	Limited time to satisfy all the requirements	3	2	Work consistently, focus more on the foundation of the product (environment, integration to hardware)

Health risk	Group member gets sick. Pandemic (covid-19)	3	3	Continuous communication between members

9.2. GANTT diagram

https://docs.google.com/spreadsheets/d/1I16YNxNBp3DltuV14HW3Dgxk4Y8vhfZVCgEVmy-TI0s/edit?usp=sharing

9.3. User manual

Executing the application through the command line

Parameter to write	Description	Available values
leftDisplay	which monitor displays the left eye	1, 2, 3
rightDisplay	which monitor displays the right eye	1, 2, 3
mainDisplay	which monitor displays the preview screen	1, 2, 3
leftKey	key input to move/look left	W, A, S, D, B, C
rightKey	key input to move/look right	W, A, S, D, B, C
backKey	key input to move/look backwards	W, A, S, D, B, C
forwardKey	key input to move/look forward	W, A, S, D, B, C
pupilDistance	distance between the left and right pupils	Any number

1) Navigate to the folder where the executable file is placed
2) Type in the command-line prompt:
   NNLEnv parameter1=value1 parameter2=value2 parameter3=value3 parameter4=value4 parameter5=value5 parameter6=value6 parameter7=value7 parameter8=value8
3) Press Return Key.

- Example:

  NNLEnv mainDisplay=1 leftDisplay=2 rightDisplay=3 fowardKey=a backKey=b rightKey=d leftKey=c pupilDistance=64