University of Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

BACHELOR'S THESIS

Study programme: Bachelor's degree in Computer Science	Spring semester, 2020 Open
Authors: Håvard Godal Kjetil Svihus	
Programme coordinator: Erlend Tøssebro	
Title of bachelor's thesis: Vocational VR - A Virtual Reality Edutainment Game	
Credits: 20	
Keywords: C#, Edutainment, Game Development, JSON, Oculus Quest, SideQuest, Unity, VR, VRTK	Number of pages: 59 + supplemental material: 7z-File Stavanger, 14/05/2020

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Vocational VR

A edutainment VR application

Bachelor's Thesis in Computer Science by

Håvard Godal

Kjetil Svihus

Internal Supervisors

Erlend Tøssebro

May 14, 2020

Abstract

Education can often be seen as uninteresting and boring. In the school context, it is important to organize the teaching so that all students learn in the best possible way. Using virtual reality, educational information can be presented in a whole new way. This allows students to engage with academic content in a more practical and fun way.

By presenting vocational careers as minigames in a virtual reality program, students both get a chance to see and experience tasks that such careers might come across. By including an aspect of fun, the minigames might become a bit more distant from real-life vocational tasks. Despite this, the games will become more entertaining, improving student engagement and the learning outcome.

As requested by Karrierelaboratoriet, the vocational careers this thesis focuses on is the plumber and the ventilation profession. From these requirements, a total of 6 different minigames and one quiz were developed to introduce students into the world of virtual reality as well as to challenge them in demanding and entertaining tasks.

The two first minigames developed were showcased at UiS' open day event on March 3th 2020. Feedback gathered from this day shows that with some adjustments regarding entertainment, the applications could be well suited for educational purposes.

Preface

This thesis documents the work done by the authors and developers Håvard Godal and Kjetil Svihus. All work is original and created by the developers alone. Previously published articles and documentation used in this thesis have been documented and credited.

The subject of this thesis is virtual reality. The subject derives from the authors' background in application development as bachelor students of computer science. VR applications are a new type of technology both suited for entertainment and business. This thesis describes the origin of VR and explains how VR can be used to bridge entertainment and education together, resulting in a final VR product; Vocational VR.

This thesis and all attached work have been produced for the Faculty of Science and Technology at the University of Stavanger.

Acknowledgements

We wish to show our gratitude to Karrierelaboratoriet for allowing us to work on this thesis. This includes the stakeholders; Chairman Paul Mathias Fiskaaen and CEO Kjell Sigve Lervik.

We would also like to show our appreciation to the Faculty of Science and Technology at the University and our supervisor Associate Professor Dr. Erlend Tøssebro - Department of Electrical Engineering and Computer Science.

We are grateful for the students who gave valuable feedback and criticism on our product, both from the open day at the university and from curious students that showed up at the lab.

Contents

Chapter 1

Introduction

1.1 Background

This thesis is made in cooperation with Karrierelaboratoriet and the Faculty of Science and Technology at the University of Stavanger. The thesis was suggested by the Department of Electrical Engineering and Computer Science. The prompt was to develop and create a set of educational and entertaining minigames to be used by students studying vocational education. The minigames were used to create an entertaining way to learn about plumbing and ventilation. The goal of the Virtual Reality (VR) application is to prove that a combination of education and entertainment in the form of virtual reality can be used as a teaching-supplement.

During the annual open day in March, high school students got the opportunity to try out some of the minigames. Their feedback and evaluation of both the minigames and the concept of edutainment were included in the further development of the application.

1.2 Introduction to Virtual Reality

The concept of providing an environment that can be experienced in an immersive has been worked on for over 50 years. From early machines capable of displaying wireframes over the real world, to fully standalone devices capable of tracking both head and hand-movement. Although virtual reality head-mounted displays (HMDs) were primarily created to entertain users, the modern world has adopted VR to both work and education. VR provides a new way of consuming information, entertainment, and knowledge, and can introduce an interesting and fun way of obtaining new knowledge.

1.2.1 What is Virtual Reality?

Virtual reality is the concept of using computer technology to create a simulated environment. VR places the user inside an experience, allowing real-life physical movement to be used to experience the environment. Virtual reality allows the user to interact with the 3D world while senses such as vision, hearing, and movement are being simulated, creating a very realistic scenario. Standard virtual reality systems use either multi-projected environments or a HMD with two small screens in front of the eyes. The person using the VR equipment can look around in the artificial environment, advance the surroundings by either moving or teleporting, and interact with objects found in the fictional reality.

To interact with the virtual environment, a set of motion controllers are used. The controllers allow the user to interact with the environment by moving their hands in real life. Most controllers are designed with ergonomics in mind, so the user can hold the controllers in a natural hand position. Designing controllers in such a way makes picking up virtual objects and interacting with them feel natural, which makes the virtual reality feel more real.

VR can be applied in a lot of different situations, both for entertainment and educational purposes. With the increased popularity of virtual reality, many video games now support virtual reality to immerse the player in a new way. VR is also being used in the workspace, providing users the ability to explore 3D models and environments of real areas, without being there in real life. The same concept can be applied in an educational context. In vocational education, being able to simulate real-life tasks that some professions might work on can be an interesting way to learn and practice.

1.2.2 The History of Virtual Reality

During the 20th century, watching movies at the cinema became a thing of the common. Some movie creators even tried to add gimmicks to the movies, such as the use of 3D goggles. Filmmaker Morton Heilig took things even further. In 1957, he invented the machine called the Sensorama, which looked like a large booth. This machine could give up to four people the illusion of being in a 3D world. The Sensorama did not only "interact" with the vision, but it also provided smell, vibrations, and atmospheric effects like the wind. A few years later, he forged the idea of the world's first head-mounted display with a wide field of view and stereo sound. Although his plan never saw the light of day, it laid the groundwork of future VR ideas. [1]

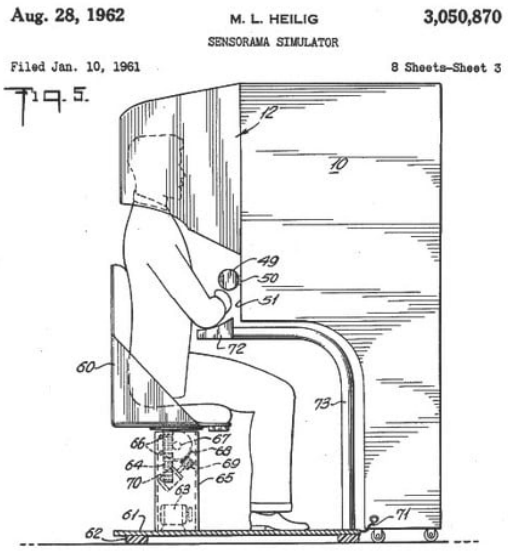


Figure 1.1: A sketch of Morton Heilig's Sensorama

The first actual VR head-mounted display was created in 1968 by the computer scientist Ivan Sutherland. Sutherland is often regarded as the father of both computer-aided design with his creation of the Sketchpad and augmented reality with his head-mounted display. The HMD could display simple virtual wireframe shapes, which changed position and perspective as the user turned their head. The wireframes were rendered over some form of glass, meaning that the user could see the real world behind the wireframes. The HMD can, therefore, be seen as an augmented reality HMD. This head-mounted headset never made it out to the public, most probably because it had to be suspended from the ceiling. [1]

Figure 1.2: The head-mounted display created by Ivan Sutherland

One of the first commercial attempts on a virtual reality HMD came in 1991 by Sega. Because the game console Genesis was beginning to reach the end of its lifespan, Sega VR was created to get a couple of years extra on the console. The Sega VR came with two LCDs, stereo headphones, and an internal sensor for tracking head movement. According to this press release from 1993 (https://www.consolecity.com/forum/archive/index.php/t-69980.html), Sega stated that "as your eyes shift focus from one object to the next, the binocular parallax constantly changes to give you the impression of a three-dimensional world." The Sega VR was never released to the public. As Sega puts it, this was because the virtual effect was too realistic, and the company was afraid that the users could hurt themselves while playing. This, combined with that almost every user that tested the Sega VR before release, got motion sickness, resulted in the cancellation of the virtual reality head-mounted headset. [1]

Figure 1.3: Sega VR

1.2.3 Modern Virtual Reality

In the 21st century, virtual reality has become much more mainstream, and many different types of VR headsets have reached the consumer. In 2010, Palmer Luckey designed a prototype of the Oculus Rift. The Rift provided a 90-degree field of view and could track head-rotation. A couple of years later, Facebook bought Oculus, indicating that virtual reality was believed to be a part of the future. At the same time, Sony announced its own VR project, PlayStation VR. Between 2014 and 2017, Oculus, HTC, Sony, and Google all created their variant of a virtual reality HMD.

Oculus released the first consumer-oriented VR headset in March 2016 named the Oculus Rift and cost $600 at launch. A few weeks later, HTC released its competitor to the Rift, named

the HTC Vive. The Vive was around $ 200 more expensive than the Rift and required more powerful PC-hardware to function properly. Some months after the release of the Rift and the Vive, Sony launched its own virtual reality headset for the Playstation 4 named Playstation VR. The Playstation VR had a launch price of $400. The Playstation VR relied on the hardware of the Playstation 4 and did, therefore, not compete with the previously launched VR headsets directly. One month after Playstation VR was released, Google announced and launched their version of a virtual reality headset, the Google Daydream. At the price of $99, the Daydream relied on a smartphone to work. The Daydream itself only consisted of a case with a couple of lenses inside. This meant that to experience a virtual environment, the user had to place their phone inside the Daydream to provide both hardware and a way to render images. In 2018, the first standalone virtual reality headsets were created, being the Oculus Quest. Being created by Oculus, the Quest was very similar to the 2016 Rift. The main difference is that the processing power normally done by an external computer was moved to inside the headset itself, removing the need for external hardware.

Today, all VR headsets can be divided into three categories: tethered, standalone, and mobile.

Tethered VR head-mounted displays connected to a PC was the first type of virtual reality headsets, having prototypes created back in 1968. Such VR headsets require a PC and specially equipped premises where the virtual reality environment will take place. Virtual reality headsets connected to a computer can use the computing power of the PC, and can, therefore, render more complex visuals while still providing an immersive virtual environment without dropping frames. The main downside of a PC-operated VR headset is the wires connecting the two. Because everything is running on the PC, the immersion the VR headset can offer is limited to stationary actions, as the wires prevent the user from being fully immersed with 6 degrees of freedom. Another drawback is that most PC VR devices are designed for VR enthusiasts, meaning that the price is often very high compared to other options. Also, a powerful PC has to be used in collision with the headset, which also has a pig price mark on it.

Standalone VR HMDs are quite the opposite of head-mounted displays connected to a PC. A standalone VR device operates fully by itself, avoiding wires to limit the movement. Therefore, standalone VR headsets can provide the user with 6 degrees of freedom, allowing movements in the real world to be interpreted into movement in the virtual reality. Being a small portable computer strapped on the user's head, a standalone VR headset does not have the same compute power as a stationary PC. This means that some applications that demand a lot of computing resources might not run as intended on a standalone VR headset. Such applications have to be designed with a standalone VR headset as the primary priority. Despite this, standalone VR headsets are among the most popular virtual reality headsets. Due to a reasonable price compared to PC operated VR headsets, standalone VR devices are more commercial.

Mobile VR-solutions consists of some sort of container, containing two lenses and a pocket for a mobile device. The main use-case for most VR headsets for smartphones is to watch videos and movies in a virtual environment. The lack of controllers makes interactions in the virtual reality difficult. Because the computing power is done by the smartphone, most mobile VR HMDs are cheap compared to other alternatives.

Chapter 2

Methods and Tools

2.1 Oculus Quest

In this thesis, the developers decided to develop Vocational VR to run on the Oculus Quest, a fully standalone VR headset. The developers and Karrierelaboratoriet agreed that an application such as Vocational VR would provide the most immersive and clutter-free experience on a standalone virtual reality headset. Because the application develops around vocational work, 6 degrees of freedom had a high priority. The application does not require a lot of computing power, meaning that the Oculus Quest's processor is more than good enough to run the application as intended. By developing an application to such a commercial VR headset, this thesis will provide an exciting insight into how VR applications that have the potential to reach out to a large number of users can be developed. [2]

The Oculus Quest is a virtual reality headset created by Oculus VR, a division of Facebook Inc. The headset uses two OLED displays, each with a resolution of 1440 x 1600 and a refresh rate of 72 Hz. The Oculus Quest is using Android 7.1.1 as its operating system, which released in 2016. The Quest is also entirely wireless and features six degrees of freedom (6DoF), meaning that it can track movements in any direction. The VR headset has a field of view of 95 degrees.

Oculus Insight is a VR system that uses computer vision algorithms to enable real-time room mapping and positional tracking on the Oculus Quest headset. Oculus Insight combines sensor data from the cameras in the VR headset, rotational velocity and linear acceleration from the internal measurement units in the headset and controllers, and infrared LEDs in the controllers that are detected by the headset cameras. It does this to give an accurate representation of the player's place in the real world. It will look for corners and edges, and then build up a 3D playspace and combine this with the gyroscope and accelerometer to give the headset's position every millisecond. [3]

Oculus has the users set up a guardian system. The Guardian boundary system is designed to display in-application wall and floor markers when the user walks near the play-area borders they defined. When the user gets too close to the edge of a boundary, a translucent mesh grid is displayed as a layer that is superimposed over the game or experience. [4]

2.2 Unity

Unity is a cross-platform game engine developed by Unity Technologies. It gives developers the ability to create games and experiences in both 2D and 3D. The engine offers a primary scripting API in C# combined with drag and drop functionality. Being a cross-platform game engine, the editor itself is supported on Windows and macOS, while the game engine supports building games for more than 25 different platforms, including Android.

2.2.1 GameObjects

GameObjects are the fundamental objects in Unity that represent props, scenery, and logic. GameObjects do not have any functionality by themselves but act as containers for Components, which implements the real functionality.

2.2.2 Components

Components define the behavior of GameObjects. They act as basic building blocks for objects and their function in a game. The components do something, such as instantiates scripts or defines a position in 3D space, on behalf of the corresponding GameObject. [5]

- Transform - Transform is the most used component in Unity. Whenever an object is added, it has the transform component. This component allows the developer to manipulate the size, position, and rotation of the object. [5]
- Mesh - Meshes are the main graphics primitive of Unity. A Mesh consists of multiple small triangles in 3D space that together forms a solid surface or shape. All visible and interactable objects in Vocational VR is created through the Mesh component. [5]
- Rigidbody - Rigidbodies enable GameObjects to act under the control of physics. The Rigidbody can receive forces and torque, such as hand interactions from the user, and move in a realistic way. [5]

- Collider - The Collider defines the shape of a GameObject for the purpose of physical collisions. The Collider is invisible, and when two colliders collide or overlap, factors such as speed and rotation of the colliding GameObjects can be read through the C# methods used by the Collider. The Collider shape most used in Vocational VR is the Box Collider. The Box Collider is a primitive Collider, requiring a very low amount of processing power. By using multiple Box Colliders on the same GameObject, the amount of processing power is still kept low, while the combined Collider creates a shape similar to the Mesh component attached to the GameObject. [5]

- Material - Materials in Unity defines how a surface should be rendered by including references to the texture it uses, tiling information, and color tints. Which properties displayed depends on which shader used. Shaders determine how texture and lighting information is combined to generate the pixels of the rendered GameObject. When the material is ready to be used, it can be applied to a GameObject. [5]

- Script - The script is one of the most important components of Unity. Scripts allow for making custom components through C#. Scripts allow the developer to trigger game events, modify Component properties over time, and respond to user interactions. By default, all scripts created contains a Start method and an Update method. The Start method is executed when the GameObject the script is attached to is loaded, and the Update method is executed every frame. [5]

- Canvas - The Canvas is a component that controls how a group of 2D elements will be rendered in a 3D environment. Unity is designed so that all 2D GameObjects must be placed as a child of a Canvas. The Canvas has multiple render modes. This means that 2D elements can be displayed to the user in several ways. When creating VR applications, the render mode must be set to world space. This means that the Canvas, along with its 2D elements, can be freely positioned and rotated in the game environment. Other options include the overlay render mode, attaching the Canvas to the eyes of the user. This means that wherever the user looks, the Canvas will always be visible. [5]

- AudioSource - An AudioSource is a representation of an audio clip in 3D. AudioSources are used in Vocational VR to give audible notifications, such as completion of a minigame. [5]

- Hinge Joint - The Hinge Joint groups together two Rigidbodies, constraining them to move like a hinge. A Hinge Joint is mostly used to allow the user to interact with doors and buttons realistically and smoothly. [5]

2.2.3 Prefabs

The Prefab system allows developers to create, configure, and store GameObjects complete with components, placement, and child GameObjects as a reusable asset for the project. Prefabs are often used when instantiating many similar assets from a script. [5]

2.2.4 Unity developer UI

The Unity user interface (UI) allows the developer to interact with the application and features a simple drag-and-drop system. The interface consist of several components: [5]

- The Scene View - An interactive view of the game environment. The scene view is used to select and position scenery, light, props, and all other types of GameObjects.
- The Game View - Renders the application the way a user would see it while playing the game. The view can easily be switched between how the user would see the game in VR (used for building and transferring the game to the Oculus Quest) and a stationary view for testing and debugging.
- The Hierarchy Window - A structured system of all the objects placed in the scene view. It also shows the different scenes used in the game.
- The Inspector View - When an object is selected either in the scene view or in the hierarchy window, the properties of the object are shown in the inspector view. The view shows the properties of the selected object and allows the developer to alter the object's functionality and settings, such as position and appearance.
- The Project Window - Contains all the imported files and assets related to the project.
- The Console View - It contains all notifications and errors created when running or building the application.

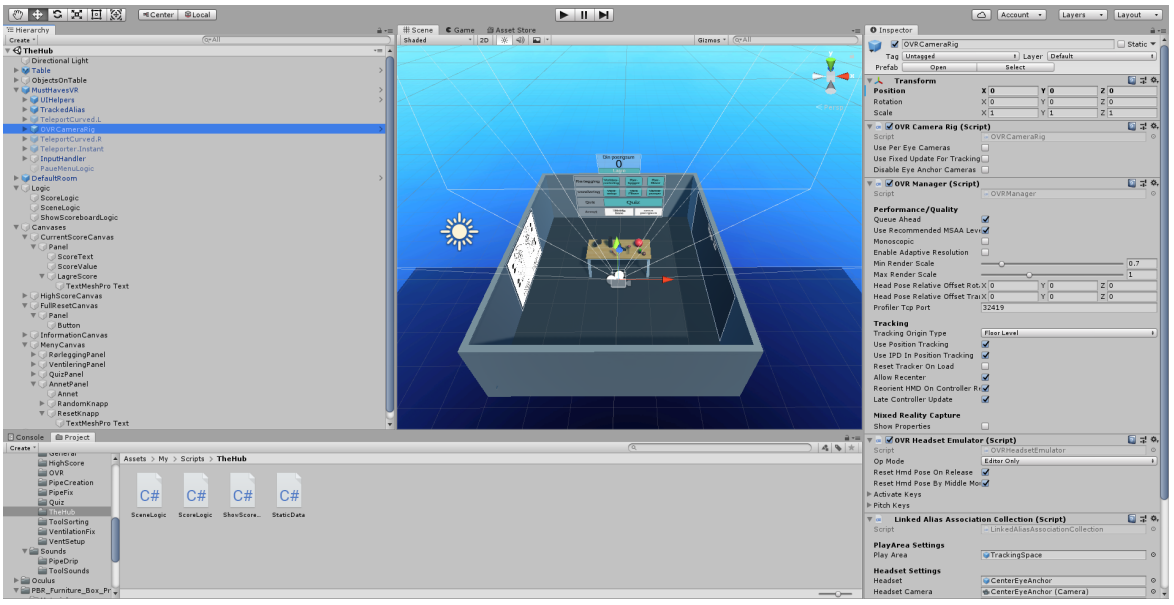


Figure 2.1: The User Interface of the Unity application with the default layout. The Hierarchy View is located on the left side, the Inspector View on the right side, and the Console View and Project View on the bottom.

2.2.5 Assets for Unity

An asset is a representation of anything that can be used in a Unity project. Examples of this include collections of 3D models, audio files, images, and code files. Assets can be imported into the unity project either through Unity's asset store or through external websites. Assets used in the development of Vocational VR include models for pipes and vents, work-tools, and other props.

The following assets have been used in Vocational VR and were downloaded from Unity's asset store:

- Oculus Integration - Unity provides built-in VR support for the Oculus Quest and other VR systems. The Oculus integration asset contains scripts, prefabs, and other resources to supplement Unity's built-in support. The asset includes an interface for controlling VR camera behavior, a first-person control prefab, a unified input API for controllers, and more. [6]
- TextMesh Pro - A replacement for Unity's UI text. TextMesh Pro uses advanced text rendering techniques along with custom shaders to create a better-looking UI text. TextMesh Pro also allows for improved control over text formatting and text layout.
- Concrete Pipes - Includes a set of pipes, both straight and curved ones.

- Workplace Tools - The workplace tools asset contains some tools with a low polygon count, making them suitable for VR projects.
- HQ Air Ducts Kit - An asset with many ventilation-prefabs, used in some minigames.
- Modern Minimalist Sofa - A simple sofa used as a prop.
- PBR_Furniture_Box_Prop - A box prefab used as electricity box.

2.3 VRTK

VRTK is a collection of useful scripts and concepts to aid in building VR solutions rapidly and easily in Unity. This includes locomotion and interactions between the user and the world. VRTK supports multiple software development kits (SDK), including Oculus. In Vocational VR, the two most important prefabs used from VRTK is the ability to pick up objects, and the ability to snap an object into a specified location once released. Such a location is called a `Snap Zone`. [7]

2.4 GitHub

GitHub is a website and a cloud-based service that offers collaboration and version control. GitHub uses git, a distributed version control system. In the development of Vocational VR, GitHub was used frequently. By providing abilities such as branching, merging, and branch history, working both separated and using the agile development technique pair programming became very easy. GitHub branching allowed the developers to work on different minigames at the same time without interfering with each other.

2.5 SideQuest

SideQuest is a PC application that makes sideloading Oculus Quest apps easily accessible for new VR developers. By providing a simple graphical user interface, Vocational VR can be installed to whichever Oculus Quest with just a few operations. To be able to use SideQuest with the Oculus Quest, the Quest has to have developer mode enabled. Sideloading Quest apps are intended for developers only, and Oculus, therefore, require the developers to create or join a developer organization on the Oculus website. [8]

Chapter 3

Introduction to Vocational VR

The main prompt of this bachelor thesis was to develop a fully functional VR game around the vocational programs plumbing and ventilation. The answer to this prompt was Vocational VR, a virtual reality game containing a leaderboard, a centralized hub, and seven small minigames that combine entertainment and education. This chapter contains the developers' description of both The Hub and all of the minigames. A video demonstration of the entirety of Vocational VR can be found on YouTube using this link (https://youtu.be/CxPB9-Rpx4Y).

3.1 The Hub

The Hub is the starting scene of the game. From here, the user can be familiarized with the environment and the controllers. By inspecting a descriptive image of the buttons on the controllers, the user will quickly learn how to interact with the virtual world, picking up objects and choosing which minigames to play.

Before entering any minigames, the user can interact with some different objects presented on a table. This, along with the descriptive image, will quickly teach the user how interactions in the virtual environment work. Most of the objects found at the table are tools and pipes that the user will use in some of the minigames. Learning how to use them before the user enters the minigames provides a smooth learning curve.

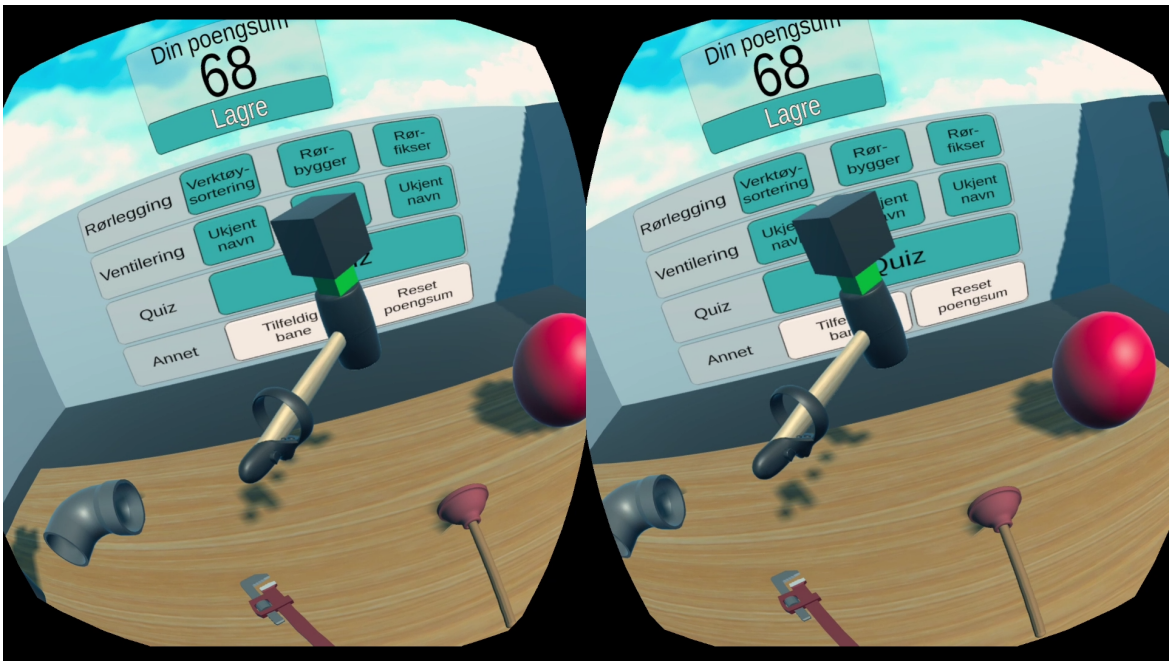


Figure 3.1: A user experimenting with physics based interactions between tools in The Hub.

In addition to choosing what minigame to play, the user can also view the current score accumulated from the different minigames. If desired, the user can save the score by typing in their name. The score will then be saved to the top 10 leaderboard, which is visible from The Hub.

The teleportation method found in most of the minigames is deactivated in The Hub. Instead, the user has a laser pointer pointing outward of the right controller. This beam allows the user to interact with non-physical objects such as canvases and canvas buttons, instead of having to walk around and push the buttons by hand. This configuration is used to choose what minigame to play, as well as to save the current score. Interacting with canvases instead of physical buttons is a common design pattern used by many big companies such as Oculus. This way of interacting with buttons focuses more on convenience instead of immersion. By using laser pointers, the user can interact with many buttons more practically and clearly, instead of walking around a big area just to push buttons physically.

All scenes and minigames are available from The Hub. When exiting a scene or a minigame, the user is returned to The Hub again. This allows the user to play whichever minigame he or she desires, instead of locking the user into a path where one minigame needs to be passed to access other minigames. Because each minigame provides the user with a score based on time spent or objects used, the user might want to replay some of the minigames to reach a better score than previously achieved. This becomes an easy task when every minigame starts and ends at The Hub.

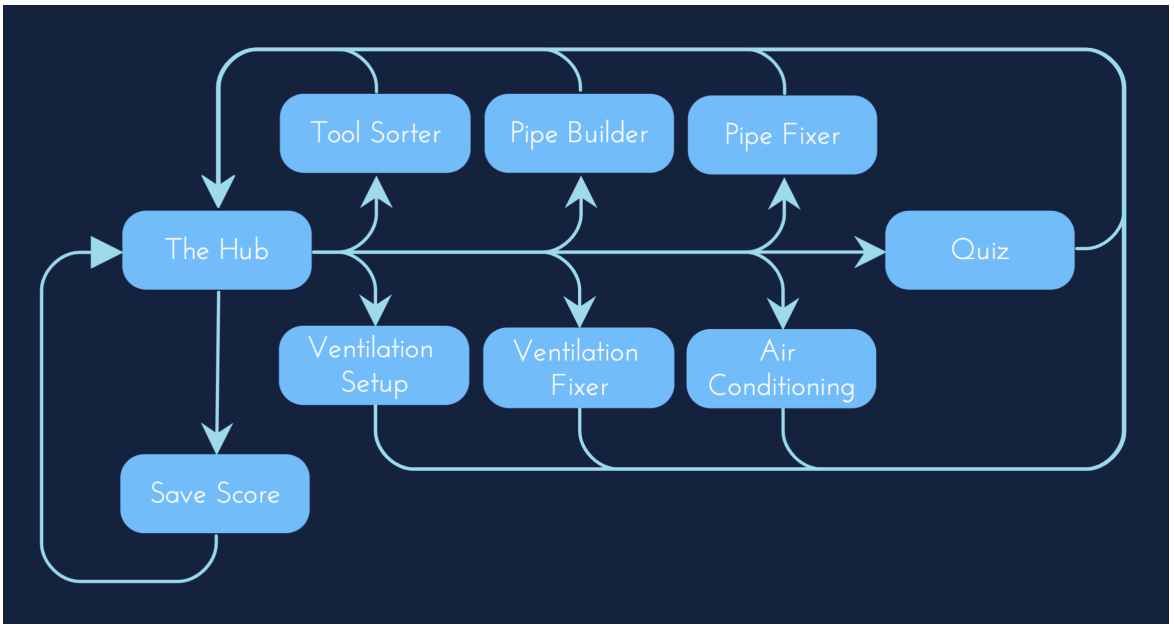


Figure 3.2: The game flow of Vocational VR

3.2 Tool-sorter

When loading into the Tool-sorter minigame, the user is presented with two tables. One of the tables contains a lot of different tools used by different professions, and the task is for the user to sort out the tools that belong to the plumbing profession. The user can read on the facing wall how many tools that belong to the plumbing profession, and receives an audible and visible alert indicating if the tool that the user moved is correct or not. The minigame runs on a timer, and points are given based on how much time the user has left when all the correct tools are moved to the empty table. The minigame then informs the user that the minigame is over and teleports the user back to The Hub after a few seconds.

A simple sorting game such as Tool-sorter might be difficult when a user tries the minigame for the first time. This is because most people do not have a clear understanding of which tools a plumber uses. Because of the design of the game-flow, the user can quickly try the minigame again to achieve a better score than last time. This means that users can play the desired minigame as many times as they would like. Instead of keeping the highest score obtained from the minigame, the user keeps the score last achieved, meaning that a replay of a minigame might result in a lower score than previously. This means that the user has to evaluate their performance and decide whether or not to play the minigame one more time based on the performance evaluation.

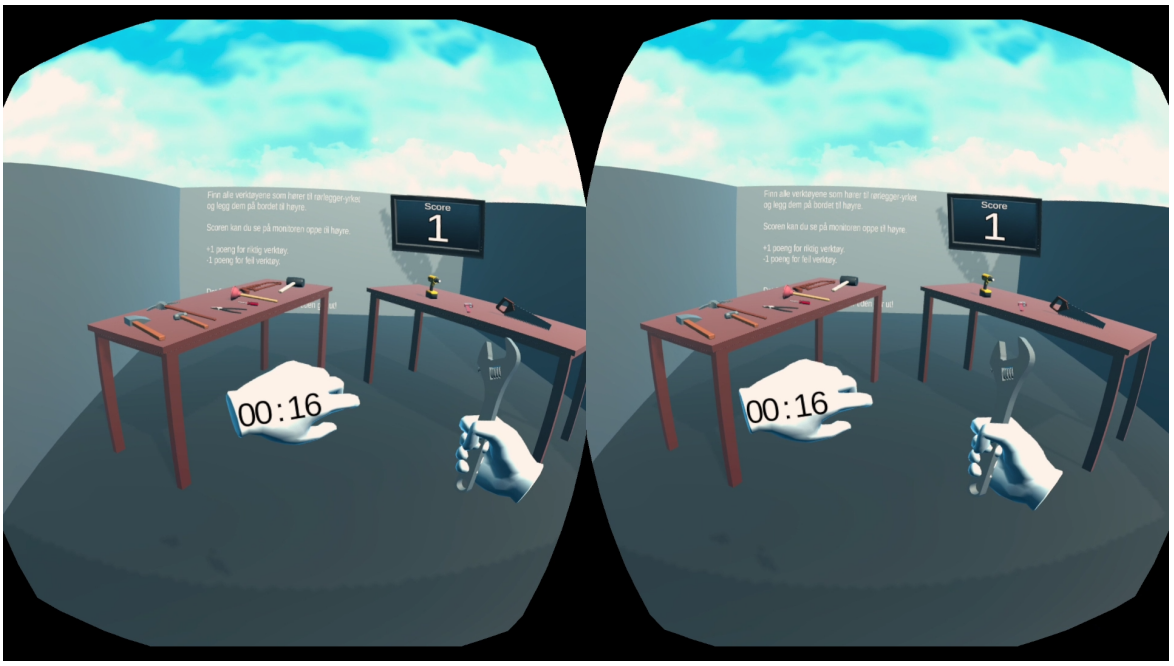


Figure 3.3: Overview of the minigame. 2 correct and 1 incorrect item is placed on the rightmost table, resulting in 1 point.

The timer is placed on the virtual left hand of the user and shows the remaining minutes and seconds. Each user has 90 seconds to complete the minigame, something that proved to be enough time during UiS open day. If the timer runs out before all plumbing-tools are found, the user returns to The Hub with zero points. If the user finds and moves all plumbing tools, and only those, the game ends with an audible notification, and the user is informed that the game is over. The user has a couple of seconds to look around before The Hub is loaded again. Because this is one of the first minigames a user might play, a simple scoring mechanism is used. Scoring more points directly by completing the game more quickly is easy to understand.

3.3 Pipe-builder

In Pipe-builder, the user is presented with a wall behind a grid of bars. At each end of the wall, two pipes are sticking out. The goal of this minigame is to connect the two pipes together by placing new pipes between them. As illustrated by 3.4, the placement of new pipes is affected by the obstacles that are blocking some of the segments shown by the grid. The user has to get an overview of where the obstacles are and plan accordingly to find the route between the pipes that uses the least amount of new pipes.

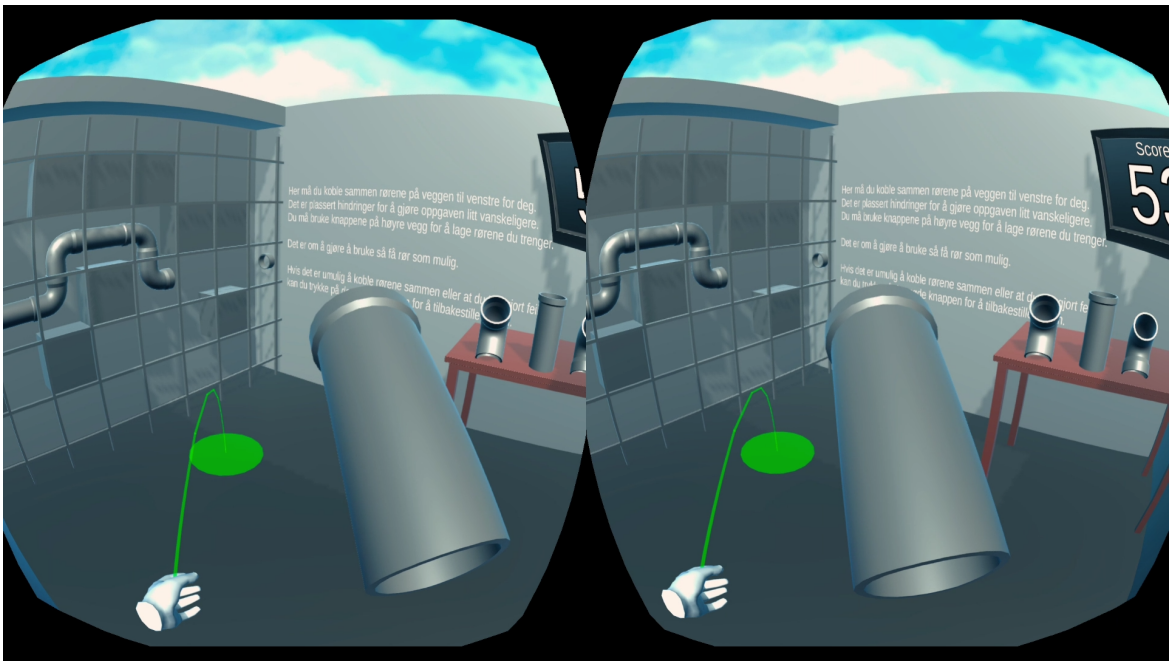


Figure 3.4: The user has placed some of the pipes necessary to complete the pipe section.

To allow the user to place pipes in pre-defined places, VRTK Snap Zone is used. By connecting the grid created by the bars with a two-dimensional array, the position of new Snap Zones can be calculated and created dynamically while the user places new pipes. When playing, this implementation of Snap Zones only allows the user to place pipes next to a pipe previously placed. It is not possible to place a new pipe that leads straight into an obstacle, as no progress would have been possible.

Along with the wall and pipes behind the grid of bars, a table, and some buttons also exist in the virtual environment. The buttons provide the user with the ability to create new pipes based on which type of pipe is needed. By having physical buttons rather than canvas buttons, the immersion is kept, as most of the actions the user can do in the game are also possible in real life.

To win, the user has to place enough new pipes such that the pipes that stick out of the walls become connected. The user starts off with 60 points, displayed on a monitor. For each new pipe that the user places, one point is subtracted from the score. This means that to reach a high score, and the user has to use as few new pipes as possible. If the best route is difficult to find, getting a high score becomes difficult.

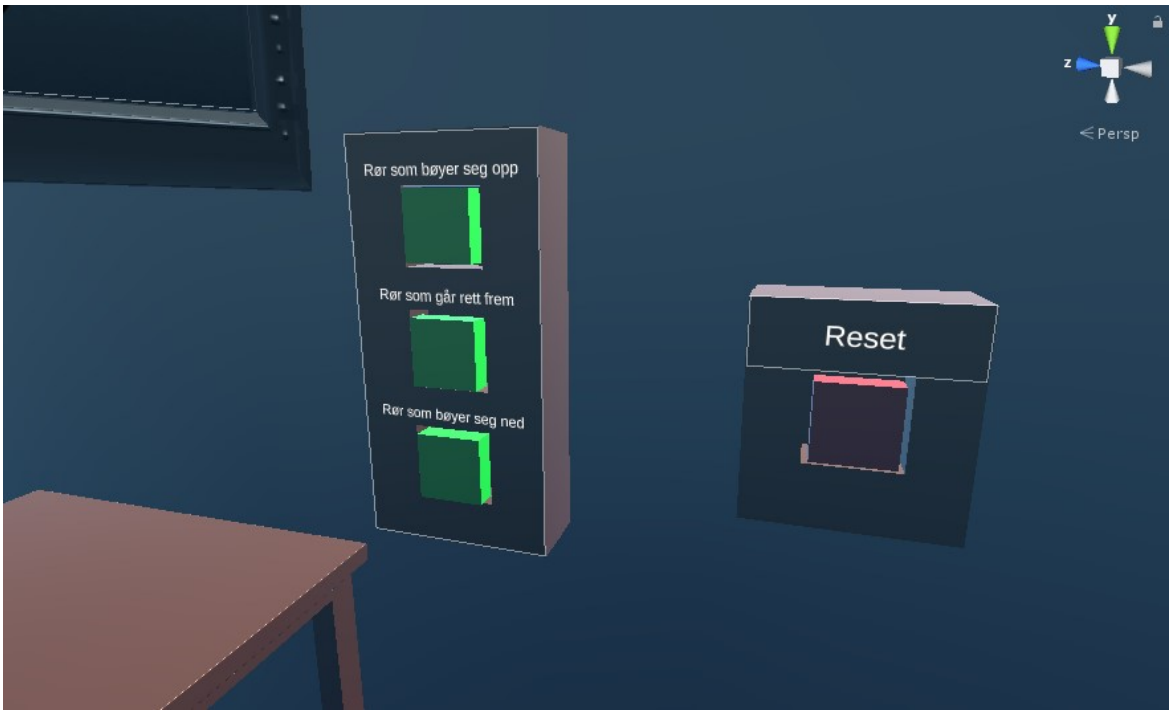


Figure 3.5: The buttons used to create new pipes and to reset the pipe wall.

To move around, the user can teleport. This design was implemented to avoid users getting nausea and motion sickness. If the implementation of moving around in virtual reality with a joystick was implemented, the connection between moving in VR and moving in real life would have been weakened. This weakening often leads to motion sickness as the movement the user sees in VR does not happen in real life. To still be able to use locomotion, teleportation was used. Teleporting quickly becomes second nature for the user. When teleporting, the screen fades to black for a split second. This means that the user does not see the change in position, avoiding the connection between movement in real life and VR to be weakened.

3.4 Pipe-fixer

The prompt of this minigame is to repair several pipes that are broken. The pipes are hidden behind a wall, and the user is presented with a pipe wrench and a hammer at the start of the minigame. Some text on a panel attached to the wall informs the user that the hammer needs to be used to break the wall covering the pipes, while the pipe wrench has to be used to fix the broken pipes.

An indication of where the broken pipes are is created by some puddles on the ground. This informs the user where the broken pipes are in the x-axis, but not on the y-axis. Also, the wall is constructed by several smaller squares, which matches the location of the pipes.

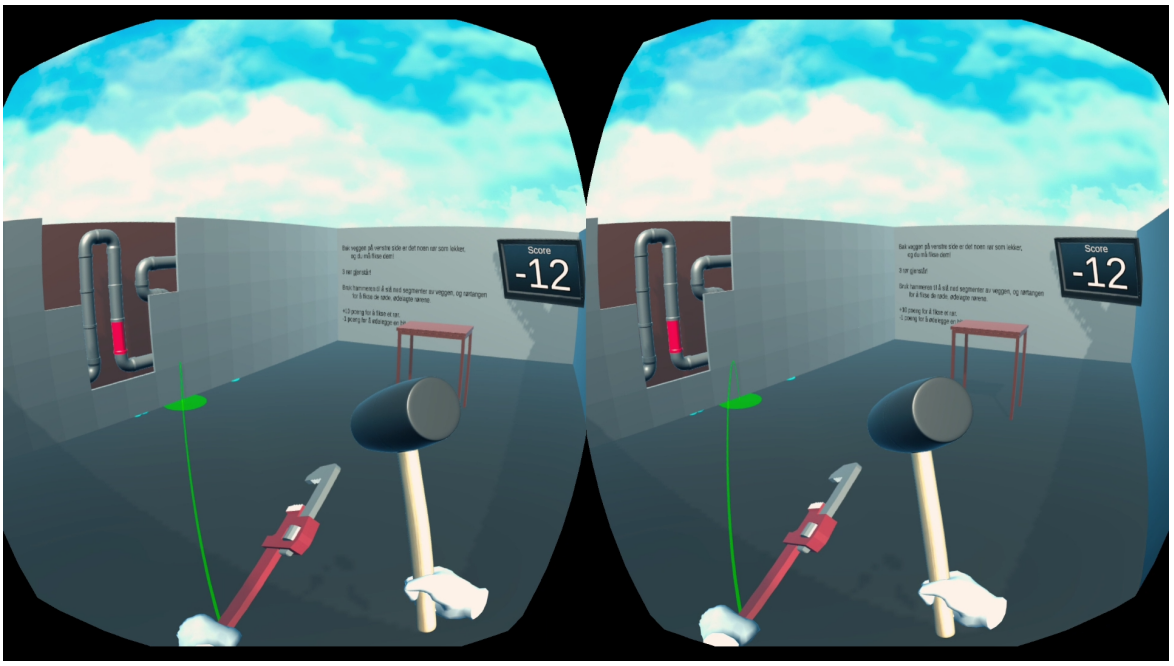


Figure 3.6: Parts of the walls are destroyed, revealing broken pipes in red.

The number of broken pipes and their location is random, meaning that the user can not obtain more points by memorizing the locations of the broken pipes. The points are given by how many wall-segments that had to be broken, subtracted by the number of pipes fixed.

3.5 Ventilation-setup

When Ventilation-setup is loaded, the player will be teleported to a room with a ventilation system. This ventilation system is not complete, so the player has to fix it. The player can read the problem on the wall in front of him.

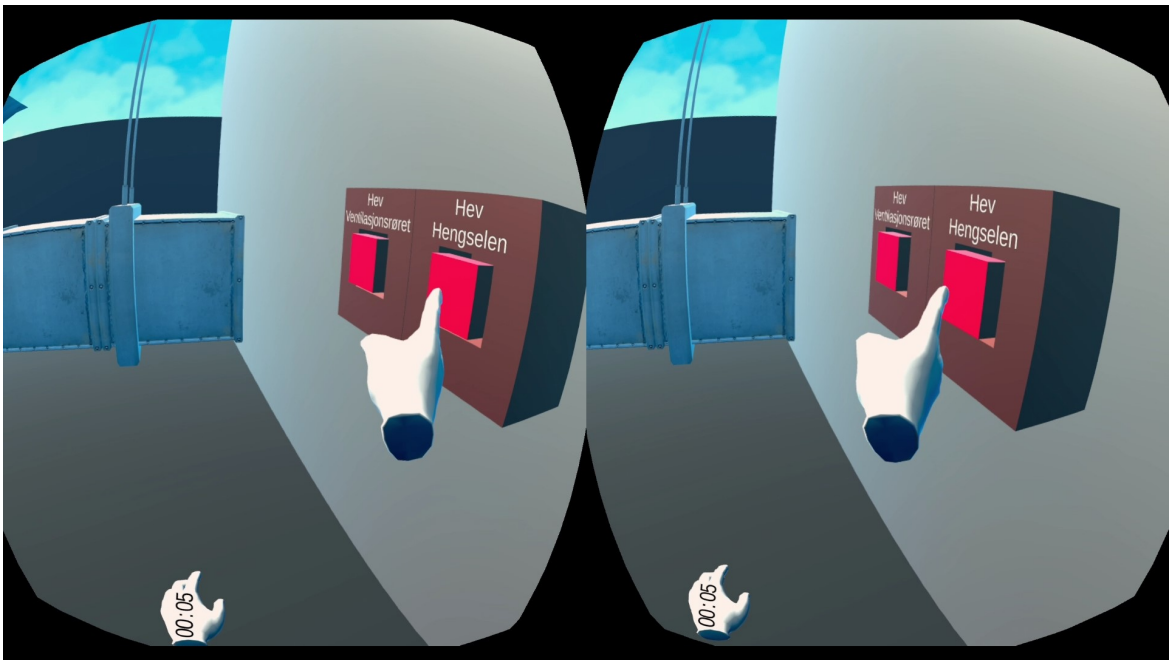


Figure 3.7: The user is about to raise the ventilation hinge.

The room contains four buttons. When a button is pressed, some of the ventilation parts will move, which part move depends on which button that the user pressed. Each button needs to be pressed once, and when all parts are in the right spot, the ventilation can be turned on. The player will know when the ventilation work because he will hear the ventilation start blowing. If the ventilation doesn‘t begin, the parts in the game are not in the right spot. The player will be teleported back to The Hub when completed. The game has a scoring system that will show on a TV screen on the wall. This score will keep track of how much the player has done. This is helpful because the player gets to know how much he has left to do before he can turn on the ventilation. The text on the wall will also tell the player when he can turn it on.

This minigame is not advanced at all. It is simple and very straight-forward. The reason for this is so that the first time users will get to know how VR works, and how to use the controllers. Ventilation-setup is also the first minigame for the ventilation category, so this is the easiest. The final score is evaluated by how fast the game is done.

3.6 Ventilation-fixer

In this minigame, the task is to fix the ventilation system. The player has to turn on the power first to get to the next level. There are three power boxes that need to be turned on and has to be turned on in the right order for it to work. The electricity boxes that are off will have a red material, and the boxes that have power have a green material.

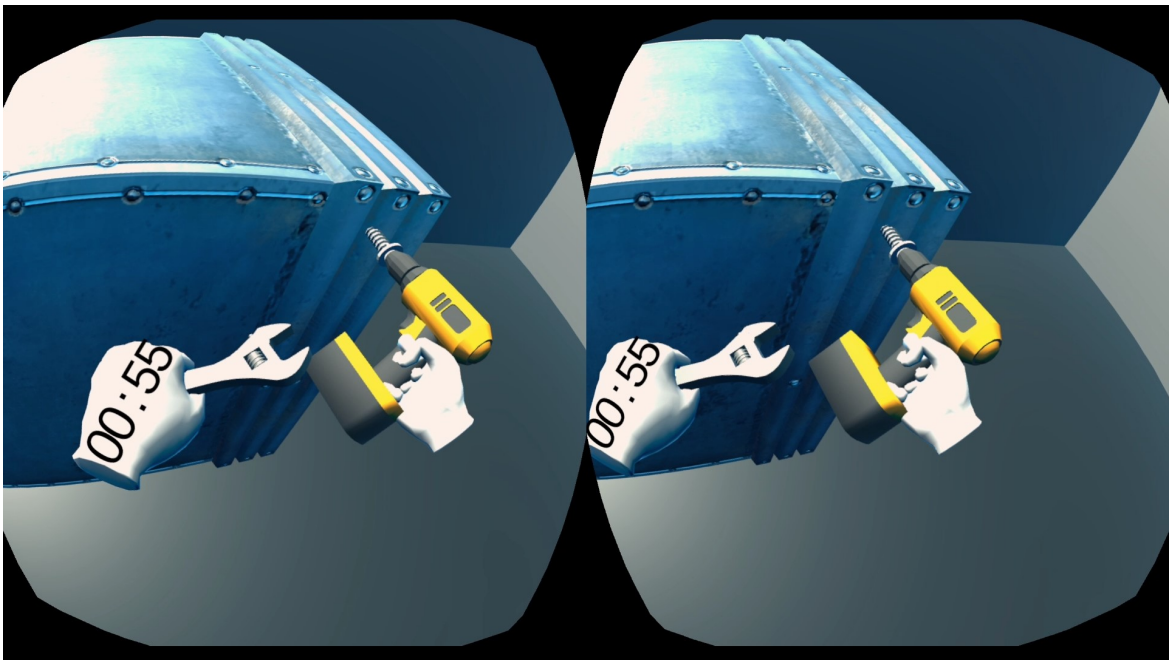


Figure 3.8: The user is unscrewing the broken part of the ventilation system.

When the power is successfully on, the player has to turn around go to the back of the room, and this is where the ventilation is. The player is presented with a broken ventilation pipe, and this needs to be fixed. The user has to use a drill and a wrench to fix it. There is on a bolt that needs to be screwed out, when the player has done that, the valve will come off and have material red. The text on the wall will be updated and tell the user that it is time to get the wrench. The tools only have to touch the objects to trigger the method that fixes the valve. When the wrench touches the valve, a new valve as a prefab will appear, and the broken one disappears. This prefab has interactions, so the user can lift it up and put in back on the ventilation system.

The player scores point for each task done, and in order to win the game, all task needs to be completed. The player will have 5 minutes to complete the game, and if the time runs out, the current score will be saved.

3.7 Air Conditioning

Upon starting the minigame, the user is placed outside a house with the front door wide open. The door indicates that the player should walk inside the house. When entering the house, the player is presented with a table containing a button, a screen showing the current score, and a panel with instructions on how the minigame works.

The minigame is divided into three steps, where the player must complete the current step to advance to the next one. The first step is to close all doors and windows that are open. This

makes the user familiar with how the house is designed and how each room functions and looks. Closing all openings also allows for better utilization of the air condition system that the user will be installed throughout the minigame. The user receives a temporary score based on the number of doors and windows closed.

When pressing the button for the first time, the inside unit of the air condition system is instantiated inside a cardboard box inside the house, and the user is instructed to place the unit on a wall so that the airflow is most optimized. The user can read some instructions on a panel close to the cardboard box to find out where the best placement for the unit is. When the inside unit is placed and the button is pressed, the user receives an additional score based on the placement of the unit, based on the choice of room, clearance under and beside the unit, and in which direction the unit faces. The outside unit of the air condition system is then also instantiated in the cardboard box.

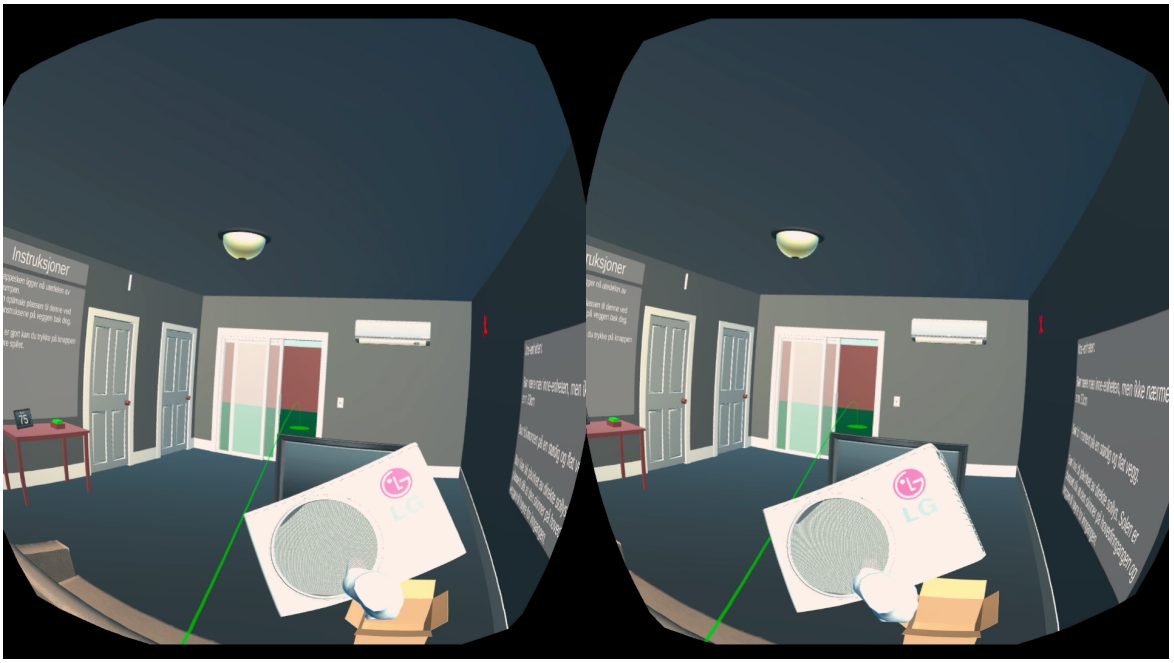


Figure 3.9: The user has mounted the inside unit and is heading out with the outside unit.

The final step that includes the outside unit is quite similar to the step with the inside unit. The user is instructed to read the guidelines on the panel over the cardboard box to decide where to place the outside unit. When the outside unit is placed, and the button is pressed again, the final score will be presented. The final score consists of the previous temporary score, in addition to the score obtained from the placement of the outside unit, which is based on distance from the inside unit as well as if it is facing the sun or not.

3.8 Quiz

The quiz consists of 14 questions regarding plumbing and ventilation. The user can answer either true or false, and is awarded five points for each question answered correctly. The questions increase in difficulty and are designed so that no users should answer correctly on all questions the first time playing. The questions consist of real-life information about tools and professions that plumbers and ventilators have to know.

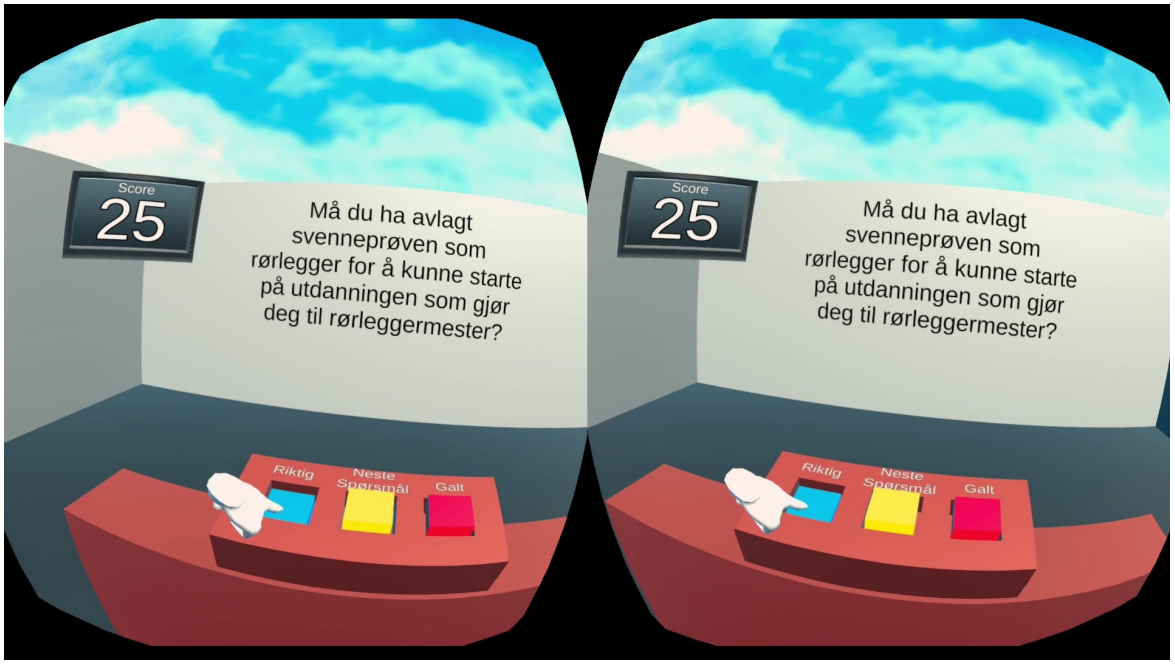


Figure 3.10: The user believes the statement is correct, and is about to press the "Correct" button accordingly.

Chapter 4

Structure

4.1 General scene design

Vocational VR consists of The Hub and several minigames. All minigames contain a game area where the progression of the game takes place and a pause area where the user can choose to resume the minigame or head back to The Hub. This also includes that all minigames contain a couple of scripts and logic used to allow for basic functionality, such as pausing the game and leaving the minigame. The scenes also consist of basic figures such as two square rooms, the logic necessary to interact with the virtual world in VR, lighting, and a canvas. Because every scene consists of these fundamental GameObjects, they have been grouped into the prefab named `RoomsAndVR`. This means that whenever a new minigame is to be created, the inclusion of `RoomsAndVR` in the scene will provide the necessary GameObjects to start working on the specifics of the minigame immediately. By changing GameObjects or script inside the prefab, all instances of the prefab will be changes. In Vocational VR regarding the `RoomsAndVR` prefab, this means that all scenes containing the prefab will be updated.

One exception to this prefab is GameObjects that need to be kept in memory across scenes. In Vocational VR, this applies to the script `GenerateJsonInfo.cs` contained inside the `JsonLogic` GameObject. `GenerateJsonInfo.cs` operates as a singleton pattern, meaning that there should not only exist more than one instance of the class at all times. The script handles reading a JSON file containing all text displayed in the minigames and stores is as a corresponding class. By keeping the GameObject containing the script in memory and implementing the singleton pattern, the JSON file is only read once, saving resources. For this to work, Unity requires the GameObject to be placed as a root-GameObject inside the hierarchy window.

In the game area, the user can interact with objects and the environment with virtual hands. The virtual hands appear and function as hands in the real world work. In the pause area, the

hands are replaced with virtual Oculus quest controllers, indicating that the pause room is not meant to simulate real-world experiences. In the pause room, the user is not able to interact with any physical objects but is rather allowed to interact with 2D elements. This interaction is designed by having a light beam appear out of the right controller. When the beam points to a 2D button and the user use the trigger button on the controller, the 2D button is selected.

`Countdown.cs` is another important script located inside `RoomsAndVR`. The script provides a basic countdown clock with capabilities of counting both upwards and downwards, as well as being paused by external scripts. `Countdown.cs` is used in three of the seven minigames.

4.2 The Hub & Save Score

The Hub extends the `RoomsAndVR` prefab with a couple of canvases and a table containing some interactable props. Because The Hub is not directly a minigame itself, the pause room has been removed. Instead, the pause room controller functionality has been set as the default functionality, while also allowing the user to interact with physical objects. This allows The Hub to primarily function as the access point for all minigames, but also familiarize new users in how to interact with objects. The Save Score scene operates in a very similar way to The Hub. Save Score uses the pause room functionality by default, and operates as an extension of The Hub to allow the user to save the current score to a leaderboard.

To keep track of the accumulated score the player earns throughout the minigames, a static class named `StaticData.cs` is used. To use scripts as components inside GameObjects, Unity requires the script to extend the base class `MonoBehaviour`, meaning that they can be instantiated. Because a static class cannot extend `MonoBehaviour`, the class cannot be associated with a GameObject. In return, this means that `StaticData.cs` will never be instantiated and destroyed when changing scenes, and can, therefore, contain the accumulated score of the player.

Because Vocational VR is designed to run on Andriod, storing information between sessions has to be handled a bit differently compared to applications running on a computer. Keeping track of the local high scores is therefore done with a class from Unity named `PlayerPrefs`. This class is capable of storing and accessing floats, ints, and strings, and allows new Unity developers to easily store values between sessions. The two-dimensional array leaderboard can be parsed into a single string with separators between the values. This string can then be stored as a PlayerPrefs and parsed back to an array when called upon. This is done inside `ScoreLogic.cs`.

4.3 Tool-sorter

Each tool in Tool-sorter is created as modifications of a prefab from VRTK called `Interactable.Primary_Grab.Secondary_Swap`. The prefab consists of a white cube mesh and several scripts that combined provide grabbing functionality. To create the tools, the white cube and its collider were replaced with tool meshes from the asset store, as well as multiple smaller colliders to match the look of the tools. The prompt of the minigame is to find the tools that belong to the plumbing profession and move them over to the empty table. This is done by tagging one collider in each tool with either a `Valid` or `Invalid` tag. Only one collider in each tool is tagged to prevent multiple registrations of the same tool. The empty table contains a box collider set to `Is Trigger` and extends three cm above the table. Once a tool enters the collider, a call to the method `OnTriggerEnter` is sent. This method checks if the tool contains a collider with either the `Valid` or `Invalid` tag, and notifies the user accordingly. If the user removes a tool from the empty table, `OnTriggerExit` is called upon.

The countdown script inside `RoomsAndVR` is in Tool-sorter set to count downwards from 90 seconds. Once the timer reaches zero a whistle will be heard, and the user will be teleported back to The Hub.

If the user succeeds in finding all the plumbing tools, a short fanfare will be played, indicating that the user finished the minigame. The countdown timer will stop, and the remaining seconds will be turned into the score.

`CheckValidTools.cs` is the main script in Tool-sorter, keeping track of the score. When tools enter or exits the empty table, a script inside the table-GameObject sends information to the `Change` method inside `CheckValidTools`. The information tells the method if the GameObject entered or exited the collider and the name of the GameObject.

4.4 Pipe-builder

The Pipe-builder minigame consists of a much smaller room compared to the other minigames. The prompt of this minigame is to fill in a gap between two pipes with more pipes, connecting the system. To indicate that a section of the wall is removed to view the pipes, a grid of rectangle GameObjects. This is done dynamically by the WallGrid script to make the grid dimensions match the pipe gap.

The main script, `SnapZones.cs`, contains the necessary logic to create obstacles between the pipes and to create Snap Zones for the pipes the user is going to place. To keep track of all the possible locations, a pipe can be placed, and a two-dimensional array is used. The array contains the integer values 0, 5, and 10, indicating an open section, a section with an obstacle,

and the goal, respectively. The obstacles are added on a random basis, but will never obstruct the area next to the start and stop pipes. Because the obstacle-placement is random, situations that prevent the user from completing the minigame might occur. This is handled by a reset button included in the scene. When the minigame starts, three different types of Snap Zones are instantiated. One for each type of pipe the user can place. If an obstacle stops the user progression the next turn, the Snap Zone that will cause the problem will not be instantiated. After a pipe has been placed, the Snap Zones not used will be deleted, and new Snap Zones will be instantiated at the end of the newly placed pipe. When a Snap Zone that leads to the end pipe is used, the game finishes, giving the user a score based on the number of pipes used.

The Snap Zone pipe prefab used contains a script named `CallSnapZones.cs`. When a pipe is placed inside the Snap Zone, the script calls a method inside the Snap Zone script, informing it that a pipe of a specific type has been placed and that new Snap Zones needs to be instantiated.

The user can instantiate the desired type of pipe by pushing the corresponding button. The scene also includes a trash bin. If a pipe enters the trash bin, it will be deleted.

4.5 Pipe-fixer

In Pipe-fixer, the user is met with a pipe wrench and a rubber mallet. With the rubber mallet, the user is going to break down segments of a wall to reveal a pipe system. The pipe system contains some broken pipes, and the minigame prompt is to fix these pipes with the pipe wrench. When the scene loads, `GenerateWall.cs` creates a wall made of several smaller wall segments. To tell the segments apart from each other, a slightly different grayscale color is used on each segment. Each wall segment contains the script `BreakWallPart.cs`. When the rubber mallet hits a wall segment with the required power, the script will turn the segment into nine smaller pieces, as if the hammer destroyed the wall segment.

When instantiated at the start of the minigame, `CreateBrokenPipes.cs` randomly selects some of the pipes and marks them as broken. Each pipe has a 10% chance of being broken, resulting in a total of six broken pipes on average. When a pipe is marked as broken, a water puddle prefab is placed on the floor under the broken pipe. The water puddle is visible without breaking the wall covering the pipe and helps the user decide where to look for broken pipes.

The pipe wrench GameObject contains the script `FixPipe.cs`. When the collider to the pipe wrench collides with a broken pipe, the broken pipe becomes normal, and the water puddle is destroyed. `FixPipe.cs` then updates `ProgessOverview.cs`, informing that a broken pipe has been fixed. When all broken pipes are fixed, `ProgressOverview.cs` ends the minigame, awarding the user a score based on the number of wall segments broken subtracted by the number of pipes fixed.

4.6 Ventilation-setup

Each ventilation part is from the HQ Air Ducts asset. These parts are not interactable, and can only move by using the buttons in the game. The ventilation system is not complete, so the prompt of this minigame is to complete the ventilation and turn the ventilation on. When a button is pressed, a new `Vector3` will be created with a new position. An `IEnumerator` is used to animate the movement of the ventilation parts. The `IEnumerator` takes a `Vector3` and a `float` duration as arguments. These arguments define the change in position and the duration of the change.

The player has a countdown timer on its left hand, which uses the countdown script in `/RoomsAndVR/Logic/CountDown`. This will create a timer counting down from 30 seconds, and the player will hear the clock ticking. If the player fails to complete the ventilation, a coach whistle will be played, and the player is teleported back to The Hub.

If the player successfully manages to complete the ventilation system in 30 seconds, a victory fanfare will be played, and the player is teleported back to The Hub with the achieved score. The score will be added to the current total score of the player.

`ScoreView.cs` keeps track of the score in the game. The script checks the score of the player each time a button in the scene is pressed and will update the instructions based on the progress of the player. The script also updates the score of the player with each button press and is displayed on the TV included in every minigame.

4.7 Ventilation-fixer

The tools used in this minigame are some of the same tools found in Tool-sorter. In the previous minigame, Ventilation-setup, the user turned on the ventilation system. In the real world, ventilation systems often get some wear and tear. In Ventilation-fixer, the prompt is to turn on the power that has been turned off and then fix a part of the system that has broken. The script `TurnOnPower.cs` is used to check if the player successfully presses the buttons in the correct order to progress in the minigame. When this happens, the player is awarded a score and can proceed with fixing the broken ventilation part.

`BoltAction.cs` is then called which instantiates a screw prefab. When the drill located in the scene is held against the screw, the GameObjects' colliders collide. `OnTriggerStay()` will then move and rotate the screw as if the drill is unscrewing it. This movement is based on a defined speed multiplied by the time interval between each update.

When the screw has finished its movement, the ventilation becomes detached from the ventilation system and changes color to red, indicating that it is broken. When the wrench prefab collides with the broken ventilation part, `OnTriggerEnter(Collider other)` is called. The method destroys the broken ventilation part and instantiates the fixed version of the ventilation part. The fixed part is attached to the ventilation system. When this is done, the player will hear a victory fanfare and is teleported back to The Hub with the accumulated score from the minigame.

The minigame is time-based. If the player does not fix the ventilation within 90 seconds, there will be no score awarded and the player will be teleported back to The Hub.

4.8 Air Conditioning

The Air Conditioning minigame removes the game room from `RoomsAndVR` and replaces it with a house GameObject surrounded by fences. The prompt is to install a split system air conditioner to the house. The progress of the minigame is contained inside the Progress.cs script and is divided into multiple steps. Each step is separated into two parts. The first part completes the previous step by calculating the score of the previous step. The other part instantiates objects and information to be used in the current step or notifies the user that the minigame is completed.

The first step instructs the user to close all doors and windows in the house. The windows are based on the VRTK prefab `DirectionalJointDrive`, which allows a GameObject to slide across a specified distance in a given direction. The doors consist of a Hinge Joint component, limiting the movement of the door to only rotate around a specified pivot point. The two scripts `DoorGrabber.cs` and `FollowPhysics.cs` allows the user to interact and naturally move the doors without any jitter. GameObjects containing a collider and `CheckIfClosed.cs` is used to validate if the doors and windows are closed. When the collider of a window or a door is inside the collider, the `isClosed` boolean changes to true. The number of `isClosed` booleans set to true decides the score the user accumulates.

The second step instantiates the indoor component of the air conditioner and updates the informative text with instructions on how to place the component. A set of guidelines on a canvas explains to the user where the best location for the indoor component is. The user can choose between multiple Snap Zones places around the house, and each Snap Zone contains `InConStats.cs` and `UsageStatus.cs`. `InConStats.cs` contains information on how well the Snap Zone is placed in regards to the placement instructions. `UsageStatus.cs` consists of the boolean `isUsed` and is true when the indoor component of the air conditioner is placed in the Snap Zone. `UsageStatus.cs` is also used in the outdoor component of the air conditioner. The score achieved form this step is the sum of each placement factor multiplied with the importance

of the factor. These factors include the choice of room, the clearance on both sides of the Snap Zone, and the clearance underneath the Snap Zone.

The third step instantiates the outdoor component of the air conditioner and updates the placement instructions just as the second step. Instead of `InConStats.cs`, the outdoor Snap Zones contains `OutConStats.cs`. The factors used to calculate the score from the placement of the outdoor component includes the distance away from the indoor component and if it is facing the sun or not. When both the indoor and outdoor components are placed, the user receives a score from the third step and is presented with the total accumulated score from all the steps combined.

4.9 Quiz

When the quiz scene is loaded, a list containing all questions, answers, and explanations is created, with parsed information from `InfoText.json`. This task is performed by `QuizManager.cs`, which is also used to monitor the state of the game with the boolean `waitingForAnswer`. When the user presses one of the buttons in the scene to make a guess, `UserGuess(bool choice)` is called, and `waitingForAnswer` changes value. Five points is awarded to the player if the answer matches the answer in the list.

The `NextQuestion()` method is called when the user wants the next question to be displayed. Because the quiz-list is in the format {question, answer, explanation}, `NextQuestion()` uses a for-loop which increments its counter by three each time. When the counter equals the length of the list, the minigame is finished. The victory fanfare will be played and the user is teleported back to The Hub together with the accumulated score.

Chapter 5

Discussion

5.1 Feedback

5.1.1 UiS Annual Open Day Feedback

During UiS annual open day, visitors and students from high schools had the opportunity to test the two first minigames developed. The minigames tested was the Tool-sorter and an early version of the Pipe-builder. Each tester was asked to complete a survey after testing the minigames. This feedback was used to improve the game. Most of the testers tried Tool-sorter. This was because Tool-sorter was at the time the only fully developed minigame. Tool-sorter does have a low skill floor, meaning that all users should manage to complete the minigame on their first try. This resulted in feedback where the most noticeable response was that the game was not very challenging.

Several users had a hard time figuring out how to use the Oculus Quest controllers properly. This was later fixed by using an instruction panel in The Hub with an overview of what each button on the controllers does. From the feedback, almost 80% of the users were already familiar with VR. Because of this, the functionality of the controllers was changed to match popular VR games. The feedback also told the developers that the entertainment value of Vocational VR was a bit low. This was taken into consideration by allowing the player to explore a house freely (in Air conditioning) and to break walls with a hammer (in Pipe-fixer).

The developers received a response that the design was a bit boring, which could have been resolved by 3D modeling props and objects. This idea was discarded as the developers wanted to focus on the scripting part of the application, instead of modeling.

5.1.2 Meeting with the stakeholders

The developers had only one meeting with the stakeholders. The meeting was used to plan the prompt of Vocational VR, and for the developers to receive an Oculus Quest for development purposes. The meeting consisted of an open discussion of what the stakeholders wanted the VR game to achieve. The stakeholders did not have any specific ideas regarding problems and challenges that the user of Vocational VR should solve. This resulted in the developers having to create tasks for vocational occupations without having any prior knowledge of the plumbing or ventilation profession.

There have been several meetings with the supervisor during the development of Vocational VR. These meetings have been helpful for the developers and have resulted in ideas becoming refined ideas, and a more formal structure of the bachelor thesis.

Looking back at the process, there should have been more communication between the stakeholders and the developers. This would have provided the developers with valuable insight into which directions the stakeholders wanted Vocational VR to take during development.

Due to the COVID-19 virus, testing the game with the stakeholders was not an option. Before the virus outbreak, visitors at UiS and the supervisor got to test the game once. Only the developers themselves had the opportunity to test the application during the outbreak, and the most important development stage of Vocational VR.

5.2 User movement in VR

When developing a VR application, the developers have to decide how the user should move inside the game. Some games use the joystick on the controller to move, and others use teleport. Vocational VR uses the teleportation system. Although teleportation breaks the realistic immersion of VR, the chance of becoming motion sick is a lot lower compared to direct movement with the joystick. Teleportation is also very convenient when the user wants to move to a new location fast.

When using the joystick to move around, the VR representation of the user moves while the actual user stands still. The body in the game is moving, but the human body does not feel the sense of the movement. This will make the user motion sick. The central nervous system causes motion sickness when it receives conflicting messages from the sensory systems [9]. The teleportation system fades the screen to black when the teleportation takes place. This results in the user not directly seeing the teleportation taking place, and thus avoids becoming motion sick.

5.3 Frame Rate

The number of frames per second (FPS) in a VR game is an essential factor when it comes to motion sickness. If the FPS is significantly lower than the refresh-rate of the VR screens, the user will experience the virtual environment as unresponsive. This can cause the player to experience the VR environment lagging behind the user movement. The displays in the Oculus Quest has a refresh rate of 72 Hz. This means that for Vocational VR to provide the best user experience, a minimum of 72 FPS has to be maintained. This will prevent the user from seeing unrendered parts of the scene when quickly rotating. This is achieved by building each scene out of GameObjects, prefabs, and assets with a low polygon count and simple materials.

5.4 Choice of Tools

Unity

When developing VR games, there are two leading platforms to use; Unity and Unreal. Vocational VR was developed using Unity due to VR frameworks such as VRTK and Oculus Integration, and the recommendation from the supervisor. The Unity game engine stands for around 70% of all VR applications. Unity has been successfully used in earlier bachelor projects, making it the clear choice of the game engine. Unity relies on a large number of tools and concepts not found in other game engines or IDEs. Because of this, the start phase of Vocational VR was used by the developers to learn how to use Unity efficiently. By developing a virtual reality game within five months, the developers can clearly say that Unity is a well-working platform for VR-development.

C#

Unity supports both JavaScript and C#. JavaScript used to be the primary language to use, but Unity has, in later years, started to discontinue the support for JavaScript. The developers found C# to be an easy language to write both simple and complex scripts. Because C# is very similar to Java in multiple ways, the developers quickly learned how C# was designed without any prior knowledge of the language.

5.5 VR Equipment

When the development of Vocational VR started, UiS had an HTC Vive at their disposal. HTC Vive is a VR headset that uses a computer to run the games. HTC Vive is one of the most

powerful tethered VR headsets on the market and is capable of running resource-demanding games smoothly when connected to a powerful computer. Because Vocational VR mostly uses low polygon assets and few parallel processes, such a powerful VR setup was not needed. Early in the development process, Karrierelaboratoriet informed the developers that they had an Oculus Quest, which could be used instead of the HTC Vive. As Oculus Quest is standalone and does not require any external sensors or computers, the choice of VR headset landed on the Quest. Oculus Quest can track the player's movement without any external sensors, which allow the user to move around in the real world more freely.

The Oculus Quest does not offer the same compute power as the HTC Vive, but because Vocational VR is developed with performance in mind, the game runs smoothly.

5.6 GitHub Collaboration

During the whole development of Vocational VR, GitHub has been used to create a steady and unproblematic development flow. Before the virus outbreak, the developers worked together using the agile software development technique pair programming. One of the developers wrote scripts and designed the VR game, while the other developer reviewed both the code and the design. The roles switched frequently.

During the pandemic, the developers had to change the workflow to adapt to the new situation. GitHub branching was used to allow the developers to work on different minigames separately without interfering and overwriting each other's code. By merging the branches into the master branch, steady development progress was sustained.

5.7 3D Models

When designing Vocational VR, the developers had to decide how each scene should look. This included the decision of how the 3D models used should be made. Creating 3D models is not possible directly from Unity, which only provides basic shapes. Because Vocational VR is a bachelor thesis in computer science, the developers decided to import most of the 3D models used instead of creating them themselves. Most 3D models have been downloaded from Unity's asset store. The rest of the models has been downloaded for free through the third-party websites https://free3d.com/ and https://www.turbosquid.com/.

5.8 SideQuest

SideQuest was used to transfer the Vocational VR APK file from the computer to the Oculus headset. Sideloading can also be done through the computer terminal, but SideQuest simplifies the process and provides a clean GUI. SideQuest was also used as it also functions as a file browser for the Oculus Quest. Oculus Quest has a built-in screen recorder. SideQuest was then used to transfer the recorded videos from the Quest to the computer to create a video demonstration of Vocational VR.

Chapter 6

Conclusion and Future Directions

The goal of this thesis was to develop a virtual reality game for Karrierelabratoriet. The VR application aims to engage students with academic content in a more entertaining way than learning about it in school. By presenting vocational careers as minigames in a virtual reality program, students both get a chance to see and experience tasks that such careers might come across. The hub was created to make users new to VR comfortable and to allow the player to choose which minigame to play. The six minigames were created to challenge the player at multiple different levels of difficulty. The quiz was designed to test the player's knowledge within pipes and ventilation. The questions range from very easy to comically difficult.

Visitors of UiS annual open day gave the developers valuable feedback. The feedback received reflected how the final product would be perceived by new players and was therefore very important to take into consideration. The feedback indicated that the minigames made so far was well designed, but a bit boring and not challenging enough. This shifted the developers' focus, which can be seen in the later minigames. Some minigames are difficult, while some are easy and entertaining. This combination creates a well-balanced mix of entertainment and educational content.

The Oculus Quest setup is far less complicated than the HTC VIVE setup. This, in addition to being standalone, makes the Quest very portable and can easily be brought along to events and meetings with students.

6.1 Evaluation

6.1.1 Personal Evaluation - Håvard Godal

I always enjoy learning new skills. Throughout my three years as a computer science student, I have learned a lot about different types of programming languages and their different uses. I have been working on creating both websites and desktop applications. Some of the things I did not learn through my courses was game development and the programming language C#.

When deciding which bachelor thesis to apply for, virtual reality game development was the first thesis that caught my eye. Something so new as VR combined with a new programming language was very intriguing.

When diving into the new world of Unity and virtual reality, I quickly noticed that a lot of important VR functionality was already developed and available as assets. This meant that in the process of development, we could fully focus on the content of the game instead of worrying about head-tracking and controller-positioning. During the development of the final product, I have used multiple techniques and solutions learned in previous courses to progress through the development stages.

The process of developing a VR game from start to finish has been very fun and educational for me. The goal of our thesis was to provide students with an entertaining and educational way of learning about plumbing and ventilating. With the feedback from UiS open day and the reaction from Karrierelaboratoriet regarding our video demonstration, I would say that we succeeded.

6.1.2 Personal Evaluation - Kjetil Svihus

In my semester in Australia, I had one subject that was about game development. I liked this subject very much, and therefore decided that I wanted my bachelor to be about something similar. I did not have much prior experience with VR development, so I did not know how this would go, but I am very pleased with our final product.

I was a bit concerned about how many resources there were available. But after discovering Unity and its assets store, this was not a problem. Our approach to the development process was to step by step to learn Unity, which both of us did not have any prior experience with.

I think this process has been very fun and educational. Also valuable because developing a product for a client will be relevant in the future.

6.2 Future Directions

Vocational VR was created with modularity in mind. This means that each minigame is it's own separate scene and functions independently from other minigames. As a result, the game can easily be expanded with more minigames from other professions such as the electrician profession. Many of the developers' ideas had to be abandoned due to the lack of focus on 3D modeling. By creating custom 3D models, Vocational VR can be further developed to increase the feeling of realism.

Appendix A

Application Setup

A.1 Playing Vocational VR

The Oculus Quest is required to play Vocational VR. To import a game to the Quest that does not exist in the Oculus Store, developer mode has to be enabled on the Quest.

A.1.1 Oculus Quest developer mode

To enable developer mode on the Oculus Quest, the account used on the Quest has to be connected to an Oculus organization. This is done through the Oculus website (https://dashboard.oculus.com/organizations/create/). Once the account and the organization are connected, the developer mode can be turned on from the Oculus mobile application.

A.1.2 SideQuest

As Vocational VR is not in the Oculus Store, the application has to be sideloaded from a computer to the Quest. This can be done through SideQuest. Once the Oculus Quest is connected to a computer, USB debugging has to be allowed between the Quest and the computer. This can be accepted inside the Quest. When debugging is enabled, Vocational VR can be imported through SideQuest to the Oculus Quest by simply dragging the APK file into the SideQuest GUI. `VocationalVR.apk` can be found at the Vocational VR GitHub repository (https://github.com/HGodal/EdutainmentVR/blob/master/VocationalVR.apk).

A.2 Modifying Vocational VR

To further develop Vocational VR and include implementations and changes, the appropriate Android SDKs and Unity are required.

A.2.1 Android SDK

The Oculus Quest is using Android 4.4 as its operating system. This means that some specific android SDK tools are necessary to build the Vocational VR Unity project as an Andriod application. The SDK tools can be downloaded through Android Studio, the official integrated development environment for Google's Android operating system. Android Studio can be downloaded through the Andriod Studio website (https://developer.android.com/studio). The required Android SDK programs that have to be downloaded through Andriod Studio are `Android 4.4 (KitKat)`, `Android SDK Tools` and `Android SDK Build-Tools 30-rc4`.

A.2.2 Unity

Vocational VR is developed with the Unity Editor version 2018.4.17f1. The specified Unity version can be downloaded through the Unity Hub application. The Unity Editor needs to have the Android Build Support module added, which also can be done through Unity Hub.

Inside the Unity Editor, the default build settings have to be changed from PC, Mac & Linux Standalone to Android. This process takes a while. Inside the Android build setting, texture compression has to be set to `ASTC`, and the build system has to be set to `Internal`.

Unity Hub can be downloaded from the Unity website (https://unity3d.com/get-unity/download).

A.3 Sites

GitHub Source code - https://github.com/HGodal/EdutainmentVR

YouTube Video Demonstration - https://youtu.be/CxPB9-Rpx4Y

Vocational VR APK file -
https://github.com/HGodal/EdutainmentVR/blob/master/VocationalVR.apk

Unity Hub download page - https://unity3d.com/get-unity/download

Android Studio download page - https://developer.android.com/studio

SideQuest download page - https://sidequestvr.com/setup-howto

A.4 Unity Project

The project in its entirety can be found at the GitHub repository for the application (https://github.com/HGodal/EdutainmentVR/tree/master).

A.5 Scripts

All scripts developed and used in Vocational VR can be found inside the script folder on GitHub (https://github.com/HGodal/EdutainmentVR/tree/master/Assets/My/Scripts). The only exception being the `InfoText.json` which is placed inside the resources folder (https://github.com/HGodal/EdutainmentVR/tree/master/Assets/Resources). The resources folder is used for creating instances of assets in a scene that cannot be attached to GameObjects. The scripts mentioned in A.5.1 are scripts used in multiple scenes of Vocational VR and is included inside the `RoomsAndVR` prefab.

A.5.1 General

CommonLogic.cs - Contains logic to wait for a given amount of seconds before changing the scene.

Countdown.cs - Consists of a counting script. The developer can decide the number of seconds given to the user, the direction of the counting, and whether or not the counter is paused.

DisplayText.cs - Operates as an intermediary between other scripts and a canvas with the `TextMeshPro` component, providing simple methods for appending or overwriting text.

GenerateJsonInfo.cs - A singleton pattern script capable of reading the value of a variable in `JsonInfo.cs` based on the input from the developer.

JsonInfo.cs - A C# representation of `InfoText.json`.

PhysicalButtonPressed.cs - Calls upon a specified method when the connected button is pressed.

PokingGestureAction.cs - Check if the user is making a poking gesture with their hands. This gesture allows the user to interact with buttons, and use the `PhysicalButtonPressed.cs` script.

A.5.2 The Hub

SceneLogic.cs - Contains methods that allows the user to load a specified scene, a random scene and to reset the current score.

ScoreLogic.cs - Consists of methods that can translate the leaderboard array to a string, and the leaderboard string back to an array.

ShowScoreboard.cs - Displays the current score and is responsible for showing the leaderboard to the user as a 2D element inside a canvas.

StaticData - Contains an array that consists of each score the user has earned from the minigames.

A.5.3 Save Score

ButtonListener.cs - Attached to every letter in the scene. Responsible for and adds the corresponding letter to the name of the user.

CreateAlphabet.cs - Creates the whole Norwegian alphabet as 2D buttons inside a canvas.

SaveScore.cs - Updates the leaderboard array with the newly created name and score tuple, placing it at the correct index.

A.5.4 Tool-sorter

CheckValidTools.cs - Increases or decreases the score based on the tools placed or removed from the rightmost table. Rewards the user with a score when all the tools are placed correctly.

DetectTool.cs - Notifies `CheckValidTools.cs` when a tool is placed or removed from the rightmost table.

A.5.5 Pipe-builder

ButtonTrigger.cs - Instantiates the pipe prefab specified by the button the script is attached to.

CallSnapZones.cs - Deactivates the interaction logic to the Snap Zone the script is attached to and calls on a method in `SnapZones.cs` based on the snapped pipe prefab.

DestroyTrash.cs - Destroys the GameObject triggered the method call.

Reset.cs - Calls a method in `SnapZones.cs` that removes all placed pipes and randomizes the obstacle placement.

SnapZones.cs - Keeps track of where new Snap Zones can be created through a two dimensional pipe array. `SnapZones.cs` also awards the user with a score when a continuous pipe path between the start pipe and the end pipe can be established.

WallGrid.cs - Creates a grid in front of the pipe location to show where the pipe array is.

A.5.6 Pipe-fixer

BreakWallPart.cs - Destroys the `WallSection` prefab the script is attached to and instantiates nine smaller wall pieces that has a lifespan of two seconds.

CreateBrokenPipes.cs - Iterates through all the pipes contained inside a GameObject and has a 10% chance of altering the components of a pipe, indicating that it is broken. The puddle prefab is added to the broken pipe.

DelayedSound.cs - A component of the broken pipe GameObject. `DelayedSound.cs` plays a dripping sound in a random interval between one and five seconds.

FixPipe.cs - Attached to the pipe wrench. When the collider at the same GameObject collides with a broken pipe, `FixPipe.cs` removes the puddle prefab and revert the changes made by `CreateBrokenPipes.cs`.

GenerateWall.cs - Creates a wall made by multiple WallSegment prefabs, all having a different grayscale color to tell the segment apart from each other.

ObjectSpeed.cs - A component of the rubber mallet GameObject, tracking its speed.

ProgressOverview.cs - Updates the information text when a broken pipe is fixed to show how many broken pipes remain. Rewards the user with a score when all broken pipes are fixed.

PuddleLogic.cs - A component of the puddle prefab. `PuddleLogic.cs` creates two sine waves with random frequencies that the puddle parts use to calculate their position.

A.5.7 Ventilation-setup

ChangeRotation.cs - Rotates the valve at the end of the ventilation system.

RiseVent.cs - This script rises the ventilation pipe and the hinge to its correct position.

ScoreView.cs - Keeps track of the score of the player and updates the informational text located on a canvas accordingly.

TurnOnVent.cs - Completes the minigame when every part of the ventilation system is in the correct position.

A.5.8 Ventilation-fixer

BoltAction.cs - Contains methods for making a screw and unscrewing it. The unscrewing method calls upon another method responsible for updating the score and the informational canvas.

ScoreCounter.cs - This script keeps track of the score and updates the user instructions as the player progresses.

Screw.cs - Checks if the drill-GameObject collides with the screw.

SluttVent.cs - Makes a new ventilation part at the end of the ventilation system.

TurnOnPower.cs - Contains methods for turning on the electricity in the minigame by pressing three buttons in one particular order.

Ventil.cs - Controls the state of the valve created by `SluttVent.cs`.

A.5.9 Air Conditioning

CheckIfClosed.cs - Validates if a window or door is closed or not.

DoorGrabber.cs - Moves the attached door handle based on the position of the interactable GameObject the user can move.

FollowPhysics.cs - Limits the effect of `DoorGrabber.cs`, resulting in a smooth movement of the door.

InConStats.cs - Calculates a score based on the placement of the indoor component of the air conditioner.

OutConStats.cs - Calculates a score based on the placement of the outdoor component of the air conditioner.

Progress.cs - Separates the minigame into multiple steps and instantiates GameObjects when needed. Rewards the user with a score consisting of the score from each step summarized together upon completion of all the steps.

UsageStatus.cs - Checks if the connected Snap Zone is in use or not.

A.5.10 Quiz

QuizManager.cs - Decides if the user should answer a question or request the next question. Also awards the player with five point for each correct answer and presents the questions to the player.

Appendix B

Acronyms and glossary

B.1 Acronyms

Acronyms

6DoF - Six Degrees of Freedom.

API - Application Program Interface.

APK - Android Application Package.

FPS - Frames Per second.

HMD - Head Mounted Display.

IDE - Integrated Development Environment.

JSON - JavaScript Object Notation.

OLED - Organic Light-Emitting Diodes.

SDK - Software Development Kit.

UI - User Interface.

UiS - The University of Stavanger.

VR - Virtual Reality.

VRTK - Virtual Reality Toolkit.

B.2 Glossary

Asset - A representation of any item that can be used in a Unity project. Assets can be created both inside Unity and outside. Examples of assets includes 3D models, images and audio mixers.

Asset Store - A library integrated into Unity for downloading of free and commercial assets. The Unity Asset Store offers assets such as textures, materials, models and animations.

Box Collider - A rectangular cuboid. A box collider can be used to prevent GameObjects from clipping into each other, as well as allowing for methods to be called when two GameObjects collide.

Components - The functional part of a GameObject. Unity has many built-in components, and it is possible to create custom components by writing scripts.

GameObject - The fundamental objects in Unity that represent props, logic and scenery. A GameObject does not do much itself, but acts as a container for components, which serves real functionality.

Prefab - An asset type that allows GameObjects complete with components and properties to be stored. A prefab can be imported into a scene and edited, without altering the prefab itself.

PlayerPrefs - Sores and accesses player preferences between game sessions. PlayerPrefs can only store primitive datatypes such as strings and integers, and can be accessed by a self-defined key-string.

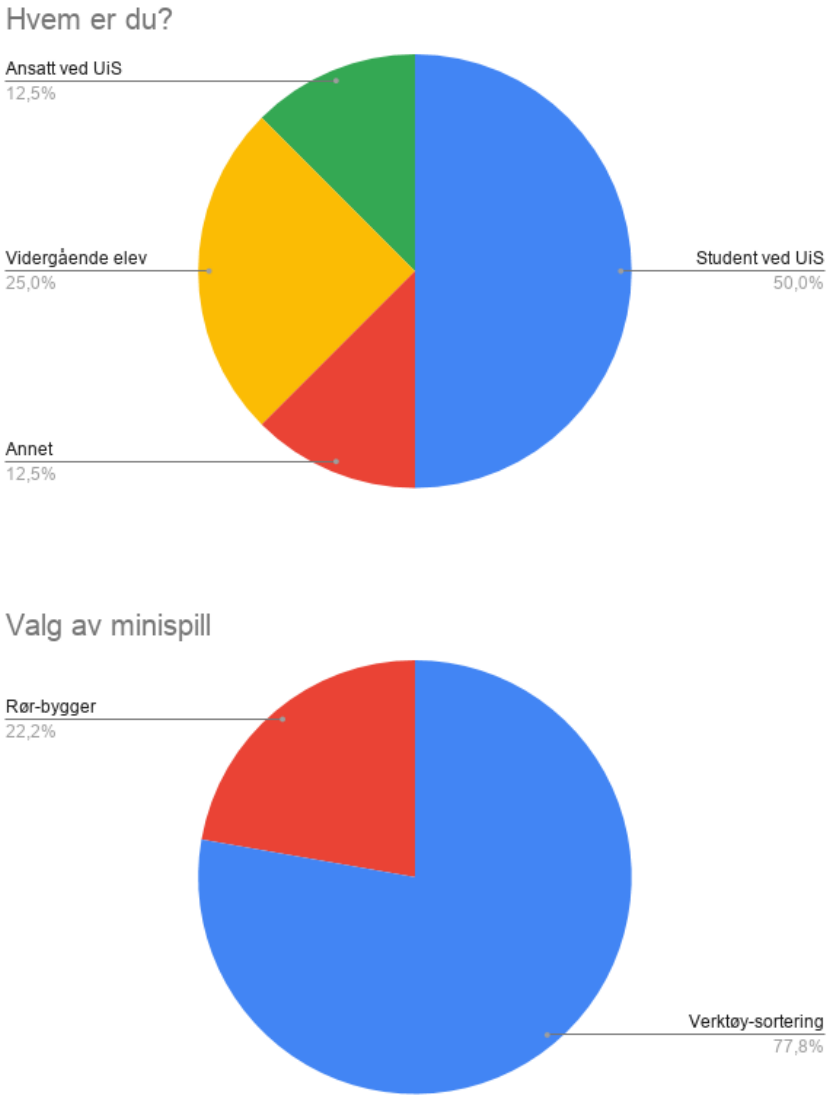
Rigidbody - A component that allows a GameObject to be affected by gravity and other forces.

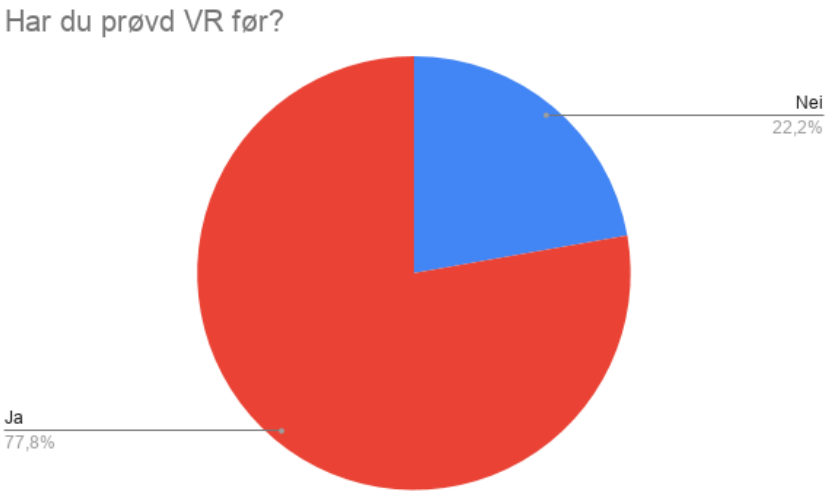
Scene - Contains the environment and menus of a game. Scenes hold all the logic, obstacles and decorations needed for a user experience to take place.

Snap Zone - A VRTK prefab that provides a predefined zone where a 3D model can be dropped and upon dropping snaps to the set snap drop zone.

VRTK - The Virtual Reality Toolkit is a collection of reusable solutions to common problems found when building for virtual reality. This includes solutions to teleportation, player-interactions with GameObjects and Snap Zones.

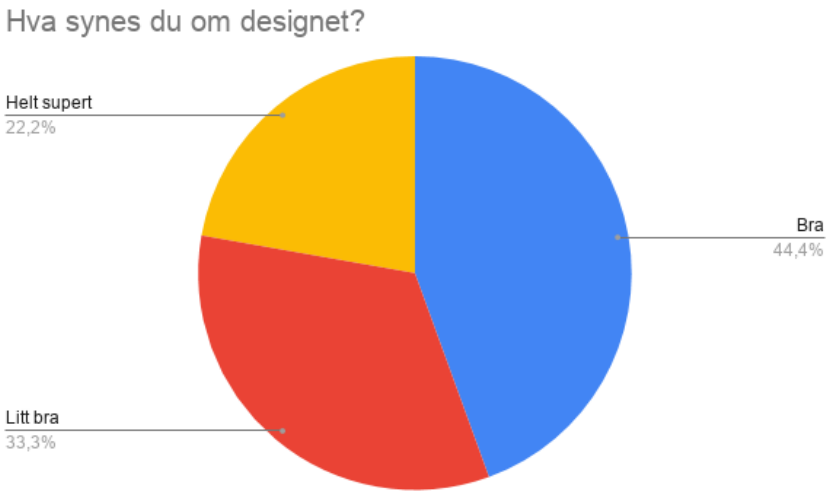
B.3 Open day feedback form

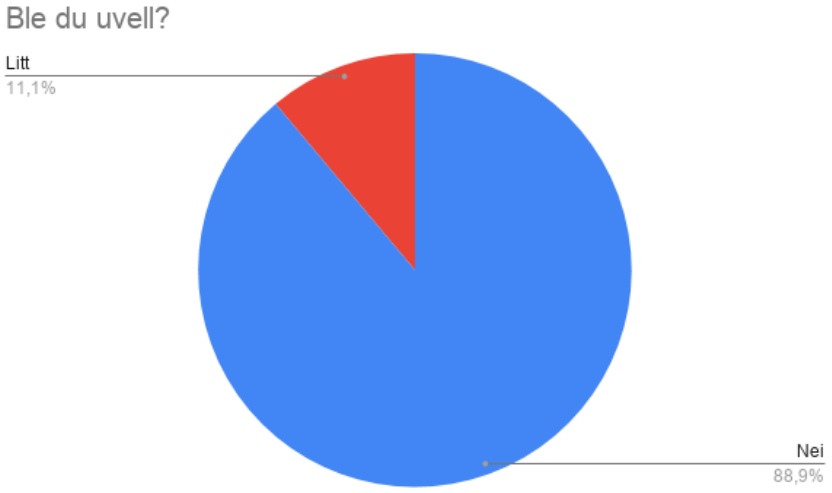


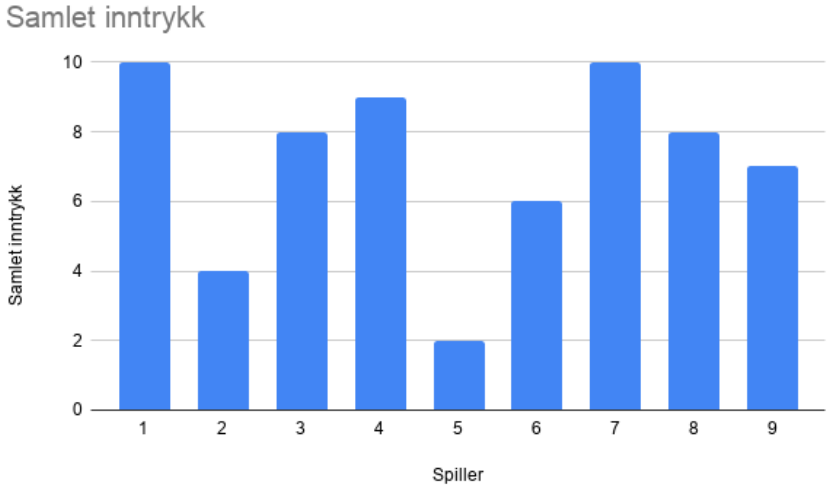

Har du prøvd VR før?
Nei
22,2%
Ja
77,8%


Hvor underholdende var spillet?
Underholdningsgrad
0
2
4
6
8
10
1
2
3
4
5
6
7
8
9
Spiller


Hvordan var vanskelighetsgraden?
Vanskelighetsgrad
0
2
4
6
8
10
1
2
3
4
5
6
7
8
9
Spiller


Hva synes du om designet?
Helt supert
22,2%
Bra
44,4%
Litt bra
33,3%


Ble du uvell?
Litt
11,1%
Nei
88,9%


Samlet inntrykk
Samlet inntrykk
10
8
6
4
2
0
1
2
3
4
5
6
7
8
9
Spiller

Bibliography

[1] Digital Trends. 8 virtual reality milestones, November 2017. https://www.digitaltrends.com/cool-tech/history-of-virtual-reality/. Last accessed 26/04/20.

[2] VisarTech. Why you should give up on pc vr and focus on standalone vr apps, November 2019. https://visartech.com/blog/advantages-of-standalone-vr-over-pc-vr/. Last accessed 26/04/20.

[3] Facebook AI. Powered by ai: Oculus insight, 2019. https://ai.facebook.com/blog/powered-by-ai-oculus-insight/. Last accessed 26/04/20.

[4] Oculus. Oculus guardian system, April 2020. https://developer.oculus.com/documentation/native/pc/dg-guardian-system/. Last accessed 26/04/20.

[5] Unity Technologies. Unity user manual (2018.4), January 2020. https://docs.unity3d.com/2018.4/Documentation/Manual/. Last accessed 05/04/20.

[6] Unity Developers. Unity integrations, February 2020. https://developer.oculus.com/downloads/package/unity-integration/. Last accessed 02/03/20.

[7] VRTK. Virtual reality toolkit, January 2018. https://www.vrtk.io/. Last accessed 06/05/2020.

[8] Joel Lee. Is sidequest safe for oculus quest?, August 2019. https://whatnerd.com/sidequest-safe-oculus-quest-sideloading-apps-guide/. Las accessed 06/05/2020.

[9] Medical News Today. What's to know about motions sickness, 2017. https://www.medicalnewstoday.com/articles/176198. Last accessed 26/04/20.