universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Developing a Framework for the Creation of Virtual Reality Experiment and Therapy Applications“

verfasst von / submitted by

Christoph Pressler, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2021 / Vienna, 2021

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 935
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Medieninformatik
Betreut von / Supervisor:	Univ. Prof. Dr. Helmut Hlavacs

Acknowledgements

This work would not have been possible without the help of my thesis advisor **Univ.-Prof. Dr. Helmut Hlavacs** - his ideas, knowledge and feedback guided me throughout this project.

I am also deeply grateful to my partner **Karoline Brabenetz**, who not only helped me directly by providing superb assets and valuable feedback, but also supported me emotionally during this time.

Additionally, I would like to thank **Veronika Pressler**, who proofread this work with me, **Mag. Dr. Oswald Kothgassner**, who provided great insight into anxiety disorders, **Daniel Martinek** and **Patrick Pazour**, who helped me solve technical problems and guided me through administrative tasks as well as my family and friends for their support.

Curriculum Vitae

Personal Information

Name	Christoph Pressler
Date of Birth	1996-01-07
Nationality	Austria
Contact	christoph-pressler@outlook.com

Education

10/17 - now	**University of Vienna** Master studies Media Informatics
10/14 - 07/17	**University of Vienna** Bachelor studies Informatics
09/06 - 07/14	**Don Bosco Gymnasium Unterwaltersdorf** Matriculation (*Matura*)

Publications

01/18	Pressler C., Hlavacs H. (2018) Timebender: A Multiplayer Game Featuring Bullet Time Mechanics. In: Cheok A., Inami M., Romão T. (eds) Advances in Computer Entertainment Technology. ACE 2017. Lecture Notes in Computer Science, vol 10714. Springer, Cham

Work Experience

07/18 - now	**Robimo GmbH** Software developer
11/19 - 09/20	**University of Vienna** Independent contractor for research project
09/18 - 09/20	**University of Vienna** Student staff research group Entertainment Computing
12/17 - 02/18	**University of Vienna** Independent contractor for research project
07/16 - 02/17	**Wein & Co Handelsges.m.b.H.** Software developer

Contents

List of Figures

List of Tables

List of Code Snippets

Abstract

The recent increase of interest by the general public in virtual reality headsets has led to many technological advancements in this area. Two of the areas potentially profiting from these advancements are psychology research and phobia therapy, since virtual reality allows users to enter scenarios which would be difficult or impossible to create in the real world and which are more realistic and immersive than conventional computer applications.

One of the main challenges in using virtual reality in these scenarios is the application creation process. Users without deep technical knowledge are unable to create virtual reality environments. Employing developers to create them can be expensive and yield unsatisfactory results.

This thesis proposes a new framework that allows for easier virtual reality therapy and psychology research application creation and that is usable by people without programming knowledge. This framework, called "Minogrid", simplifies the creation process into answering a number of single choice questions and installing the required software.

It is shown that this solution can be operated by users without programming knowledge by a usability evaluation. During this evaluation eight participants each created their own virtual reality applications. This task was split into two scenarios and involved the participants filling in an after-scenario questionnaire after each and a post-study usability questionnaire at the end. Scores indicate that the participants were satisfied with the presented system as well as that it is useful and has a high information and interface quality.

Zusammenfassung

Das verstärkte Interesse der Öffentlichkeit zu dem Thema „Virtuelle Realität“ (VR) führt zu zahlreichen technologischen Verbesserungen in diesem Bereich. Davon profitieren unter anderem psychologische Forschungen und Phobiebehandlungen, da Anwendungen, welche die Vorteile der virtuellen Realität nutzen, es ermöglichen Welten zu besuchen, welche in der realen Welt nur schwer oder gar unmöglich nachzubauen sind, und gleichzeitig den Realismus und die Immersion im Vergleich zu herkömmlichen Computeranwendungen deutlich erhöhen.

Eine der größten Herausforderungen der Nutzung von virtueller Realität in diesen Szenarien ist der Erstellung entsprechender Anwendungen. Nutzern ohne tiefgehendes technisches Wissen ist es nicht möglich solche Welten zu kreieren, wobei gleichzeitig Entwickler mit der Erstellung solcher zu beauftragen ein kostspieliges Unterfangen ist, welches zu unerwünschten Ergebnissen führen kann.

Diese Arbeit stellt ein neues Framework für die einfachere Erstellung von VR Therapie- und Forschungsanwendungen vor, welches auch von Nutzern ohne jegliche Programmierkenntnisse benutzbar ist. Dieses Framework, genannt „Minogrid“, erlaubt es Nutzern lediglich durch die Beantwortung diverser Single Choice Fragen und der Installation benötigter Software die Erstellung von VR Anwendungen.

Mit einer Usability Evaluation wird gezeigt, dass das Framework auch für Personen ohne Programmierkenntnisse benutzbar ist. Im Zuge der Evaluation haben acht verschiedene Benutzer je eine VR Anwendung erstellt. Diese Aufgabe wurde in zwei Unteraufgaben unterteilt, nach welchen je ein „after-scenario questionnaire“ zu beantworten war. Am Ende der Evaluation wurde ein „post-study usability questionnaire“ ausgefüllt. Die Ergebnisse zeigen, dass das vorgestellte Framework eine hohe „information quality“, sowie „interface quality“ aufweist und als nützlich empfunden wird.

1 Introduction

1.1 Motivation

1.1.1 Virtual Reality

Virtual reality (VR) allows users to enter a virtual world rendered by a computer. Head movements as well as hand movements are tracked and translated into this virtual space. In contrast to traditional computer applications this greatly increases realism and the sense of spatial presence and can improve the feeling of immersion [23]. Thanks to a growing consumer interest in this technology [49][11][24], a lot of research and high investments are currently concentrated on related technologies [10]. This has lead to significant advances of virtual reality in the last decade, while also increasing the affordability of such systems.

1.1.2 Psychology Research

Since the birth of virtual reality it has been a technology often used in psychology research, with early publications dating back to the 1980s [10]. Virtual reality allows researchers to create experimental setups not feasible in the real world. It is now possible to create an environment exactly tailored to the needs of the researcher, or to generate an environment at runtime to create a new, unique experience every time. Working in a virtual environment also allows to easily monitor and record the user's behaviour. For example, sensors mounted on the body of the user could record different vital parameters and a second screen could allow the researcher to see exactly what the subject is seeing.

Another use-case for virtual reality also related to psychology research is phobia therapy [19][16]. Many of the advantages of virtual reality applications for psychology research also apply to phobia treatments [8][26]. Therapists can transfer the user into virtual environments where they can slowly and carefully be exposed to scenarios based on their phobia. For example, a person with glossophobia[1] could try to speak in front of a virtual crowd with the help of their therapist. The therapist can always stop the session and bring the user back to a safe space if necessary.

1.1.3 Reducing Costs

To summarize, conducting experiments and therapy sessions with the help of computers leads to a plethora of positive outcomes [42]. More parameters can be controlled than in the real world, and it is possible to create otherworldly environments or to add random elements to an existing scenario by generating parts of, or even the whole, environment. Of course, using computer-based scenarios instead of real world ones greatly reduces realism. This generally does not have to be a bad thing, since there are situations where reduced realism

[1] The fear of public speaking.

is desired. However, more often than not, the goal is to increase presence in the virtual world. VR increases presence while still allowing the user to quickly leave the simulated scenario. Thanks to current research, investments and the increased interest of the general public, this technology is developing quickly while its costs are decreasing, making it a good choice for psychology experiments as well as therapy sessions.

Yet, to create such applications, usually advanced technical knowledge is needed. This presents potentially interested researchers and therapists with multiple challenges. External developers are needed to create the environment(s) and logic. Finding such developers is no easy feat, especially for smaller enterprises, and takes a considerable amount of time and money. Of course, paying these external developers also further increases expenses.

Conducting such a cross-disciplinary project in itself is a very challenging task with a wide variety of potential pitfalls and problems [28]. One of the most error-prone areas of such projects is the communication between the project members [15]. This increases development time and cost and can amount to a finished product that differs significantly from what was intended.

1.2 Problem Statement

To decrease development costs and potential communication problems a number of tool sets and frameworks exist. These constructs allow users with different technical knowledge levels to create their own applications, or at least adapt existing ones to their use case. The related works section conducts a survey of already existing solutions to this problem.

The goal of this thesis is to develop a state of the art framework for the creation of virtual reality experiments and therapy applications. This framework should allow users with little technical knowledge to create applications that can be used for phobia treatment sessions as well as psychology research, while users with more technical knowledge are provided access to more advanced options.

This implies the necessity of an easy to understand step-by-step process for the creation of the application as well as a more complex editor for experienced users, all in addition to data capturing and evaluation mechanics. To allow for state of the art rendering to increase presence [17] while at the same time not exceeding the scope of this work, it was decided to use an existing 3D rendering pipeline.

To summarize, the goals of this thesis are:

1. An evaluation of existing virtual reality frameworks
2. The development of a virtual reality framework
   (a) That is usable by users without technical background
   (b) That allows users with advanced technical knowledge to capitalize on their abilities
   (c) That generates applications which use state of the art technology
   (d) That is compatible with a wide variety of VR headsets
   (e) That has mechanisms for data capturing and evaluation
3. The implementation of the developed framework
4. The evaluation of this implemented virtual reality framework

2 Related Work

2.1 Overview

As mentioned in section 1, a number of frameworks and tool sets already exist to help developing virtual reality applications for psychology research and phobia therapies. In this section, four different approaches and one paper focused on investigating such systems are discussed. While the following publications are primarily aimed at psychology research, their fundamental goals align with ours: To create an application, a framework or a tool set that helps people with different technical knowledge levels create their own 3D applications.

These approaches are categorized according to the focus of the system, the technical knowledge needed by the user of the resulting software environment and whether it supports virtual reality in table 2.1:

System	Focus	Necessary knowledge level	VR support
Maze Suite [6]	Environment generation Data capturing Data analysis	Medium	No
VREX [47]	Environment generation Data capturing	Medium	Yes
EVE [14]	Data capturing Data analysis	High	Yes
SNaP Framework [4]	Data capturing	Low	Yes

Table 2.1: Categorization of software development environments for the creation of virtual reality experiments.

2.2 Literature Overview

2.2.1 Maze Suite 1.0: A complete set of tools to prepare, present, and analyze navigational and spatial cognitive neuroscience experiments

This paper [6] written by Ayaz et al. in 2008 aims to provide researchers without programming knowledge with a set of applications enabling them to create navigational and spatial cognitive neuroscientific experiments which take place in a virtual environment. Although the Maze Suite does not offer support for virtual reality, it is relevant in the context of psychology experiment creation frameworks and is one of the first efforts to support both 3D environment creation and in-depth data analysis.

The Maze Suite consists of the following applications:

- MazeMaker: The MazeMaker is used to built 3D mazes in a 2D drawing environment. Users can draw lines that represent walls in a bird's-eye view and add simple 2D shapes which represent 3D objects (as shown in figure 2.1). It is also possible to change certain properties of these drawn walls and objects, such as color and texture.
  The MazeMaker also incorporates a tool called MazeListBuilder which can be used to combine multiple mazes as well as text messages (interactive, dialog based) that are shown between these mazes. These combinations of mazes and text form an experimental protocol.

- MazeWalker: The MazeWalker, shown in figure 2.2, is a plain 3D engine which processes the experimental protocol created with the MazeMaker and renders the 3D environment specified in this protocol.

- MazeViewer: The MazeViewer is used to display the data gathered by the experiment. It can access log files created by the MazeWalker containing data about the behaviour of the subject in the maze, such as the time needed to complete the maze or the number of errors. Researchers can also display the time-correlated path of the subject or export the gathered data to process it with other software.

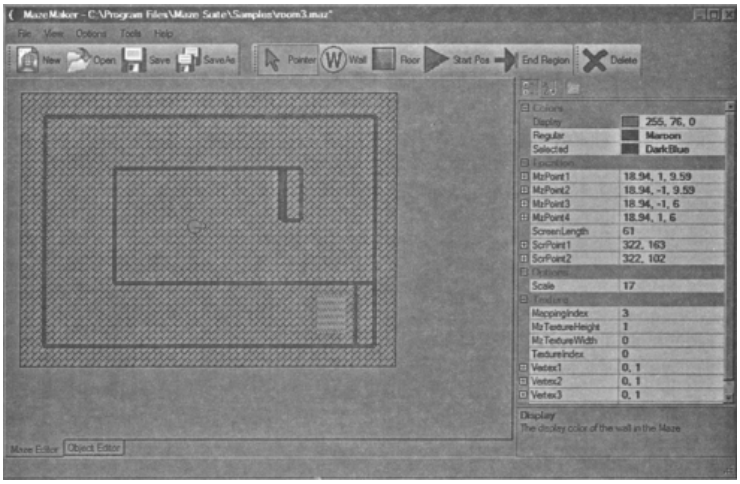

Figure 2.1: Figure from "Maze Suite 1.0: A complete set of tools to prepare, present, and analyze navigational and spatial cognitive neuroscience experiments" [6], showing the MazerMaker.

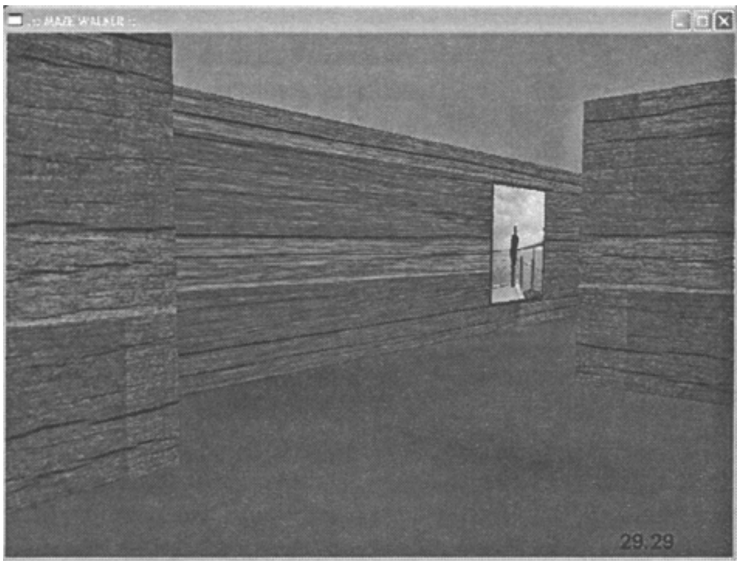

Figure 2.2: Figure from "Maze Suite 1.0: A complete set of tools to prepare, present, and analyze navigational and spatial cognitive neuroscience experiments" [6], showing the MazeWalker.

2.2.2 VREX: an open-source toolbox for creating 3D virtual reality experiments

This paper [47] by Vasser et al. from 2017 builds upon the basic idea of the Maze Suite of creating simple 3D environments without advanced computer science knowledge but uses more recent technologies. It transfers this idea into the virtual reality space and allows researchers to conduct different types of experiments.

With VREX, users can create virtual environments consisting of a number of connected rooms. An environment can either be manually created by combining rooms one-by-one or auto generated. A room may be populated with objects. As before, this can either happen automatically (as shown in figure 2.4) or manually with an editor (as shown in figure 2.3) specifically provided for this purpose. VREX offers a selection of predefined objects to enable users without any 3D modelling skills to create environments, while more advanced users are given the possibility to add their own models. The created environments can then be sequenced to create experiments.

Figure 2.3: Figure from "VREX: an open-source toolbox for creating 3D virtual reality experiments" [47] showing the 3D editor. A user can place objects (a), select objects (b) and change object properties (c).

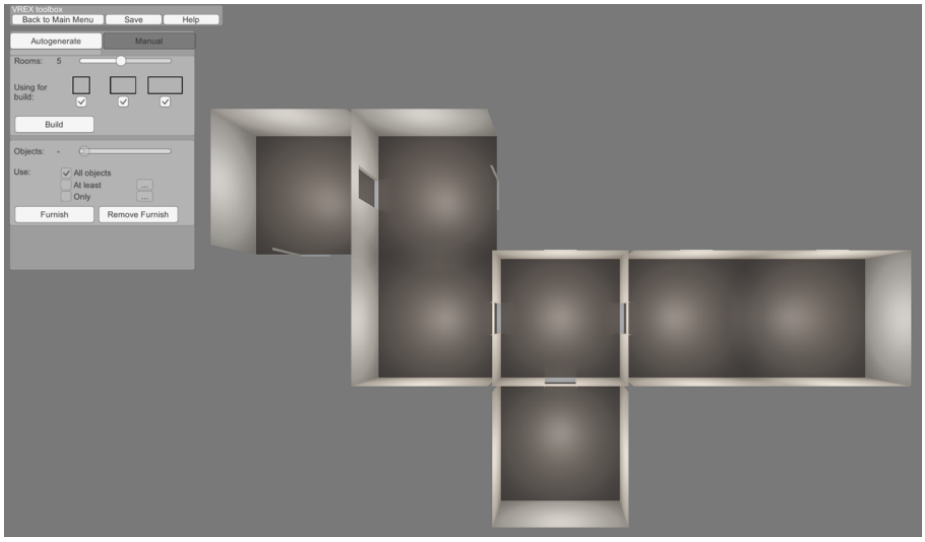

Figure 2.4: Figure from "VREX: an open-source toolbox for creating 3D virtual reality experiments" [47], showing an automatically generated maze.

VREX supports two different experimental paradigms:

- Change blindness: When a specified object falls outside the field of view of the subject, it can change visibility, color or location. It will alternate between two predefined states. The participant can click a button as soon as he identifies the change and the response time is logged.
- False memory: This experiment is divided into two phases. During the first phase all objects seen by the participant are logged. In the second phase, the subject is presented with multiple different objects, including the logged objects of the first phase. Participants must recall all previously seen objects. The answers are logged.

The logged data is stored in a file which can then be post-processed by the conductor of the experiment.

Instead of creating a new 3D engine, the authors decided to build VREX as a toolbox for the Unity [44] engine by Unity Technology. It adds a simple, easy-to-use UI for researchers without modelling or programming knowledge but still allows more advanced users to use all capabilities of a modern, full-fledged game engine, like 3D spatial audio, physics and animation.

2.2.3 EVE: A Framework for Experiments in Virtual Environments

The authors Grübel et al. describe a framework for the setup, implementation and evaluation of experiments in virtual reality in this paper [14] published in 2017. The EVE (Experiments in Virtual Environments) framework is, like VREX, based on the Unity engine but does not hide away its complexity behind a new user interface. Instead, it relies on users learning how to work with Unity to create and modify virtual environments.

To help researchers create virtual reality experiments, EVE offers a large set of experiment-oriented assets which can act as sensors that are placed in the virtual world to collect data as well as canned UI elements and questionnaires which may also pop-up during runtime to gather information during an ongoing experiment. The complete pipeline is shown in figure 2.5.
The data collected during an experiment is stored in a MySQL [38] database. An evaluation menu (shown in figure 2.6) provides easy access to this data as well as data management and visualization options. Researchers can replay a participant's walked path from either a first-person or a bird's-eye perspective and export the data in the CSV format. Additionally, the authors offer a package for the statistical programming language R [39] for more sophisticated interactions with the data.

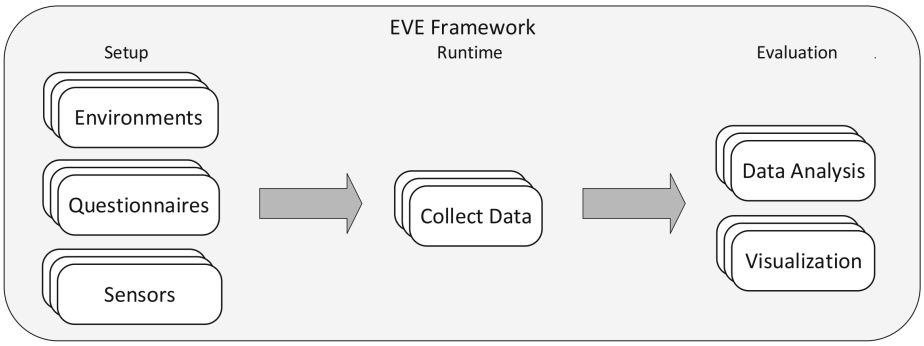

Figure 2.5: Figure from "EVE: A Framework for Experiments in Virtual Environments" [14], showing an overview of the three stages of an experiment supported by the EVE framework. The authors describe the three stages as follows: "In the setup stage, experimenters can upload and manipulate their VE, load XML-based questionnaires, and place their virtual sensors. EVE generates executable files that can be used during runtime to start the experiment and data collection. In the evaluation stage, experimenters can perform different types of analyses and visualizations and export the collected data in different formats." [14].

Figure 2.6: Figure from "EVE: A Framework for Experiments in Virtual Environments" [14] showing the participant's path and options to export the collected data.

2.2.4 VR for Everybody: The SNaP Framework

The SNaP (Spatial Navigation Paradigm) Framework from Michelle Annett and Walter F. Bischof, published in 2009, allows users with different technical skills to create spatial navigation research experiments.

In the SNaP Framework, experiments are defined through an XML-based parameter file and one or multiple Virtools [1] Composition files. The parameter file is user generated and specifies one or more experiments, which are defined as a number of trials, the trial conditions and the desired peripherals. The Virtools Composition files are canned, one for every experimental paradigm, and define the virtual world, consisting of a number of custom scripts that define user interaction, behavioural recording and environment models. The 3D environment is then rendered using the Virtools VR Player. The data captured during an experiment is stored in text and bitmap files which can then be processed with other software.

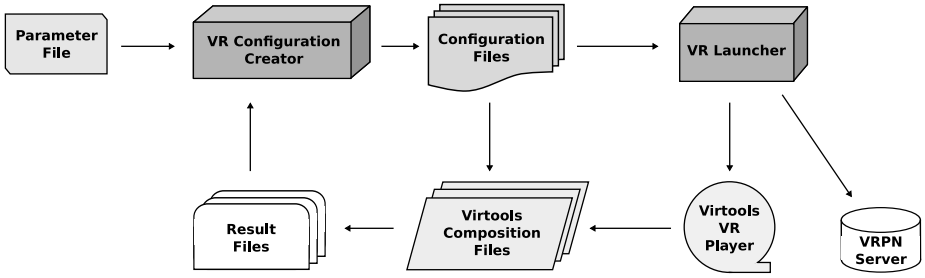

Figure 2.7: Figure from "VR for Everybody: The SNaP Framework" [4], showing the SNaP Framework pipeline.

To evaluate usability, the following user study was conducted: Eight participants, split into three groups (with three participants completely unfamiliar with VR technologies, three participants with advanced computer knowledge and two participants with advanced computer and virtual reality knowledge), were asked to specify and deploy two experiments with the SNaP Framework. It took them on average ten minutes to specify and deploy an experiment, and many participants indicated that they easily and quickly understood the structure and formatting of XML.

While extending the existing Virtools software allowed for quicker development of the SNaP framework, the authors also state that it created a number of problems that required them to develop a variety of "hacks" to ensure functionality.

2.2.5 Investigating the Application of Virtual Reality Systems to Psychology and Cognitive Neuroscience Research

This paper [5] by Annett and Bischof published in 2010 investigates different groups of VR users and their needs as well as usability and user requirements of current virtual reality systems in the context of psychology and cognitive neuroscience research.

The authors identify five main components that would be the main beneficiaries of usability assessments and subsequent improvements, as shown in figure 2.8:

- Environment Creation and Deployment: How does a user create and orchestrate 3D VR environments and the models used in these environments?
- Environment Constraints and Behavior: How is logic added to the environment? Are predefined experiment protocols available?
- User Interaction: How does a subject move within the environment? How can they interact with the environment?
- Interfaces and Devices: How are devices integrated? Are behavioral measurement devices, e.g. an EEG or eye trackers, supported?
- Measurements: What data is captured? How can a user access this data?

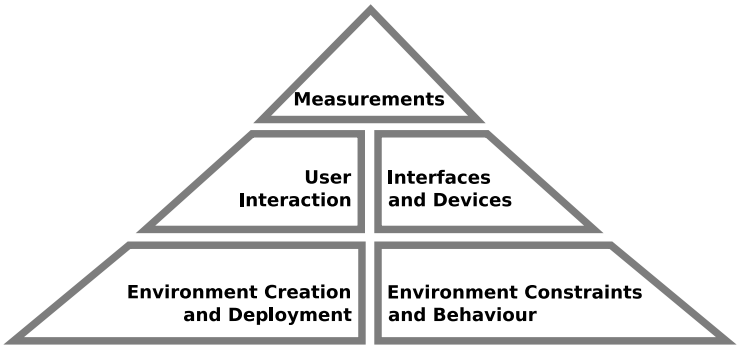

Figure 2.8: Figure from "Investigating the Application of Virtual Reality Systems to Psychology and Cognitive Neuroscience Research" [5], showing five components of VR systems that would benefit from a usability assessment and subsequent improvement.

The authors state that "it is is not possible to create a system that allows novice users to have full control over all aspects of an environment or experiment. In the short term, it may be reasonable to restrict the capabilities of users, but as users begin implementing complex paradigms or interaction metaphors; this is not a feasible solution." [5].

Another key insight of the publication is that the best evaluation of such a system can be done by creating a set of general, abstract tasks that are required for all virtual reality experiments. Nonetheless, it remains to be determined which tasks fulfill this condition and how to assess and compare task performance.

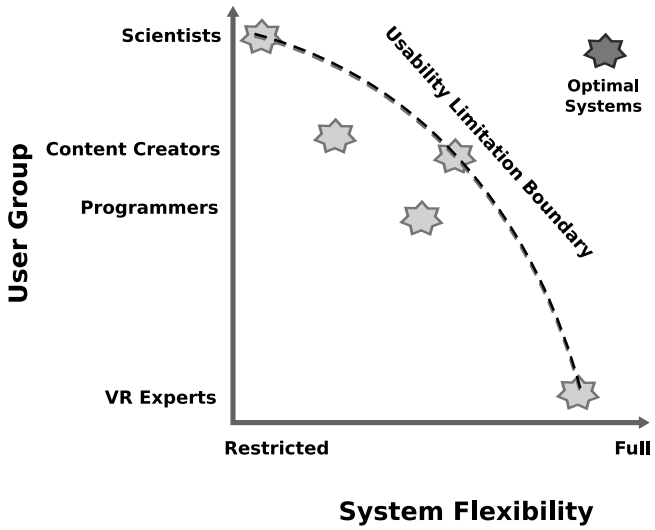

Figure 2.9: Figure from "Investigating the Application of Virtual Reality Systems to Psychology and Cognitive Neuroscience Research" [5], showing "a comparison between the user groups that each VR system is aimed toward and the flexibility that is inherent in each system, from very restricted to fully flexible" [5]

2.3 Analysis and Comparison

The main differentiator between these systems is the balance between complexity and restriction. While EVE [14] allows researchers to use all capabilities of the Unity engine and requires them to learn how to operate such a complex environment, the SNaP Framework [4] hides away nearly all complexity while also severely restricting the options a user has when creating an experiment. VREX [47] and the Maze Suite [6] mark the middle ground between these two approaches. VREX also uses Unity as basis but adds a new simplified UI that lowers the difficulty of using it. The Maze Suite uses its own tool chain that allows researchers to create and conduct experiments using 3D mazes. This strategy restricts possibilities, but also reduces complexity.

While each of the investigated systems allows scientists to capture a subject's data during an experiment, [6] and [14] also contain advanced data analysis functionalities. Although it is possible to create and alter 3D environments using [14] or [4], it is not the focus of these systems, and both require users to learn how to operate other software to accomplish this, while [6] and [47] offer easy-to-use interfaces to alter the environment.

As noted by [5], there exists no standardized way of evaluating such a system. Only the authors of the SNaP Framework evaluated their solutions by conducting a user study. It is important to define a set of tasks to evaluate future frameworks in order to provide a better way of comparing these solutions with each other, as the number of comparable systems will increase as technology gets both cheaper and easier to use and virtual environments become more realistic and immersive.

3 Concept and Methodology

This section takes a look at the approach developed for this thesis. An overview over the idea behind this framework and the main concepts is provided. The components in play and how they communicate with each other is outlined as well as the steps a user takes in order to create a VR application.

3.1 Overview

As shown in section 2, a number of different approaches for the development of a software suite exist that help create, distribute and evaluate virtual reality experiments. Each software has its own benefits and is targeted at a specific user group, as shown in figure 2.9.

The goal of this thesis is to provide researches as well as therapists with a software package that they can use without days, weeks or even months of prior tool training. At the same time more advanced users should also be able to make use of their skills and knowledge.

Today, more high quality and easy-to-use game engines exist than ever before. This thesis aims to leverage the power of an existing game engine while adding additional software to lower the entry bar even further.

3.2 Three Component Approach

3.2.1 Idea

For this thesis, a new approach, combining the approaches discussed in section 2, was taken. The presented approach is based on three different components: An application that allows easy step by step project creation, an already existing 3D engine and a plug-in that enables the created application to communicate with the existing engine.

This approach combines the strengths of an advanced, state of the art 3D engine with the low complexity of a simple step by step project creation wizard. While users with a lower technical skill level can create their own versions of existing scenes, more advanced users can fine tune every detail or even implement their own logic while still taking advantage of the options the framework provides, such as data capturing (data capturing is discussed in more detail in section 6.3.4).

3.2.2 Potential Pitfalls

However, using three different components may lead to various problems. For one, users have to install three different tools to create an application. Secondly, all three tools have to work together, which can lead to compatibility problems if one application is running a version incompatible to the version the other applications expect.

To counter these problems, it was decided to use a web application for step one, project configuration. Users only have access to the most recent version of this application. Once the user downloads the fully configured project and the plug-in, they are automatically informed which version of the engine they will need to install to be able to open the project.

3.3 The Role of Each Component

3.3.1 Web Application

The starting point of each project is the web application. Here, users can choose an environment for their project (for example a city). These environments are tagged with example use cases (for example "fear of heights"), allowing users to quickly get an overview over the existing environment library and find the one best fitting their use case. After selecting an environment, users can then change various properties and parameters of this chosen environment. For example, the appearance of the player, the weather or where the player starts in the scene can all be changed.

The web application then generates a project containing the files needed for rendering the scene as well as a configuration file specifying the properties selected by the user and the engine plug-in.

After the user has downloaded all the necessary files, the web application will go on to explain how to install the 3D engine and how to import the downloaded files. This explanation is presented in a step-by-step fashion, the goal being to make the entire process as easy and intuitive for the user as possible.

3.3.2 Plug-in

Once the downloaded files and the plug-in have been imported into the engine, the plug-in will parse the configuration file and build the user specified scene with the provided assets. The plug-in offers a plain UI, allowing the user to reload the configuration. The plug-in also offers the means to build the application into an executable file and save it to a user specified location.

The plug-in also allows users to disable or enable application sound and data capturing. The sound switch was primarily added to enable therapists to communicate with their patients while they are in VR if so desired. The ability to disable data capturing is included for privacy reasons. Lastly, the plug-in also allows the user to view data of past sessions if data capturing was enabled during these sessions.

3.3.3 Scenes

The engine itself is responsible for a wide variety of technical features. It renders video and audio, allows support for a range of head mounted displays and manages scene logic. More advanced users can also use it to manipulate the scene and change the existing logic by connecting their own logic via specified interfaces while retaining access to the existing data capturing features.

The scenes themselves should expose the user to their respective fears. To best fit the scope of this thesis, scenes best used with room-scale VR[2] were chosen.

3.4 Summary

While a self developed engine can certainly be of advantage for specialized use cases, the main emphasis of this thesis lies in investigating possible usability advancements; therefore, the decision was taken to use an already existing engine. This also allows the framework to support many different VR configurations out of the box as well as providing users with state of the art technology.

Users navigate an easy to understand web application to create their project based on existing project templates. This web application provides the user with instructions on how to install a 3D engine and import their project into this engine. A plug-in for the engine reads the configurations the user set and creates the specified scene. This plug-in acts as an simple interface to a complicated and very feature rich engine. Using this plug-in, it is possible to create an executable application, which can then be used to start scene playback. Each playback can be monitored. The monitored data is stored in an capture file, which can be opened and viewed using the engine plug-in. Figure 3.1 illustrates this process.

3.5 Methodology

The project started with the idea to automate some parts of the VR experiment creation process. The first step was to find and analyze related work. Based on the findings of this literature survey, a new concept was created to try and take advantage of as many of the possibilities that new technical advancements provide as possible.

After creating the concept, research to find the most advantageous technology for the new proposed framework started and a matching software architecture was developed. Based on the architecture and selected technology, a development environment for each component was set up.

Unity [44] was chosen as 3D engine for this project. This free-to-use engine provides a state of the art feature set and comes with an extensive and detailed documentation.

[2] An environment where the user walks in VR by walking in the real world.

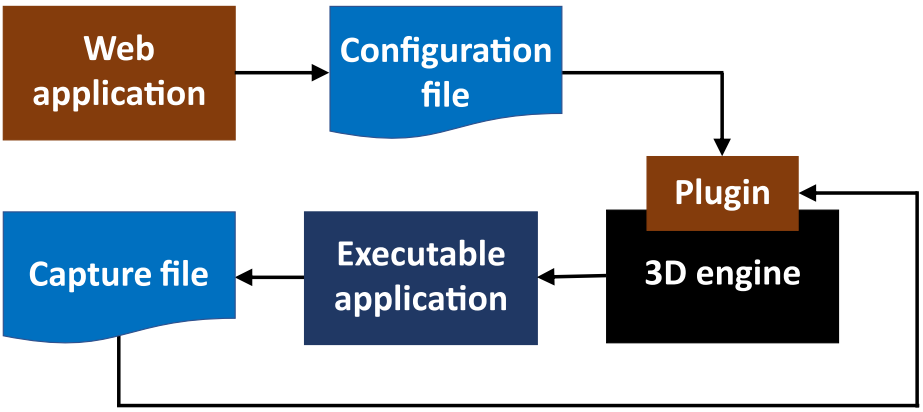

Figure 3.1: An illustration of the approach presented in this thesis. The user creates an configuration file using a web application. This file is read by an plug-in that was developed to enhance an existing 3D engine. The engine can generate an executable application which captures player data. This captured data is then read by the plug-in and can be played back inside the engine.

After creating fitting development environments, implementation of the web server began. The next step focused on translating the data generated on the server to the Unity engine. For this purpose, a new Unity plug-in was created. After that a Unity base scene was created, followed by two scenes, one dealing with fear of heights and one dealing with fear of thunder and lightning. This software suite was named "Minogrid". The web application was called the "Minogrid server", the plug-in "Minogrid plug-in" and the Unity base scene "Minogrid base".

Since all three main building blocks - the server, the plug-in and the Unity scenes - have dependencies towards each other, it was necessary to rework some parts of every component after adding functionality to another. One example of this are the images displayed when creating a scene on the server: The screenshots of in-engine material had to be done after the scenes in the engine are actually created, so the server data had to be altered based on the new Unity scenes.

To develop the VR scenes, the Oculus Quest with the Oculus Link Cable was used. All applications were developed using a PC with an Nvidia GeForce GTX 1080 and an AMD Ryzen 7 3700X.

3.5.1 Development Tools

The applications described in the following sections were developed using the programs outlined in table 3.1 below:

Tool	**Usage**
Visual Studio Code [31]	Code editor for web application
Visual Studio 2019 [30]	Code editor for Unity plug-in and Unity code
Unity	Creation and management of 3D scenes
Blender [7]	Creation of 3D objects
MongoDB Compass [33]	Debugging the database of the web application
Git [22]	Source code management
Microsoft OneNote [29]	Project management tasks and organizing work

Table 3.1: Tools used for the implementation of the VR framework presented in this thesis.

4 Design and Implementation of the Web Application

4.1 Design Considerations

As discussed in section 3, the web application acts as the starting point of the application creation process. The web platform allows users to register and later log in with the account they created, as shown in figure 4.1. This means users do not have to download and install anything when first starting to create a project. Once a user is registered, they can then create and manage projects. The status of each project and user is stored in a database.

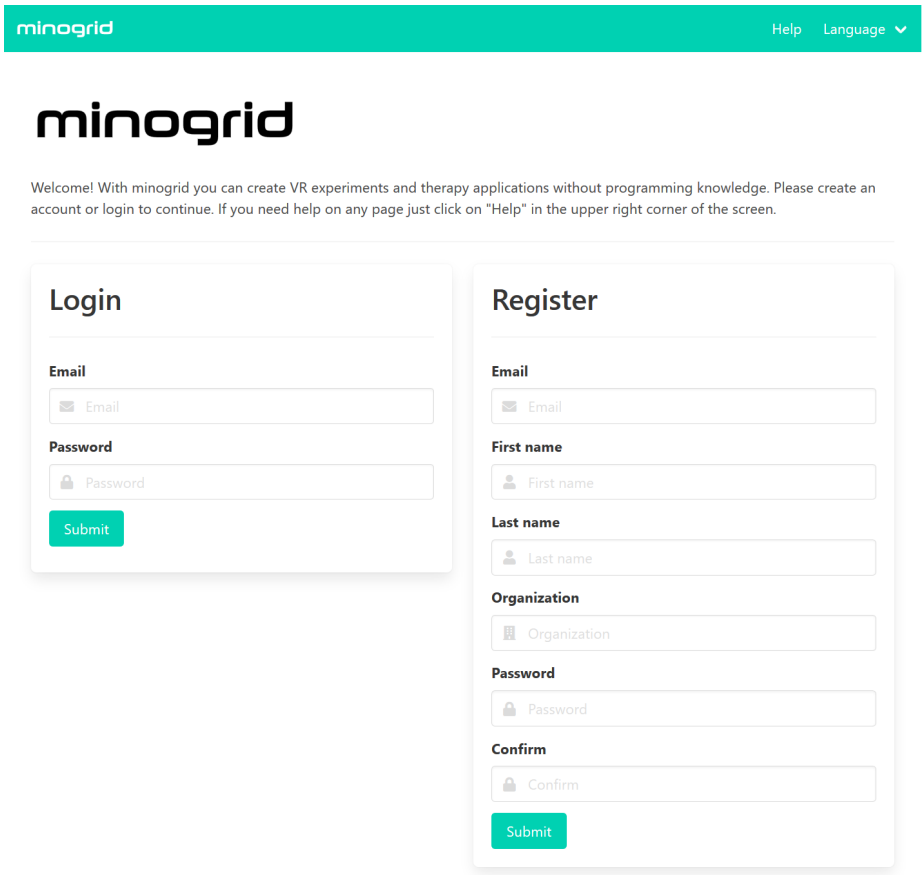

Figure 4.1: The landing page for a user that is not logged in to an account, showing the general aesthetics and color scheme of the web application. Users can switch languages in the upper right corner, login with an already existing account on the left side of the page and register on the right side of the page.

A project consists of one or more scenes, and each scene is created step by step using a scene creation wizard. The user selects an environment for the scene and is then presented with multiple choices to enable them to adjust the scene to their needs. The created scenes can be renamed and put into the desired order by the user on a project overview page. Once the user is satisfied they can download the completed project.

The downloaded package contains all files Unity needs to create a working project as well as the plug-in discussed in section 5. The web application also allows the user to switch languages between English and German.

Once they have registered and are logged in, users will be presented with an overview of their projects (cf. figure 4.2). Here they can delete existing projects or create new projects. Once they click on a project, they will be forwarded to the overview page of the selected project.

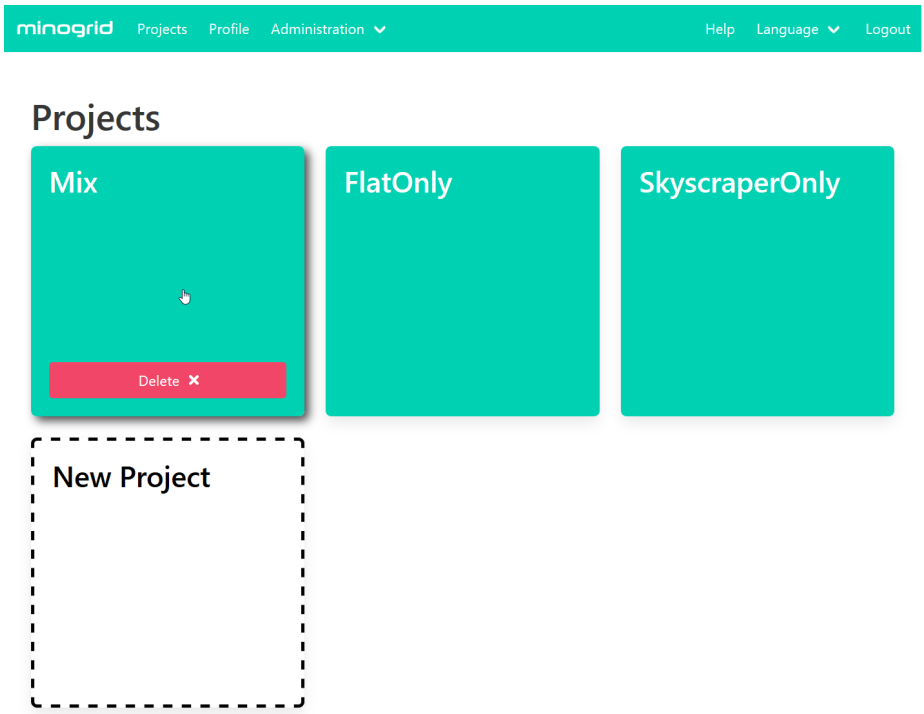

Figure 4.2: The projects overview page, showing three different projects. When hovering over a project, a "Delete" button that will delete the project when pressed appears. When clicking on a project, the user will be forwarded to that project's overview screen. When clicking on "New Project", users will be prompted to enter a name, erasing the "New Project" text and substituting the name chosen by them.

The project overview page, as shown in figure 4.3, shows the user a list of all scenes created for that project and allows them to reorder them as they see fit. There are also buttons to delete existing scenes or create new scenes. The project overview also allows users to download the project in order to import it into Unity. Additionally, a button "Next Steps" allows users to view instructions clarifying how to install all needed software and import the downloaded project into Unity.

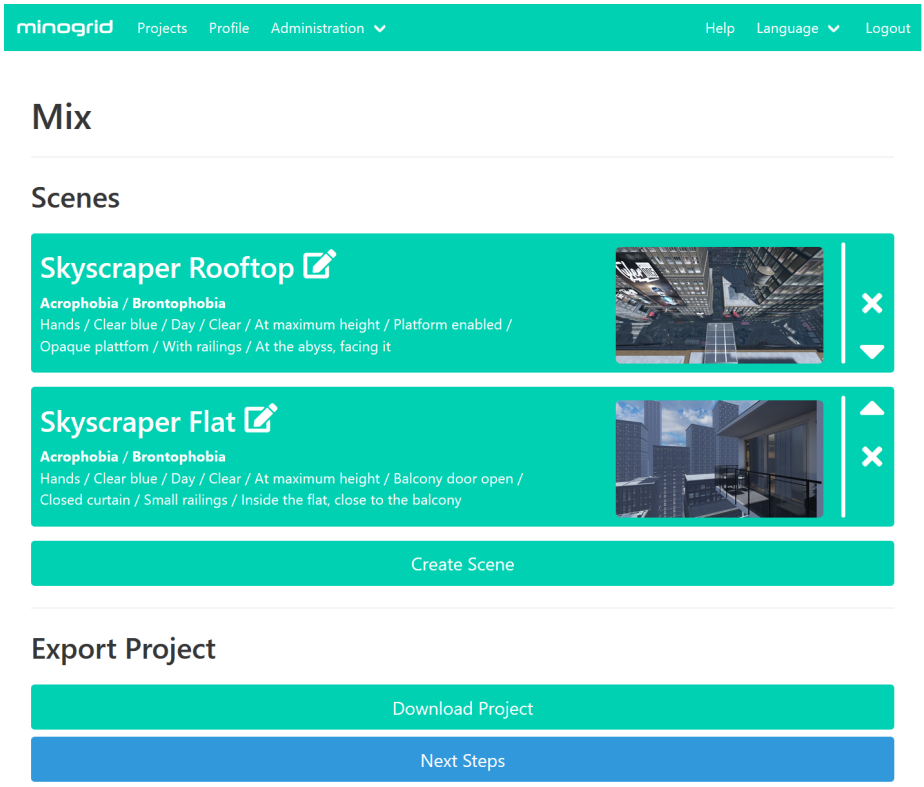

Figure 4.3: The project overview page shows the user the project name at the top, followed by a list of the scenes of this project. Users can reorder the scenes by pressing on the arrows on the right hand side of the scene's window or delete scenes by pressing on the "x". The order of scenes will be visible inside the Unity engine as well as in the finished application. Users can also rename the scenes and see tags associated with the scene environment; a short description of the scene is provided as well. To create a new scene, the user presses button "Create Scene" situated below the existing scene. The bottom section of the menu allows users to download the open project and to view instructions on how to get the project running.

In the right upper corner, a "help" button opens an overlay containing helpful information about the currently open page when clicked. Additionally, users can also navigate to general help and information pages about "Minogrid" with this overlay. Figure 4.4 shows the help overlay when the help menu is called from the project overview page.

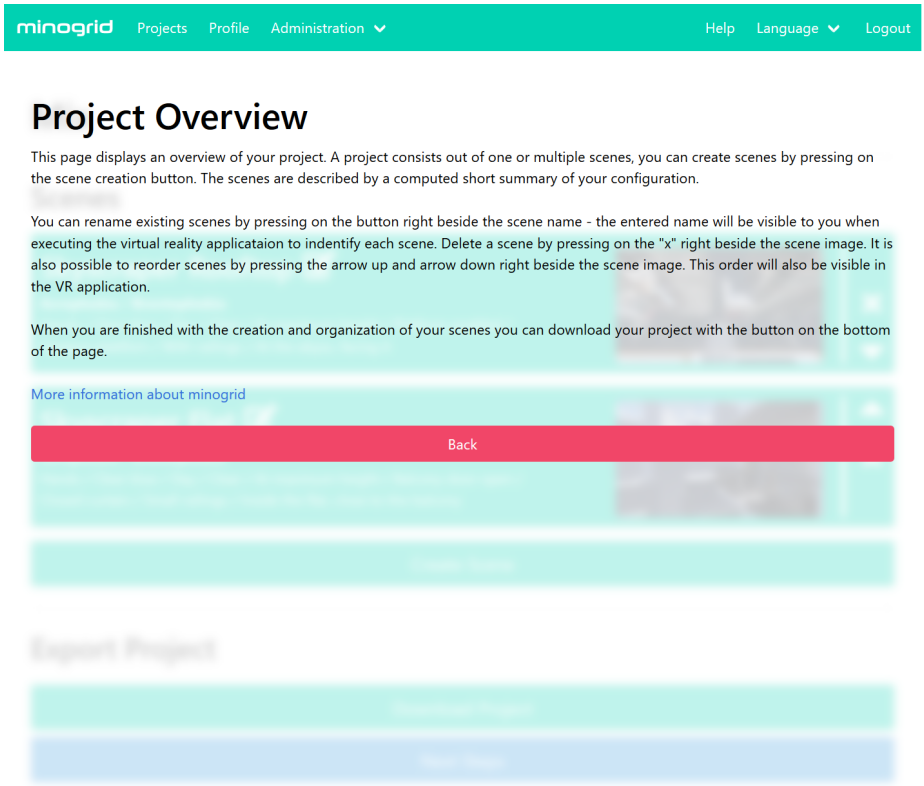

Figure 4.4: An overlay showing information on how to use the project overview screen.

To create a scene, users are guided through a number of questions. The questions and their corresponding answers are designed to be as simple and easily comprehensible as possible. Optional images and descriptions help to explain answer possibilities even further. Figure 4.5 shows two different example questions.

As with most web applications with a user account system, a profile page allows users to change their personal information and password. Furthermore, users with administrative rights have a dashboard that allows them to reload data that is only loaded when starting the server, for example translation data, which will be discussed in section 4.3.7, and to view all registered users.

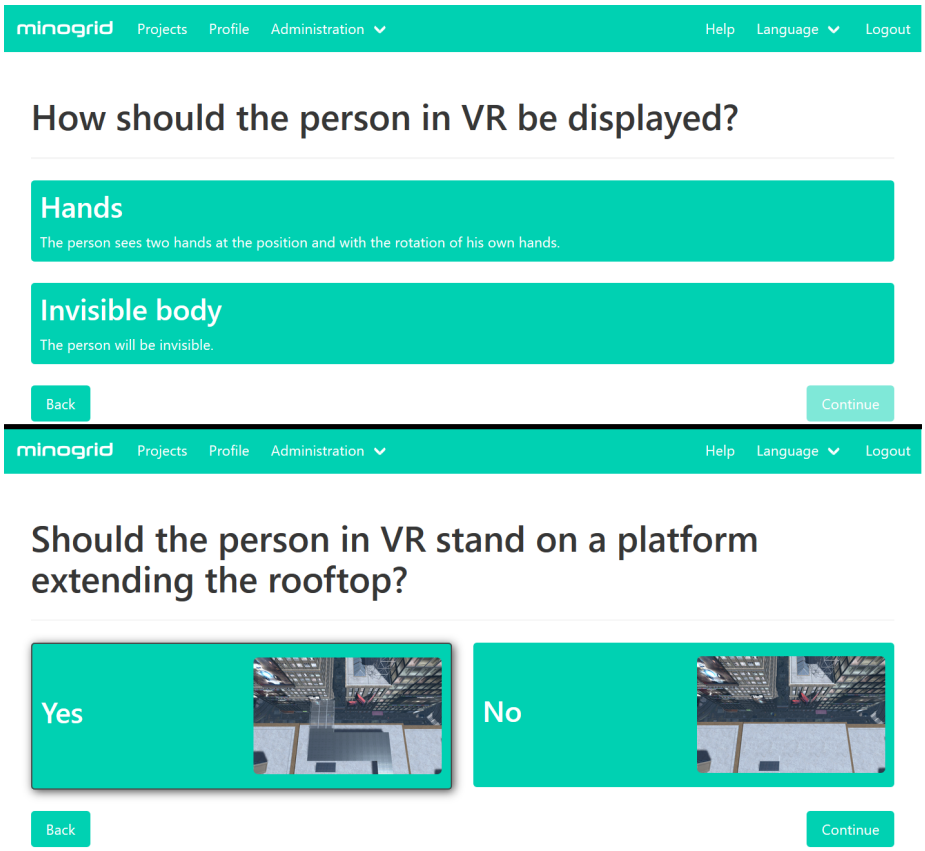

Figure 4.5: Two different questions about the environment that is to be created: The top question consists of one column ("twoCols": false) and both available answers have descriptions attached to them. The lower question consists of two columns ("twoCols": true) and both answers have a media item attached, but no description.

4.2 Technology

4.2.1 Overview

An overview of the employed technology and modules can be found in table 4.1 below:

Technology / Module	**Usage**
TypeScript	Back end and front end language
Node.js [36]	Server environment
Express.js [41]	Server framework
npm [37]	Node.js package manager
MongoDB [32]	Database
mongoose [45]	MongoDB object modeling
EJS [27]	HTML templating
Bulma [20]	CSS framework
Sass [34]	Stylesheet language
webpack [48]	Application packaging
helmet [2]	Back end security
bcrypt [35]	Password hashing
Archiver [9]	Dynamic archive creation
class-transformer [43]	Object transformation
dotenv [40]	Environment variable loading
express-session [12]	Session handling
jQuery [21]	DOM tree manipulation

Table 4.1: An overview of the technology and third party modules used by the web server.

4.2.2 Backbone

TypeScript was used as language for the back end of the web server. TypeScript is a superset of JavaScript developed by Microsoft. It adds optional static typing to JavaScript, allowing for more robust applications and reduced development time. In contrast to JavaScript, TypeScript must be compiled. The result of this process are traditional JavaScript files.

To lower development time, the web server relies on an already existing, state of the art web-framework, namely **Express.js** for **Node.js**. Node.js is an open source server environment that allows JavaScript to run on the server side. **npm**, is a Node.js package manager, was used to extend Node.js with already existing functionality. Express.js allows for easy setup of a web server, as it provides many helper functions to quickly add functionality to the server.

This server setup uses state of the art technology with a focus on easy extensibility. Due to this, many of the required features are already readily available,

resulting in a significant decrease in development time. Another advantage of this setup is that it presents an opportunity to use the same code in the backend as well as the frontend, reducing redundancy.

As database, **MongoDB** is used. MongoDB is a cross-platform document-oriented NoSQL database that works well with modern JavaScript applications since it uses JSON-like [3] documents for organizing and storing the data. **Mongoose** is used to model the application data. It provides mechanisms for type casting, validation, query building and more. This makes it possible to combine the flexibility and speed of MongoDB with at least some of the architectural benefits of traditional database systems.

4.2.3 Page Rendering

To render web pages, **EJS** (Embedded JavaScript templating) is used. EJS allows users to create HTML templates enriched with logic in the form of JavaScript code. This leads to a lower coding work load since pages can be reused and automatically adjusted depending on the content to be displayed.

To style the web pages, **Bulma**, a free, open source CSS Framework based on Flexbox that provides many CSS classes for easier and quicker styling as well as faster creation of layouts, is used. For additional styling, **Sass**, a preprocessor scripting language that compiles into traditional CSS, is used. Sass extends CSS by adding variables, nested rules, mixins, functions and other features for better code organization.

4.2.4 Security

To increase the security of the web server, **helmet** is used. It automatically sets various HTTP headers and uses 11 other middleware to provide various security mechanisms. To hash users' passwords for database storage, **bcrypt** is used.

4.2.5 Additional Helpers

To dynamically create archives for the user to download, **Archiver** is used. To cast plain objects deserialized from JSON to a class instance, **class-transformer** is used. To load some environment variables, like the server port and the database URI, from an external ".env" file in order to better decouple this configurable parameters from application code, **dotenv** is used. For session management, a middleware called **express-session** is used. Express-session stores a session ID in a cookie in the client's browser while storing the associated data on the server.

[3] JSON stands for JavaScript Object Notation and is an open standard data interchange format that stores data in a form accessible to humans. MongoDB uses BSON, which stands for binary JSON. It offers more data types than JSON and is, like the name suggests, not accessible to humans.

Like the back end, the front end uses TypeScript. For easier DOM tree manipulation, **jQuery** is used.

4.2.6 Organization

Webpack is used to automatically compile and copy all necessary files and run all necessary scripts. This reduces the time from code to running application by a significant amount and also allows for hot module replacement, which in turn allows for changes on a live, running application.

4.3 Architecture

4.3.1 Overview

The web application consists of multiple modules. Modules are usually made up of a router (responsible for providing endpoints), a service (providing the logic), and a model (providing data structures), though some modules may not contain all three. For example, the "common" module provides access to helper functions for EJS templates - this makes it possible to change content if necessary, such as when the user logged in has administrator rights. Since this functionality requires neither a router nor a model, this module only contains a service component. Table 4.2 gives an overview of the modules and their purpose.

4.3.2 Server Initialization

An "index.ts" file functions as the entry point for the web application. It initializes the Express.js server framework and starts a web server listening on a port specified by an ".env" file, which is loaded using dotenv. For initialization, the web application connects to a database with an URI also specified in the ".env" file.

The server exposes a public directory containing the compiled stylesheets and JavaScript files relevant for the front end as well as fonts, translation data and images. Next a router for every module that needs it (cf. table 4.2) is set up, providing end points for users and other modules to send requests to.

During server initialization, all used middleware are initialized. These encompass a middleware that logs events, all security focused middleware used by helmet and a middleware for session handling. Next the admin service tries to load SSL related data. If a private key file and certificate are found, the server initializes using HTTPS, if not the server is only available via the HTTP protocol. As the final step, webpack is configured to allow for hot module reload so that server development can take place during runtime.

Module	Route	Purpose	Main Functionality
User	/user	User management	Creation, removal, modification and authentication of users
Project	/projects	Project management	Creation, removal, modification and export of projects
Scene	/scenes	Scene management	Creation, removal and modification of scenes
Translation	/language	Translation handling	Translation of sequences to a defined language
Admin	/admin	Server setting management	Reloading certain server data
Debug	/debug	Providing debugging functionality	Logging data and creating test users
Root	/	Providing the root route	Redirecting from the root URL to the page the user expects
Common	-	Providing general functionality used by EJS templates	Providing access to the translation module and checking the users role

Table 4.2: An overview of the server modules and each module's purpose.

4.3.3 Rendering of a Page

To process of rendering a web page begins with a GET or POST request sent by the user. In a first step, the server checks if the request was sent from a user that is currently logged in. For this, the server checks if the session object contains a user and if this user exists in the database. If the user is not logged in but tries to access a route that requires the user to be logged in, they will be redirected to the start page with an according message.

Next, according to the user's request, the responsible module's router will call the functions of the service component. The service will handle data and contains the application logic. Once all data handling and computations are finished, the router will return an HTTP status and, depending on the request, render an EJS template. The router may expose data and functionality to this template, which in turn may contain logic that uses this data for functionality expressed through JavaScript code embedded into traditional HTML. For example, the router nearly always grants the template access to the user that is currently logged in and to the functionality defined in the common module. The template then, among other things, changes the menu on the top to either show a button that allows users to login or, if they are already logged in, to logout.

4.3.4 Data Model

As discussed in section 4.2, the web application uses a MongoDB database. Since MongoDB is a document based NoSQL database, the schema is defined inside the web application using mongoose. The defined schema is relatively simple, consisting only of one collection[4] named "Users". As the name suggests, this collection stores all users. Figure 4.6 shows this data model as class diagram.

A user consists of an email, a password, a first name, a last name and an optional organization. Additionally the user has a role (either administrator or user) and any number of projects. The email and password are used to authenticate the user. The field concerning the user's organization is currently not in use and is merely a place holder for the future possibility of offering license sharing agreements, for example between members of a university.

A project stands in for a Unity project. It consists of a name, a related Unity version and a Minogrid version. Additionally, a project has any number of scenes. The name is just for organizational purposes. The Unity and Minogrid versions are checked by the Unity plug-in.

Scenes represent Unity scenes. A scene consists of an environment ID, an environment name, an index, any number of base options and any number of environment options. The environment ID is the ID of the predefined environment; for example, the ID could represent an environment of a city with skyscrapers or an apartment. The server, as well as the Minogrid plug-in, know which ID represents which environment - this ID is never directly shown to the user and is only in use behind the scenes. The environment name, however, is entered by the user. It is used to recognize the created scenes inside the Unity engine. The index stores the order of the created scenes so it can be displayed inside the Unity engine.

The base options and environment options arrays are both of the type "minogridOption" which consists of an question, represented by a number, and an answer, which is also stored as a number. These number pairs define the scene. The base options are options that every scene has, for example how the player looks like, while the environment options are dependent on the selected environment, for example how high the skyscraper in the city scene is.

[4] The SQL counterpart of a MongoDB collection is a table.

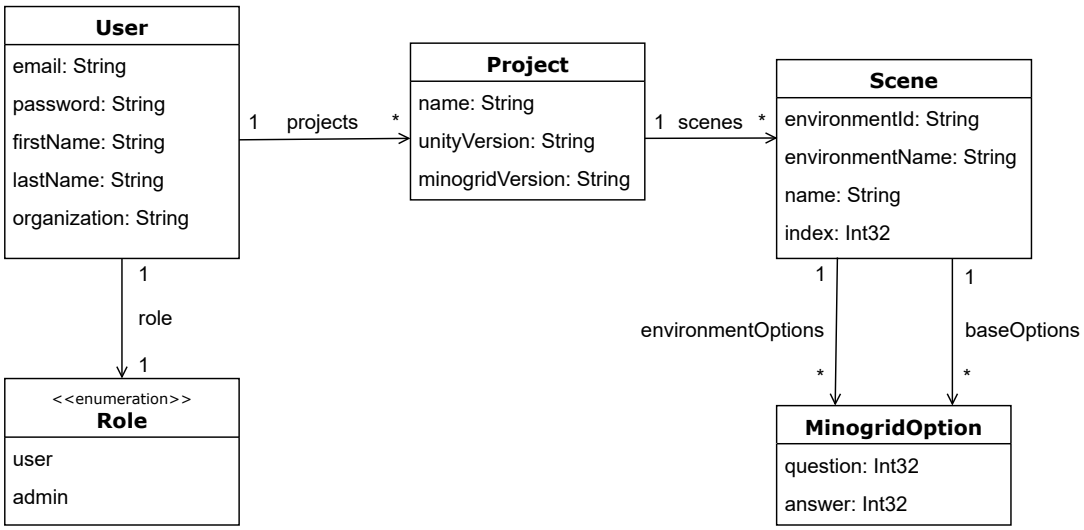

Figure 4.6: A class diagram representing the database schema.

4.3.5 Environment System

An environment represents an already existing Unity scene. As discussed in section 4.3.4, these existing environment are modified through parameters set by the user.

On the server, these environments and their parameters are defined by human readable JSON files. Listing 4.1 shows an excerpt of the skyscraper environment definition. The "id" and "name" correspond to the data fields discussed in section 4.3.4, the "tags" are shown at the environment selection at the start of the scene creation process. The "description" is a keyword that is looked up in a corresponding environment translation file (more information on this can be found in section 4.3.7). The translated text will also be shown on the environment selection screen. The "media" text links to a picture in the servers public directory and is also be shown to the user via the environment selection screen.

The "sharedDatasets" array defines a number of asset packs that are used by these environments. This allows for efficiently packing together assets that are used by multiple environments, since this data needs to be included only once even though it is used by multiple scenes. The "sceneQuestions" define a number of questions to customize the environment. The "id" once again corresponds to the ID of one of the options discussed in section 4.3.4, though now the "question" is translated into a question text. The parameter "twoCols" changes the appearance of the answers, as shown in figure 4.5.

All environments are stored in a folder containing the corresponding configuration file, a translation file storing the translations of the sequences in the configuration file and a folder containing Unity data the user has to download when using this environment. Besides the selectable environments, one "base" environment exists defining the base options, for example what the player body looks like and the base Unity files, for example the game management scripts.

The array "dependencies" define a number of question-answer combinations that have to be met for this question to show up. One example would be a question about the appearance of an optional object. This question would only make sense if the user decided to include this object in an earlier question. A dependency is an object consisting of the id of the question that should be checked and an array of valid answers.

The array "answers" contains all possible selectable answers. Each "id" corresponds a possible answer choice, as discussed in section 4.3.4. The "text" is once again a keyword that will be translated to another text. The "description" is an optional text that is shown on the question screen, the "shortdescription" a very short description that is shown on the screen summarizing a project. This enables users to quickly get an overview of the options they selected. The "media" is an optional link to a picture stored in the server's public directory and shown to the user via the question screen.

4.3.6 Project Export

When downloading a project, the server will bundle all necessary files together and create an archive that is sent to the user's browser. The data that is packed into this archive consists of the environment files discussed in section 4.3.5, the Minogrid Unity plug-in and a generated configuration file.

The resulting archive is a ready-to-use Unity project, which means that after downloading and installing the required Unity version, users can directly import the downloaded and extracted archive.

Since the plug-in lies in a special project directory defined by Unity, Unity instantly recognizes that it is an editor plug-in and acts accordingly. More details about this can be found in section 5.

The configuration file is dynamically created by the project data structure the user downloads. This configuration file contains the general project information (consisting of the project name, the target Unity version and the target Minogrid version) as well as an array containing all project scenes. As discussed in section 4.3.4, a scene consists of an ID, a name, an environment name, an index, one array "baseOptions" holding the general scene configuration and one array "environmentOptions" holding the environment specific scene options.

All this data is stored in a JSON file that gets parsed and processed by the Minogrid plug-in once the user opens the project.

Code Snippet 4.1: An excerpt of the skyscraper environment definition.

```
{
  "id": "001",
  "name": "skyscraperRooftop",
  "tags": [
    "acrophobia",
    "brontophobia"
  ],
  "description": "skyscraperRooftopDescription",
  "media": "skyscraper/skyscraper.png",
  "sharedDatasets": [
    "city-1"
  ],
  "sceneQuestions": [
    ...,
    {
      "id": "2",
      "question": "platformDesign",
      "twoCols": true,
      "dependencies": [
        {
          "id": "1",
          "answers": [
            "0"
          ]
        }
      ],
      "answers": [
        {
          "id": "0",
          "text": "opaque",
          "description": "",
          "shortdescription": "opaqueShort",
          "media": "skyscraper/platform_metal.jpg"
        },
        ...
      ]
    },
    ...
  ]
}
```

4.3.7 Translation System

To allow users with different language requirements to use the platform, a translation system was implemented. Users can select their preferred language in the upper right corner of the page; the selected language will be stored in the session discussed in section 4.2. The translation system uses translation items that consist of a keyword ("abbreviation") and multiple languages that this keyword can be translated into. Listing 4.2 shows the definition of the translation of the keyword "userDeletionConfirmation". Additionally, it is possible To define variables ("vars") that will be replaced when translating the text.

Code Snippet 4.2: A translation item that defines the translations of the keyword "userDeletionConfirmation" in English (en) and German (de).

```
{
    "abbreviation": "userDeletionConfirmation",
    "languages": [{
        "language": "en",
        "translation": "Please confirm the deletion of
            the profile of [firstName] [lastName] ([
           email])."
      },
      {
        "language": "de",
        "translation": "Bitte bestätigen Sie das Lö
           schen des Profils von [firstName] [lastName
           ] ([email])."
      }
    ],
    "vars": ["firstName", "lastName", "email"]
}
```

These translation items are defined in multiple different JSON files - one file for the general web application interface and one file for each environment. These files are loaded into server memory at server startup and then used by a translation service to translate occurring keywords.

When translating, the translation service searches for a translation item with the specified keyword; a keyword that is not found will not be translated. If found, the translation service tries to locate a translation in the target language. If this translation does not exist, the first defined translation (generally English) will be used. As the final step, the translation service seeks out any variables marked by brackets and defined in the "vars" array and subsequently replaces them with word pairs specified by the function caller. If a "var" is not defined by the function caller it will be ignored.

4.3.8 Front End

Like the back end, the front end uses Typescript. The application structure is very simple, consisting of only six source files. An index file initializes all button listeners using jQuery. The primary function of the front end scripts is to send requests to the appropriate routes of the server when a button is clicked and to display and hide elements based on user input.

5 Design and Implementation of the Engine Plug-in

5.1 Design Considerations

The Unity plug-in acts as the connection between the data generated by the web application and the Unity engine. It is packed into the package users download from the server - this means that the imported Unity package already contains the plug-in.

Once in Unity, users can open a "Minogrid" window, shown in figure 5.1, via the top bar of the editor. Instructions on how to do this are provided to the user in the web application, as discussed in 4.1.

Users can use this window to change build settings and build the application. It contains options to toggle whether the application should output audio and whether the application should capture a session's data. With the button "Build application", users can select a path for the built application to be stored and start the Unity building process. Once this process has been completed, users have a ready-to-ship executable file containing all the necessary data.

As discussed in section 4.3.6, the configuration is stored in a JSON file which in turn is stored inside the project. This file is loaded during plug-in initialization, which happens when the project is opened. If a user has manually changed the configuration, they can update the loaded configuration by pressing the button "Update configuration".

The button "Show captures" changes the menu to the layout shown in figure 5.2. This menu shows all recorded sessions grouped by date and time. Once a session is selected, multiple subentries, one for every scene recorded during this session, are displayed. Users can start the playback of the selected capture, export a video from this capture or delete it.

At the bottom of both plug-in menu pages users can change the display language. The current options are the same as the web application options, namely English and German. The plug-in window itself is, like every Unity window, dockable inside the Unity editor, which means that users can create their own engine layout should they so desire.

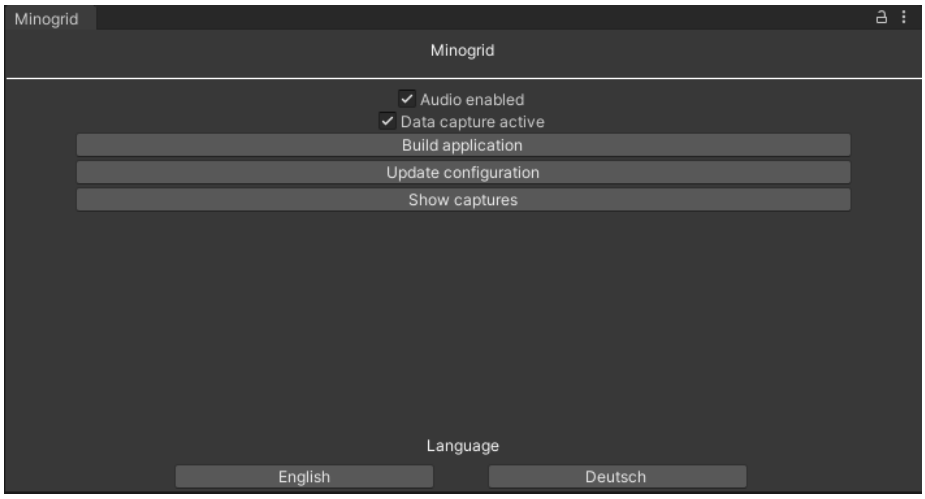

Figure 5.1: The main menu of the plug-in; it gives users the option to disable/enable audio, disable/enable data capturing, build the application, reload the configuration and show capture data.

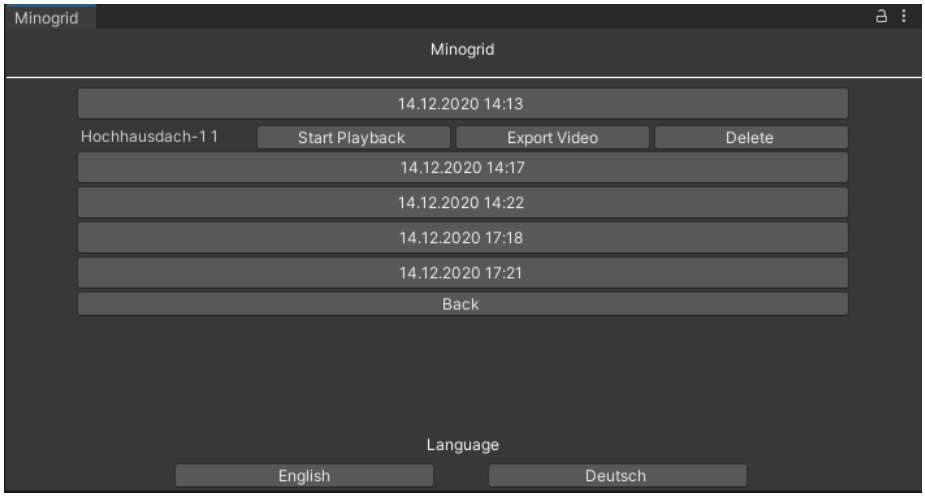

Figure 5.2: The evaluation menu of the plug-in. Captures are grouped by date and time, and every scene is one capture item. Users can playback the captures, export them to a video or delete them.

5.2 Architecture

5.2.1 Overview

Unity differentiates between two different plug-in types: "Managed plug-ins" and "Native plug-ins". Managed plug-ins are managed .NET assemblies. Since Unity uses the .NET tools to compile scripts, it can access managed plug-ins. This also implies that, like all Unity scripts, managed plug-ins use C# as programming language. Native plug-ins can be written in C, C++ and Objective-C. Developers can access functions from these native plug-ins in their game scripts. This allows for integration of middleware libraries or existing C/C++ game code. Since the goal of this thesis was to create new code, a new managed plug-in was created.

To use the plug-in inside Unity, it is built to a DLL [5] file. This file is stored in a directory called "Editor" whose parent is a folder called "Plugin". This way, Unity recognizes the plugin as a managed editor plugin, attempts to load it at startup and does not include it in application builds.

The plug-in consists of several modules. Every module is encapsulated in one C# source file. The function of every plugin is listed in table 5.1 below:

Module	**Functionality**
Manager	Plug-in initialization
Logger	Event logging
Window	UI definition and management
Builder	Application building
ConfigManager	Configuration management
Capture	Data capture management
Translator	Translation management

Table 5.1: An overview of the plug-in modules.

5.2.2 Data Management

At startup, the plug-in tries to load the JSON configuration file. Unity includes a "JsonUtility" that allows the creation of objects from JSON files. The configuration data structure, discussed in 4.3.6, is also defined inside the plugin, which means that the configuration JSON can automatically be converted to a configuration object. The configuration definition of the plug-in has two fields the configuration definition on the server lacks: "soundAllowed" and "captureAllowed". These two new fields correspond to the two enable/disable options in the plug-in UI, discussed in section 5.1.

[5] Dynamic-link library (DLL) is a shared library on Microsoft Windows

To allow the running application to change according to the selected configuration, the application has to have access to it. To allow a running Unity application to access a static file and include it in a build, it is necessary to store the file in a special location, namely a directory called "StreamingAssets" inside the project directory. Since it is possible to change the configuration with the plugin-in, it first reads the configuration from the location the server has put it and then writes it to the "StreamingAssets" directory. While the original configuration file is never modified, the configuration file used by the built application is overwritten every time the plug-in loads the original configuration, or the user decides to change the current configuration using the plug-in.

To make scene recognition in a running application easier, thumbnails are displayed on the screen selection screen. Like the configuration file, these thumbnails have to be copied into the "StreamingAssets" directory. Both these tasks, configuration file management and thumbnail copying, are done by the ConfigManager module.

The capture files are stored in the "persistentDataPath". This location is defined by Unity so that running applications can read and write to this directory. Using this directory, the plug-in is able to read all existing captures that were created with any application built with Minogrid on this specific device. The metadata of each capture (date, time of creation and scene names) is encoded in the directory and file name of each capture file inside the "persistentDataPath" and parsed by the plug-in.

Once a user wants to playback a specific capture or export it as a video file, the plug-in creates a temporary ".token" file containing the path to the capture file that should be processed as well as the information on how it should be processed (playback or conversion to video). Next, the plugin starts the Unity application inside the editor. If the application is executed inside the editor, it will check for ".token" files inside the "persistentDataPath" and, if any are found, act accordingly (this will be discussed in more detail in section 6.3.4). The reading of and writing to the capture file is, like the name suggests, done by the Capture module.

5.2.3 UI

The UI is defined inside the window module. At startup and every time an UI event happens, Unity calls the "OnGUI()" function. The UI is defined by calling UI functions inside the "OnGUI()" function. For example, to create a label the "Label()" function is called, to create a button the "Button()" function is called. It is possible to create different kinds of UI elements and layouts by calling several of these functions in certain defined order. Some of these functions provided by Unity also allow to specify event callbacks; an example of this would be the specification of a function that will be called once a button is pressed.

To manage the two different plug-in menus - the main menu and the evaluation menu - an enum "Menu" and a simple if statement are used. Depending on the state of the enum, different elements are rendered inside the "OnGUI()" function.

5.2.4 Application Building

The builder module can, as the name suggests, build the application to an executable. To do that it adds all scenes defined in the configuration file to the Unity build settings. This is done by calling functions defined by Unity with the path to the target scene as argument. Since Minogrid has a fixed scene path pattern, the plug-in can assemble all scene paths with the "environmentName", discussed in section 4.3.5. In addition to the base scene and the user created scenes, light settings scenes are also added to the build settings (these will be discussed in more detailed in section 6.3.5). The build process itself consists of the user specifying a target path and the plug-in calling the appropriate Unity functions and setting the right build options.

5.2.5 Other Modules

The logger and translator modules are C# ports of their TypeScript counterparts on the server. Thus the translator module uses JSON files of the structure discussed in section 4.3.7. Like the plug-in itself, the translation file is stored in the project exported by the server. The log files from the plug-in are stored inside the project directory as well.

The manager module initializes the other modules at startup. This includes loading the configuration, writing the configuration and scene thumbnails and adding the scenes to the build settings (though not building the application).

6 Design and Implementation of the Virtual Reality Environments

6.1 Design Considerations

6.1.1 Overview

The third part of the Minogrid framework are the Unity scenes themselves. A Unity scene is a 3D environment coupled with logic and rendering settings. The focus of this work lies on one base scene. All environments are Unity scenes that are loaded in addition to the base scene. For this thesis, two different environments, both focused on acrophobia [6] and brontophobia [7], were created.

While the environment is in both cases a cityscape consisting of skyscrapers, the cities themselves are different. In one scene, the player stands on the rooftop of a skyscraper; in the other scene, the player starts in a flat or the balcony connected to this flat.

The idea is that the rooftop scene conveys a sense of freedom while at the same time making the player feel more vulnerable. While it rains or a thunderstorm is raging, the player is situated in the center of the action - the rain hits the body in the virtual world. The flat, on the other hand, provides the player with a safe space, walls and a rooftop shielding them from the elements and the outside world.

6.1.2 General Options

As discussed in section 3.5, a base scene exists that defines logic and options relevant for all environments. These general options are:

- Player body
  Regarding the player body, a user has two different choices: Either the player can see their hands or they have no body at all.
- Body material
  If the user enables the option to see the player's body's hands, they can choose the material of the hands. There are six choices: Blue, light skin and dark skin, each either translucent or opaque.
- Time of day
  Users can choose if the scene should take place in the daytime or at night.
- Weather
  There are three different weather situations available: Clear, rain and thunderstorm. The lighting in the rain and thunderstorm settings is same, but lightning and thunder sounds are added in the thunderstorm setting,

[6] Fear of heights
[7] Fear of thunder and lightning

resulting in four different lighting scenarios overall: Clear day, cloudy day, clear night and cloudy night.

6.1.3 Skyscraper Rooftop Scene Design and Options

Within this environment, the user starts out standing on a rooftop in the center of a city consisting of other, higher skyscrapers. The other skyscrapers are higher to reduce a player's fear to be hit by lightning during a thunderstorm. Users can alter the rooftop environment in various ways:

- Rooftop height
  The rooftop height determines how high the player's starting position is. There are four options: At maximum height, high, near the ground or on the ground. If the user chooses to create an environment where the player starts on ground level, the rooftop will be at maximum height but the player will be standing in the street down below.

- Platform
  Users can choose whether a platform should extend from the front of the building, as shown in figures 6.1 and 6.2.

- Platform design
  If the user chooses to add a platform, they can adjust the appearance of this platform by choosing the material it appears to be made of: Glass, which allows the player to see through the bottom of the platform to what lies below, or metal, which is opaque.

- Railings
  Users can choose whether there should be safety railings of not. This applies to the platform if it is active and to the rooftop itself if it is not.

- Starting position
  The starting position of the player when first loading the scene. There are four different options available: Near the abyss or near the center of the rooftop and looking at the abyss or looking away from the abyss respectively. The starting position is also adjusted according to whether the platform is available in the chosen scenario or not.

Coupled with the general options discussed in section 6.1.2, these settings allow for vastly different experiences. Settings that influence the player's perception of the scene in meaningful ways and that touch on the phobias relevant to this environment were chosen. There is also room to gradually increase the severity of the situation. One could, for example, start at street level to get accustomed to the VR environment and then slowly progress through different scenarios to the rooftop, finally ending up standing on a glass platform at maximum height. Figure 6.4 shows how the lighting of this environment changes depending on the selected time of day and the chosen weather conditions.

Figure 6.1: A bird's eye view of the rooftop scene.

Figure 6.2: A frontal view of the rooftop scene.

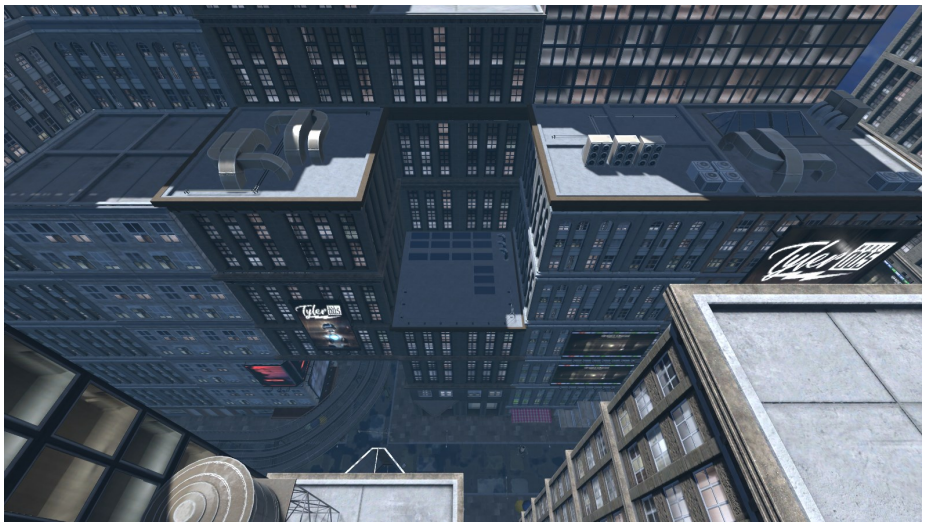

Figure 6.3: A variation of the skyscraper rooftop: Here, there is no platform, the railings are enabled and the height is set to "high".

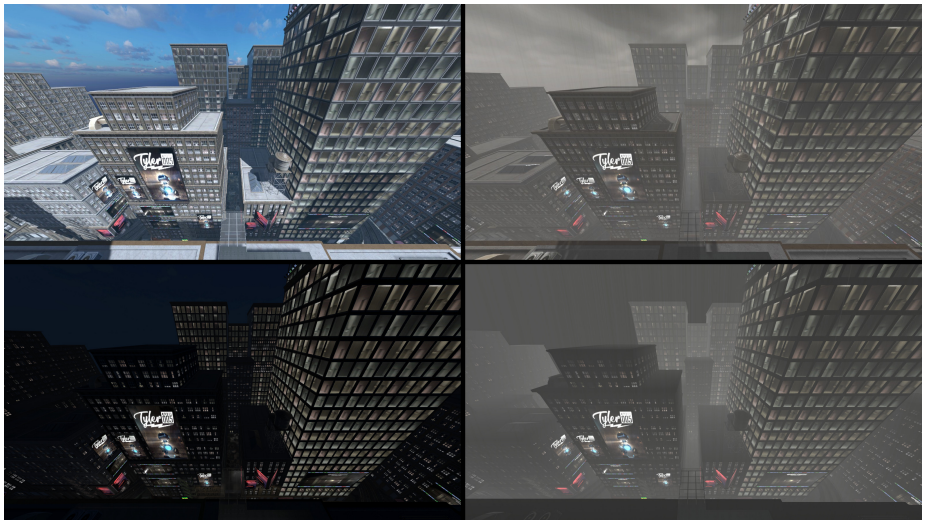

Figure 6.4: The four different lighting scenarios of the rooftop environment. From the top left to the bottom right the scenarios are: Clear day, cloudy day, clear night and cloudy night.

6.1.4 Skyscraper Flat Scene Design and Options

When choosing the flat as environment, the user starts out inside a flat or on that flat's balcony, the flat being situated in the center of a city made up of skyscrapers. The flat environment uses the same assets for the city as the rooftop scene, though they are laid out in a different matter, creating an overall different look. Like the rooftop environment, this environment can be altered via multiple settings:

- Flat height
  Like the rooftop, the height of the flat is changeable. There are two possible options: High and low. Figure 6.6 shows the outlook of the "high" flat.

- Door
  The user can decide if the door to the balcony should be open or closed. Depending on this and the weather settings, the curtains display a different intensity of movement, since the wind intensity changes based on these settings. Figure 6.5 shows the flat with an open door.

- Curtains
  This setting determines if the curtains of the flat are open or closed. When the curtains are closed and the player wants to switch from the outside to the inside or the inside to the outside, they have to move the curtains with their hands or their body. The physics of the curtains are simulated with the Unity "Cloth" component. Figure 6.5 shows the flat with the curtains pulled back.

- Railings
  Users can change the railings of the balcony. There are three different options available: No railings at all, "half" railings, as shown in figure 6.6, and "full railings", as shown in figure 6.7.

- Starting position
  The player's starting position when loading the scene. There are four different options available: On the balcony looking either out at the city or back into the flat, in the flat near the balcony doors looking towards the balcony or further inside the flat looking towards the balcony.

The four different lighting scenarios of the flat are shown in figure 6.8. The balcony view of these lighting scenarios is shown in figure 6.9. Like the options of the rooftop scene, these options were chosen with the targeted phobias in mind. Each choice changes the experience in a meaningful way and allows for gradual progression.

Figure 6.5: The interior of the flat.

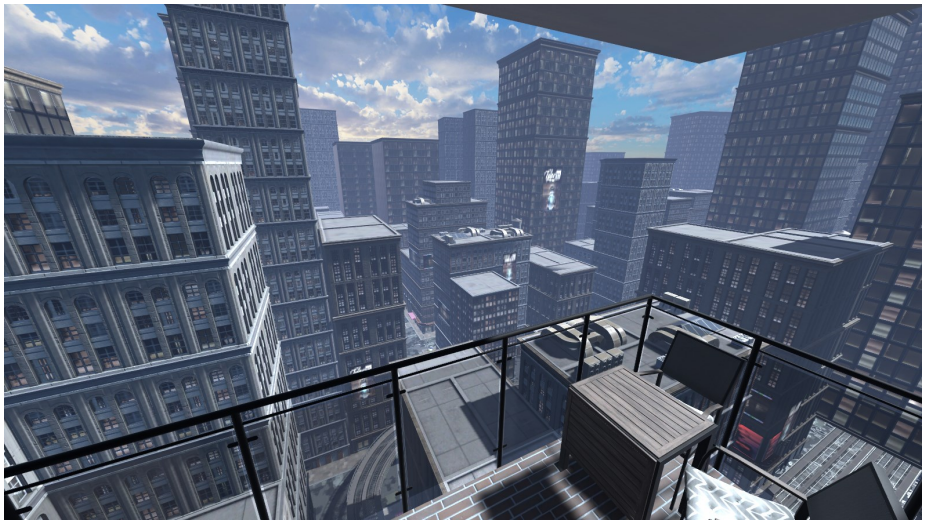

Figure 6.6: The view from the balcony of the flat.

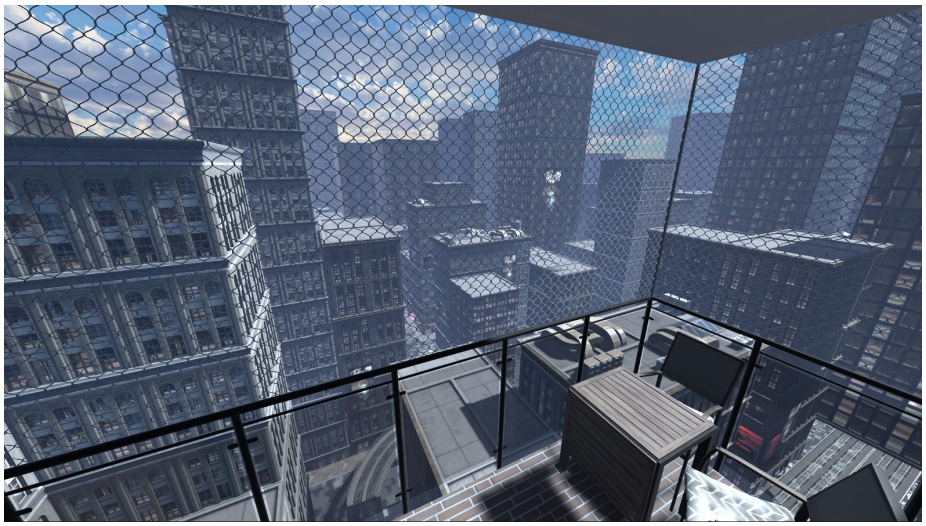

Figure 6.7: The view from the balcony of the flat with full railings.

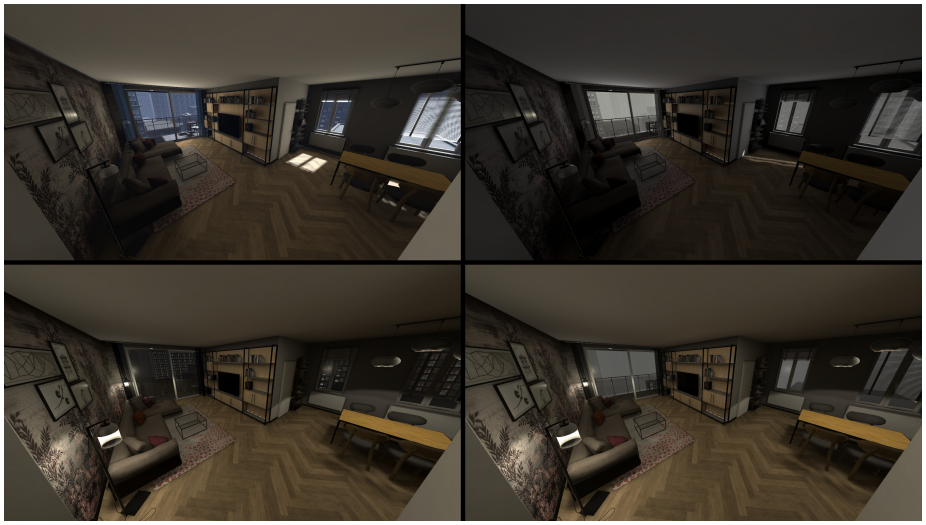

Figure 6.8: The four different lighting scenarios of the flat environment as seen from inside the flat. From the top left to the bottom right the scenarios are: Clear day, cloudy day, clear night and cloudy night.

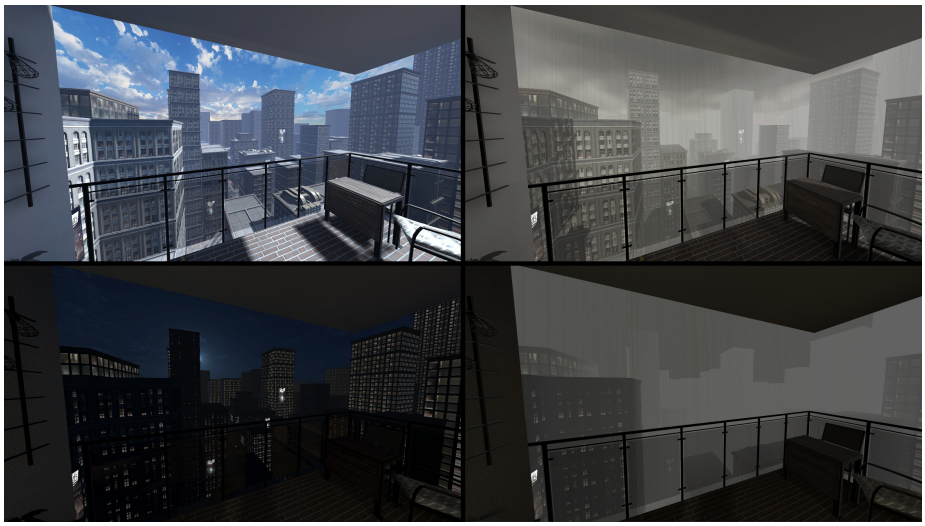

Figure 6.9: The four different lighting scenarios of the flat environment as seen from the balcony. From the top left to the bottom right the scenarios are: Clear day, cloudy day, clear night and cloudy night.

6.1.5 Second Screen

To allow researchers and therapists to monitor the VR session, a second screen was implemented. This second screen allows other parties to see what the person in VR is currently seeing as well as to switch between scenes, initiate or end a pause in the VR environment and exit the application.

Figure 6.10 shows this second screen. The project name is displayed at the top while the user controls are situated on the right-hand side. The ID is associated with the session and is used for evaluation purposes, so that the capture of this session can be found more easily. The items below the ID input field show all scenes of this project. It is possible to switch between the different scenes during runtime. A press on "Pause" fades the screen in the VR to black and disables sound so that they can communicate with others. The "Exit" button closes the application.

When playing back a capture, the second screen changes layout and becomes the main screen, as a scene switch is no longer possible and there is no active VR session running. Figure 6.11 shows this layout. Users can play/pause the playback in the lower left corner of the display. It is also possible to change playback speed between -2x and 2x in 0.5 steps. It is possible to see and change the current playback position with the bar at the bottom of the screen. The playback image also refreshes while skipping through the capture with the playback bar, which makes navigating a capture is as easy as possible.

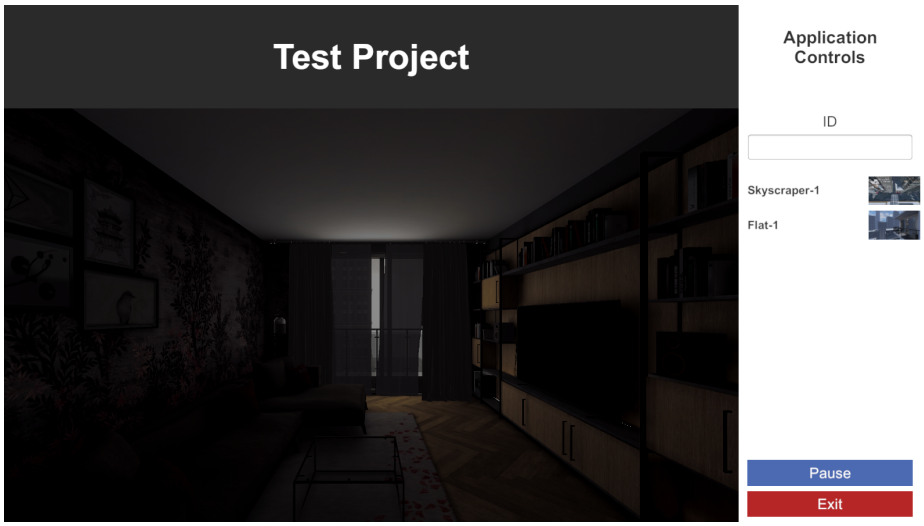

Figure 6.10: The second screen shows what the person in VR sees and offers simple controls during a session; it is pictured here set to the rainy flat environment with an open door and closed curtains.

Figure 6.11: The second screen becomes the main screen during capture playback.

6.2 Technology

6.2.1 Overview

As thoroughly discussed in previous sections, this project uses the Unity engine. The programming language of Unity scripts is C#, so naturally the used programming language for this component is C#.

Unity allows users to select one of three different preconfigured rendering pipelines: The High Definition Rendering Pipeline (HDRP), the Universal Render Pipeline (URP) and the Built-in Render Pipeline. HDRP and URP are both relatively new, "scriptable" render pipelines, allowing users to customize many aspects of the rendering. The Built-in render pipeline is much older and comes with limited customization options.

The difference between HDRP and URP is the target platform and intended use case: HDRP is very taxing on the hardware and allows for many different high-end visual effects. URP is made for a broader range of devices and looses some realism compared to HDRP. Since the goal is to allow users with different ranges of hardware to run this project at 70 frames per seconds upwards in a VR environment, the URP was chosen for this project.

6.2.2 Third Party Assets

Package Name	**Author**	**Functionality**
OpenVR Unity XR Plugin	Valve Corporation[1]	One interface for rendering on all major VR devices
No package / Custom made for this project	Karoline Brabenetz[2]	The flat from the flat environment, various 3D models on the rooftop in the rooftop environment
City Builder: Urban	Reversed Interactive[3]	Various 3D models of city objects (skyscrapers, streets etc.)
Universal Sound FX	Imphenzia[4]	A package containing a large number of sound effects
AllSky Free	Richard Whitelock[5]	A collection of sky boxes
lightmap-switching-tool	Laurent Harduin[6]	A tool to switch between pre-baked lighting scenarios

[1] https://www.valvesoftware.com/
[2] https://k-brabenetz.github.io/index.html
[3] https://www.reversedinteractive.com
[4] https://www.imphenzia.com/
[5] http://www.richardwhitelock.com/
[6] https://laurentharduin.fr/

Table 6.1: An overview of the Unity asset packs used

Since the creation of Unity assets such as 3D objects and sounds is not the focus of this project, it was decided to use already existing asset packs from the Unity asset store. Table 6.1 presents an overview of the third party assets that were used.

6.3 Architecture

6.3.1 Overview

This section describes the basic Unity framework that was implemented. While developing this framework, the main objectives were code readability, easy to understand structures and well defined interfaces in order to make it as easy as possible to add new environments and change existing environments.

Most of the logic is defined in one base scene (more information on this can be found in section 6.3.2). All environments are loaded on top of this base scene. This section will focus on showing and explaining the architecture behind this base scene and how it works together with other environments, rather then describing the two environments implemented for this thesis in great detail.

Every Unity scene has its own hierarchy consisting of "GameObjects". Various components can be attached to these entities, allowing them to change appearance and behaviour. Scripts that inherit from a class "MonoBehaviour" can also be attached to such GameObjects and inherit a number of useful functionalities in order to work with other Unity systems, such as an update function that is called once per frame.

To reduce the length of this thesis and increase readability, this section will not focus on the Unity GameObject hierachy but instead focus on the bigger picture: The software architecture that results when all these scripts are combined. Looking at the base scene through this lens, one can group the implemented functionalities into the following subsystems:

- Scene Management
  Scene management encompasses the loading and unloading of scenes and the initialization of these scenes.
- Player
  The player subsystem handles player movement and animation.
- Data Capture
  If the data capturing subsystem is enabled, data of the various other subsystems is captured and can be played back at a later time.
- UI
  The UI subsystem manages the input given from the second screen or the main screen when playing back a capture.

6.3.2 Scene Management

Unity scenes bundle virtual 2D or 3D environments and logic together into one unit. One could imagine Unity scenes like levels of a jump-and-run game - in this analogy, one scene would be one stage. However, conversely, since one can manipulate nearly every aspect using scripts, it would also be possible to create one scene and then modify this scene depending on the stage. While this appears to work against the idea of scenes it is actually possible, hence the definition that a Unity scene is a bundle of a virtual 2D or 3D environment and logic.

As discussed in section 6.1.1, one base scene contains all the basic logic and systems needed for all other environments, and all other environments are loaded on top of this base scene. The "GameManager" is responsible for scene switching and initializing.

Once a scene is selected on the second screen, the GameManager clears up the current scene, if one is currently loaded, and starts loading the new one. During this process the player screen fades to black. Once the new scene is loaded, the GameManager tries to find an object of the type "MinogridSceneManager".

Every scene except the base scene contains a MinogridSceneManager. This manager is responsible for all logic regarding this one scene. For example, the flat scene manager controls the curtain movement inside the flat.

Once found, the scene manager takes over scene initialization from the GameManager. In this step, all options the user set are applied and scene geometry, lighting and logic are modified to match the configured settings. The MinogridSceneManager defines the following functions, which are overwritten by scene managers extending the abstract MinogridSceneManager:

- InitializeWithOptions(List<MinogridOption> sceneOptions)
  Initializes the scene based on the options set by the player.
- SetSceneSpecificPlaybackSpeed(float playbackSpeed)
  Updates the scene during capture playback if the playback speed changes. For example: The scene manager of the flat environment changes the simulation speed of the curtains.
- UpdateSceneSpecificWeather(TimeOfDay timeOfDay, Weather weather)
  Updates the scene based on the currently set time of day and weather. While functionality that switches the basic light setting is already implemented in the MinogridSceneManager, some environments, like the flat environment, need additional changes depending on the weather. One example of this is the curtain movement, which is more intense if a thunderstorm is raging.

This system makes environment creation and modification easy. To modify the impact of user choices, logic merely needs to be injected into one of these three

functions. When creating a new scene it suffices to create a class extending the MinogridSceneManager, implementing these three functions, and add this class to a GameObject. Finally, some fields defined by the MinogridSceneManager have to be set which hold references to the specific objects every environment needs. These fields are:

- LocalLevelLightmapData
  An object containing the lighting settings for this scene (more about this in section 6.3.5).
- rainSystems
  An array holding all particle systems that resemble rain in this scene.
- rainAudios
  An array holding all audio sources that resemble rain sounds.
- thunderstormController
  An object of the type "ThunderstormController", responsible for managing the thunderstorm if that weather option is set.
- thunderstormCatcher
  A Catcher object that captures lightning strikes during a thunderstorm so that when playing back a scene the same lightning strikes hit the same positions.

6.3.3 Player

The core of the player subsystem consists of three different classes: The "PlayerManager", which manages the other systems of the player, the "PlayerInput", which takes care of controller inputs, and the "PlayerMovement", which actually moves the player's body.

Once initialized, the PlayerManager initializes the PlayerInput and PlayerMovement objects. The PlayerInput Object uses the Unity "XRNode" system. Using this system, all trackable body parts are encapsulated in XRNodes. The PlayerInput stores the current state of all relevant XRNodes and updates them regularly. The PlayerMovement object updates the position and rotation of all body parts using the data stored in the PlayerInput object.

Apart from movement, the PlayerManager also deals with audio output, can fade the screen to black and return the player to the scene. This functionality is used when switching between scenes or pausing the application.

6.3.4 Data Capture and Playback

The data capturing system does not capture a video as such, but instead every relevant event that needs to be recreated for playback is stored with a timestamp. Compared to traditional video capture this offers a number of advantages. The

CPU and GPU workload is reduced, since no real time video encoding takes place. This also greatly reduces the amount of storage needed, since text only requires a small fraction of the storage space a video would need.

Objects inheriting from the class "Catcher" can register themselves to a "MinogridCapture" capture managing object. These objects can capture and playback certain events, defined by a class called "CaptureElement". A CaptureElement contains a type, a timestamp and data - all three of these properties are encoded as strings. Each "Catcher" is responsible for CaptureElement objects of a certain type. The MinogridCapture object controls if these "Catcher" objects shall capture events or play them back.

When capturing the MinogridCapture stores a list containing all captured events. At the end of each scene, this list as well as the environment settings used in this scene are written to disk encoded as a JSON file.

When playing back the list from this JSON file is loaded into memory. Depending on the current playback position, the MinogridCapture object will try to find a "Catcher" that can playback the CaptureElement nearest to this timestamp.

To illustrate the capturing mechanic with an example: When data capturing is active and the scene starts, a PlayerMovementCatcher registers itself with the MinogridCapture object. This MinogridCapture object will then start the data capturing mechanism of this "Catcher". The PlayerMovementCatcher notifies the MinogridCapture object 30 times per second about the current player's head and hand positions. The positions and rotations of these objects are encoded within a string making up the "data" field of the CaptureElement; the event type is "movement". Once the scene is finished, the MinogridCapture writes all captured data to disk. When playing back this capture, the PlayerMovementCatcher will once again register itself with the MinogridCapture object. Since this is a playback, the MinogridCapture will not start the data capturing mechanism. Trying to playback all recorded events, the MinogridCapture object will check if the PlayerMovementCatcher can handle playback for each of the events. If an event is of the type "movement" the PlayerMovementCatcher will indeed be able to play it back, decode the data stored in the data field and set the positions and rotations of the player's head and hands.

This approach makes it possible to easily create and add new event types that can be captured and played back without the need of altering the logic of the base scene. Additionally, a "MinogridRecorder" can record a capture playback. With this class, it is possible to export existing captures as video.

6.3.5 Light and Weather System

As discussed in section 6.1.2, there exist six possible combinations of the light and weather settings: Day and night, each either clear, rainy or thundery. As

mentioned, the rain and thunderstorm, while representing different weather settings, do not result in different lighting scenarios. This results in four different lighting scenarios for every environment: Clear day, cloudy day, clear night and cloudy night.

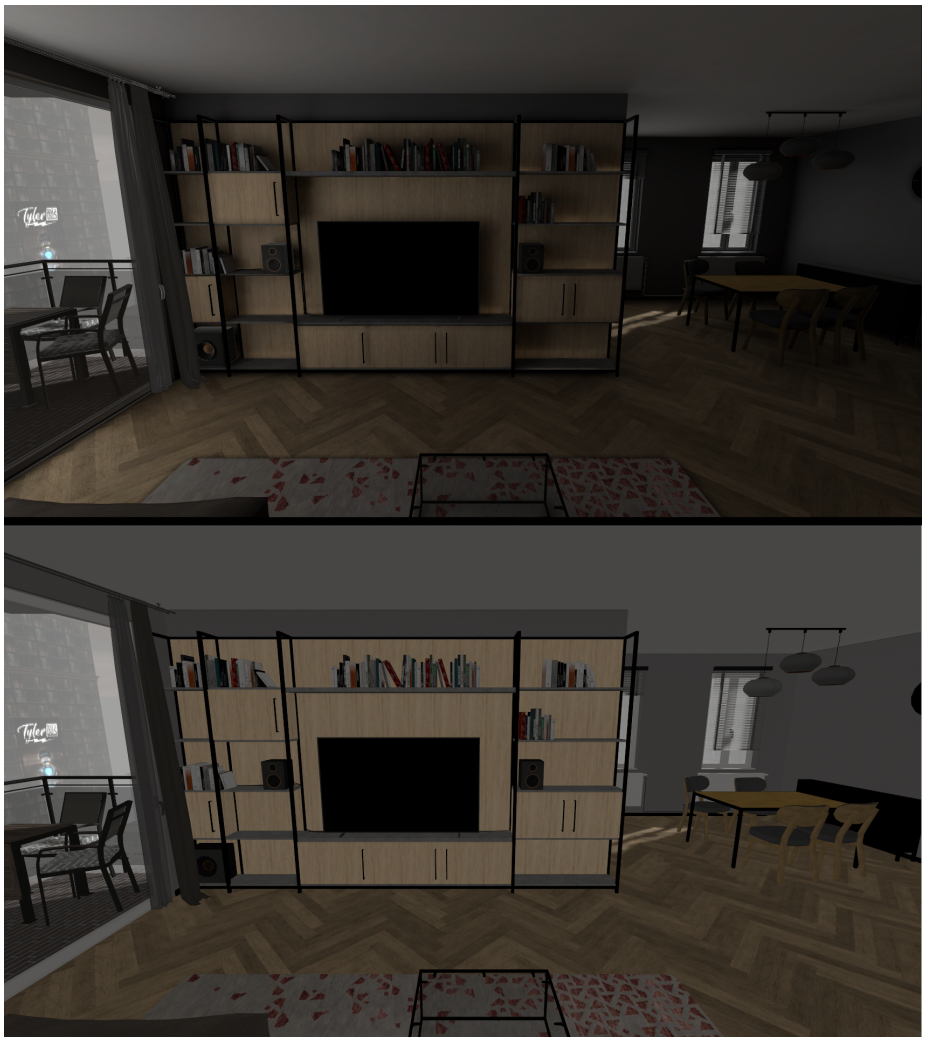

Figure 6.12: Comparison between enabled global illumination (upper image) and disabled global illumination (lower image), using the flat environment on a cloudy day. Area lights at the balcony, the windows and the set of shelves produce a more natural, realistic lighting scenario.

To increase realism, this thesis aims for light baking, which precalculates indirect lighting, storing it into lightmaps. Figure 6.12 shows the difference that global illumination makes. Since Unity does not have a built-in system to allow for lightmap switching within one scene and it is necessary to allow users to change light settings, a need to find a third party solution or create a new solution arose. Code from the "lightmap-switching-tool" from Laurent Harduin was modified to fit this thesis' project needs. Using this tool, it is possible to create different lighting scenarios for one scene using light subscenes that are loaded in addition to the environment and contains all lights and lightmap data relevant for a single lighting scenario.

For rain simulation, this project uses its own particle systems using the Unity particle component. The thunderstorm is managed by a "ThunderstormController", which spawns GameObjects resembling lightnings and plays thunder sounds.

6.3.6 Other Systems

The base scene also encompasses various other systems that will be mentioned only briefly since they are not meant to be the center of attention of this project.

The UI handling is done by an "UIManager" object, which adds listeners to the various buttons and the input field of the user interface. A "PlaybackBarButton" object manages the playback bar when playing back captures. It changes the position of the rectangle representing the current position of the playback and listens for user changes to this rectangle.

The "MinogridLogger" logs events for debugging purposes. The "MinogridConfigurationManager" loads the user defined configuration stored in a JSON file from the "StreamingAssets" folder at application startup, as discussed in section 5.2.2.

7 Evaluation

7.1 Type of Evaluation

Finding the right type of evaluation for this thesis was a difficult process. There is no obvious or well-tested evaluation method for virtual reality frameworks, much less so for virtual reality frameworks for therapy and research applications. Additionally, the presented software suite contains many aspects offering themselves to evaluation.

To reiterate, the goal defined in section 1.2 for the developed VR framework was the creation of a system that:

a. [...] is usable by users without technical background

b. [...] allows users with advanced technical knowledge to capitalize on their abilities

c. [...] generates applications which use state of the art technology

d. [...] is compatible with a wide variety of VR headsets

e. [...] has mechanisms for data capturing and evaluation

To evaluate goal a, it was decided to conduct a usability study, which is described in sections 7.2 and 7.3. Since Unity was used, goal b and d are met. Users with advanced technical knowledge can tune nearly every variable of an experiment or create new environments using the functions provided for the second screen, data capturing and the other features described in previous sections. Additionally, thanks to Unity and the OpenVR Unity XR Plugin from Valve, the framework is compatible with every major VR device, including the HTC Vive, HTC Vive Cosmos, Oculus Rift, Oculus Rift S, Oculus Quest (Link), Windows Mixed Reality, and Valve Index. The implemented framework also allows to capture VR sessions, satisfying the requirements of goal e.

While evaluating the resulting VR application would be very interesting, especially regarding the fulfillment of goal c, and yield valuable data, unfortunately the COVID-19 pandemic made this a very difficult and potentially dangerous task to undertake. While there was an attempt to create custom systems to use Android smartphones as VR headset so that every user could use their own smartphone as safe VR headset, the resulting experience was altered too much to allow for valuable conclusions. The main reason for this was the reduced visual fidelity due to the lower resolutions of smartphones as well as the slower and less exact head tracking and non-existing hand tracking.

However, this goal can be considered met, as this project uses the URP rendering pipeline of the newest Unity version (see section 6.2.1) as well as a number

of post processing effects, high resolution 3D models, textures and audio, while still maintaining a high frames per seconds[8] counter.

7.2 Experiment Setup

7.2.1 Tasks and Questionnaire

In order to evaluate if the framework is usable by people without technical background, it was decided to conduct a usability study. To create a virtual reality application with Minogrid, one has to follow these steps:

1. Create a web application account (if not already registered)
2. Create a new project inside the web application
3. Create a number of scenes for the project in question inside the web application
4. Download the project
5. Install Unity Hub for Unity Engine management (if not already installed)
6. Install Unity Engine with Unity Hub (if not already installed)
7. Install Steam [46] for VR headset usage (if not already installed)
8. Extract the downloaded project
9. Import the extracted project
10. Build a VR application

For evaluation purposes all steps were split into two main tasks, namely:

1. Create, configure and download a project (Step 1-4)
2. Local environment setup, project import and build process (Step 5-10)

After each task an After-Scenario Questionnaire (ASQ) [25] was used to assess the usability of each task. Subsequent to the completion of all tasks, a Post-Study System Usability Questionnaire (PSSUQ) [25] was used to assess the usability of the complete framework. Both questionnaires aim to provide generalizability of results and allow wide applicability.

The ASQ consists of three, the PSSUQ out of 19 different questions. Users can choose a value between one (strongly agree) and seven (strongly disagree) for every question. The questions were purposefully designed to offer some room for interpretation; two example questions are: "Overall, I am satisfied with how easy it is to use this system." and "I feel comfortable using this system.".

[8]The number of images the graphical processing unit renders per second. Especially important in virtual reality scenarios to help prevent motion sickness [50][3]

7.2.2 Procedure and Participants

Due to the COVID-19 pandemic, there was a need to develop an evaluation procedure that could be carried out without physical attendance. While under normal circumstances it would be best to conduct the evaluation in a lab in order to be able to talk directly to the participants and watch them interact with the system, it was decided to import the chosen questionnaire into Google Docs Forms [13].

Using Google Docs Forms, it is possible to define multiple different sections, which can contain text, media and a number of different question types. The finished questionnaire consists of 9 sections:

1. An introductory text
2. General questions about the participant (age, level of computer literacy etc.)
3. Description of task 1
4. ASQ for task 1
5. Description of task 2
6. ASQ for task 2
7. Information about the PSSUQ
8. PSSUQ for the whole system
9. Thanks and sendoff

Participants were asked to visit the web page containing the form and follow the instructions there. Depending on their internet connection, CPU, computer knowledge and previously installed software the whole process took participants on average between one and two hours.

Eight subjects participated in the evaluation process, which satisfies the general rule of '10±2' participants for general usability evaluations [18]. Four of the participants use a computer regularly but had no programming background. The remaining four possess programming knowledge, and two of these also have experience in developing virtual reality applications.

7.3 Results

The results of the evaluation are shown in table 7.1.

Metric	Score (Average)	Standard Deviation
ASQ 1	1.50	0.71
ASQ 2	1.83	1.18
OVERALL	1.58	0.71
SYSUSE	1.59	0.76
INFOQUAL	1.64	0.61
INTERQUAL	1.42	0.70

Table 7.1: The results of the usability evaluation.

As discussed in "IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use" [25] the OVERALL score rates overall satisfaction, the SYSUSE score system usefulness, the INFOQUAL score information quality and the INTERQUAL score interface quality[9]. These scores represent the average of a group of questions. "ASQ 1" and "ASQ 2" are the averages for the first and the second ASQs. As discussed earlier, the possible answers range from one to seven, with a one representing strong agreement and a seven strong disagreement.

The obtained results indicate that the developed system is useful (1.59) and satisfactory (1.58) in addition to possessing a high information (1.64) as well as interface quality (1.42). Participants stated that the system was very easy to understand and use; their comments provided valuable feedback regarding the layout of some of the web application's pages.

One of the main concerns of the participants was that some of the available help functionalities were not visible enough. A separate introduction to the application, its purpose and all possible functions after registration was one of the top requested features.

The low standard deviations and figure 7.1 show that there was only little scattering of the participants' answers. As indicated by some of the comments from participants, the information quality is one of the main areas that would benefit from improvements. The INFOQUAL score confirms this conjecture, as it is rather high compared to the other scores. The high ASQ 2 average implies a need to improve the usability of the steps of task 2. The main point of criticism during the evaluation concerning this task was the time it took to complete. One of the reasons for this was the slow upload speed of the server used for the evaluation, which resulted in slow project download speeds for participants.

[9]The interface in this context includes the items that are used to interact with the system, for example the keyboard, the mouse and the screen (including text and graphics).

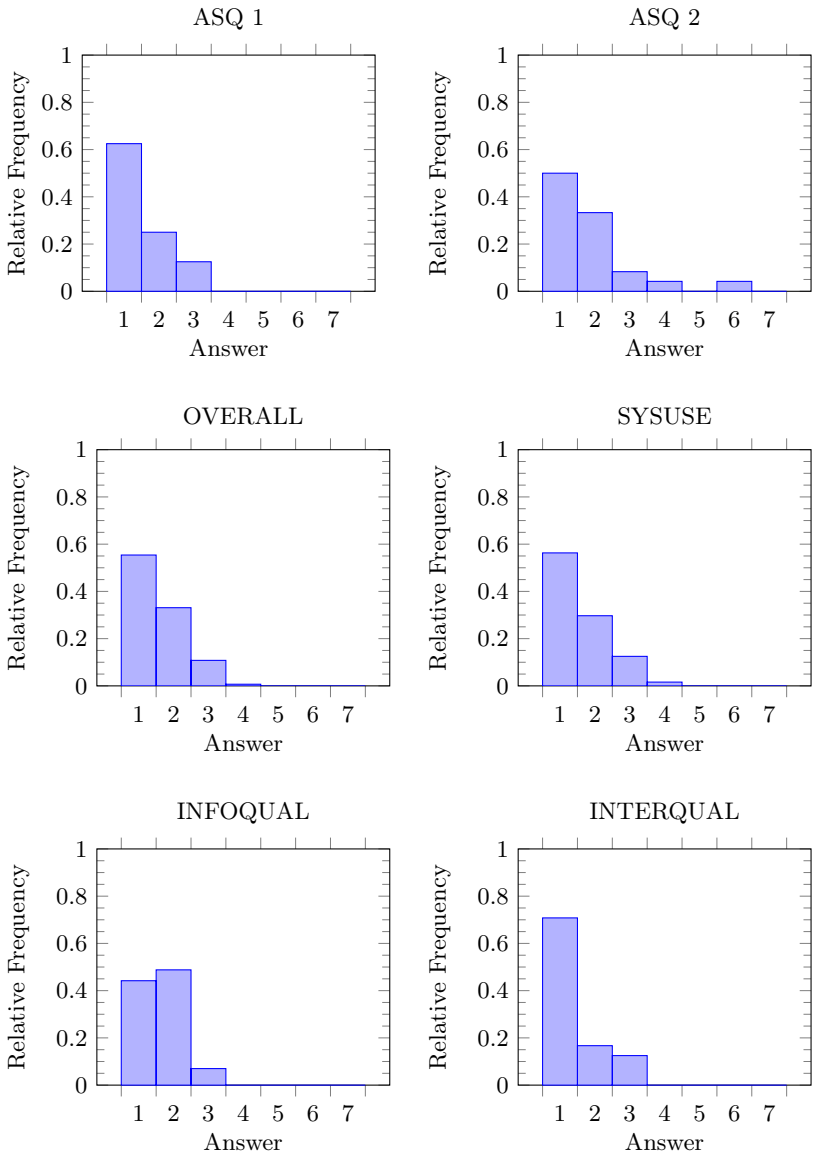

Figure 7.1: Histograms showing the distribution of the participants' answers.

8 Conclusion

This thesis presented a system designed to make it possible for users without advanced computer and programming literacy to create their own virtual reality experiments and therapy applications. A number of targets for this newly created system to meet were defined. Existing systems were presented, analyzed and categorized; the results of this analysis were used to refine the approach and methodology of the thesis' project.

The system this thesis presented is made up of three different components: One component to define the scenes, one component that communicates with a third party, state-of-the art engine and one component to process and render the output of the previous components to create actual virtual reality scenes in accordance with the options set via the first component.

The resulting implementation is called "Minogrid" and consists of three components: A web server, a Unity plugin and predefined Unity scenes. The design decisions and software architecture for each part of this thesis was described and a server on top of the Express.js server framework using TypeScript as language was built.

When using Minogrid, scene definitions for a project are created by answering a number of simple questions with predefined possible answers. The presented plugin provides users with the means to create running standalone applications using the project created during the first step.

A specially developed Unity base scene that contains nearly all the application's logic is loaded concurrently to every scene. Thus, it is easy to create new scenes that take advantage of the features Minogrid provides. These features encompass state of the art rendering techniques, resulting in realistic graphics, high frame rates, compatibility to all major VR headsets, head and hand tracking, a second screen UI which allows a second user without VR headset to monitor the person in VR and control the session (including giving them the ability to pause the session and switch scenes), a weather and lighting system allowing for multiple different light settings, including baked lighting, for each scene and data capturing. All captured data is stored and can be played back using the plugin.

This way, all the goals that had previously been defined where reached. One of these goals was further evaluated by conducting a usability study with eight participants using ASQ and PSSUQ. Due to the COVID-19 pandemic, an evaluation procedure that allowed participants to evaluate Minogrid at home without physical attendance had to be created. The evaluation consisted of two different tasks, with one ASQ after each task and one PSSUQ after both tasks.

8.1 Challenges and Lessons Learned

The main challenge of this project was to remove the need for the user to deal with the complexity of diverse programming and other background tasks when it comes to creating a VR experience. While the presented framework still retains some complexity, consisting of multiple different components working together, the user is now presented with an easy to understand, step-by-step process that they can simply follow along with even if they do not possess deeper knowledge of what happens behind the scenes.

Of course, it was also a challenge to work on all three systems concurrently. Altogether seven different programming/scripting/data definition languages were used, namely: C#, TypeScript, JavaScript, HTML, CSS, JSON and BSON. Frequently switching between these seven languages and the three different architectures proved to be rather complicated but had to be done in order to ensure that all components worked together seamlessly.

One of the main problems that occurred during development was the selection of the rendering pipeline best suited to the given purpose. At the start of this project, the HDRP was used to create the first scene. However, the first play tests quickly revealed that the performance was nowhere near the targeted goal and that the resulting scene as rendered was unusable with VR. After initial attempts at adjusting rendering settings and changing the scene, use of the HDRP was scraped altogether and a new project using the URP was created. This move payed off, since results gained with the URP look very convincing when post processing and global illumination is used while still being very performant.

Another big problem was the evaluation. While initial plans were to conduct an evaluation in a lab, during the development of this project it became clear that this would not be feasible. While it would have been desirable to evaluate not only the framework but also the resulting VR application, this unfortunately proved improbable in the end due to the ongoing global pandemic. Instead, an evaluation that can be done without oversight at home was created, which offered its own challenges.

8.2 Future Work

One can easily imagine many possible future extensions to the framework presented in this thesis. One that immediately springs to mind is the inclusion of additional external sensors that monitor the participant's vital signs in real time. This feature could also be added to the data capturing mechanism, allowing therapists and researchers to analyze the data in a greater depth.

A more encompassing but also very useful extension would be the inclusion of a visual scripting system to allow people without programming knowledge to

add logic to scenes. This would allow for completely new experiences, differing substantially from the currently possible setups.

Additionally, it would be possible to add certain experiment types meant for research only. These could contain special options only available for these research scenarios. One example would be an experiment type "Spatial Navigation Research" which allows for automatic environment generation and which not only alters existing environments, but creates completely new environments. This generation process could then be tuned via parameters available through the web application.

One more option would be the addition of a voice synthesizer and microphone input. This could allow for communication between the person in the VR and any observers on the outside, without the need to pause or mute the current session.

Another area that could benefit from additional features is the Unity editor itself. One could imagine a simplified 3D editor, provided by the Minogrid plugin, that allows users who have not previously worked with 3D editing software to alter the environment. This plugin could also connect to the web server and load additional assets on demand, allowing the user to add new assets befitting the current environment.

It is to be expected that virtual reality use will become more prominent in the future, leading to more technical advancements in this area. Thus, VR will bring even more advantages to psychology research and therapy applications and it will be very interesting to see how advancements in this area will be integrated into future research.

References

[1] 3DVIA. Virtools. URL: https://www.3ds.com/fileadmin/PRODUCTS/3DVIA/3DVIAVirtools/demoshowcase/html/index.html.

[2] Adam Baldwin. Helmet, version 3.23.3. URL: https://helmetjs.github.io/.

[3] G. Ae Ryu and K.-H. Yoo. *Key factors for reducing motion sickness in 360° virtual reality scene: extended abstract.* In *The 25th International Conference on 3D Web Technology*. Association for Computing Machinery, New York, NY, USA, 2020. ISBN: 9781450381697. URL: https://doi-org.uaccess.univie.ac.at/10.1145/3424616.3424723.

[4] M. Annett and W. Bischof. Vr for everybody: the snap framework, Jan. 2009.

[5] M. K. Annett and W. F. Bischof. Investigating the application of virtual reality systems to psychology and cognitive neuroscience research. *Presence: Teleoperators and Virtual Environments*, 19(2):131–141, 2010. DOI: 10.1162/pres.19.2.131.

[6] H. Ayaz, S. Allen, S. Platek and B. Onaral. Maze suite 1.0: a complete set of tools to prepare, present, and analyze navigational and spatial cognitive neuroscience experiments. *Behavior research methods*, 40:353–9, Mar. 2008. DOI: 10.3758/BRM.40.1.353.

[7] Blender Foundation. Blender, version 2.91. URL: https://www.blender.org/.

[8] S. Bouchard, P. Renaud, G. Robillard and J. St-Jacques. Applications of virtual reality in clinical psychology: illustrations with the treatment of anxiety disorders. In *IEEE International Workshop HAVE Haptic Virtual Environments and Their*, pages 7–11, 2002. DOI: 10.1109/HAVE.2002.1106906.

[9] Chris Talkington. Archiver, version 5.2.0. URL: https://www.archiverjs.com/.

[10] P. Cipresso, I. A. C. Giglioli, M. A. Raya and G. Riva. The past, present, and future of virtual and augmented reality research: a network and cluster analysis of the literature. *Frontiers in Psychology*, 9:2086, 2018. ISSN: 1664-1078. DOI: 10.3389/fpsyg.2018.02086. URL: https://www.frontiersin.org/article/10.3389/fpsyg.2018.02086.

[11] L. Donatiello, L. Gasparini and G. Marfia. Laying the path to consumer-level immersive simulation environments. In *2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pages 1–4, 2020. DOI: 10.1109/DS-RT50469.2020.9213691.

[12] Douglas Wilson. Express-session, version 1.17.1. URL: https://github.com/expressjs/session#readme.

[13] Google LLC. Google docs. URL: https://www.google.com/docs/about/.

[14] J. Grübel, R. Weibel, M. H. Jiang, C. Hölscher, D. A. Hackman and V. R. Schinazi. Eve: a framework for experiments in virtual environments. In T. Barkowsky, H. Burte, C. Hölscher and H. Schultheis, editors, *Spatial Cognition X*, pages 159–176, Cham. Springer International Publishing, 2017.

[15] R. Hassani and Y. E. B. E. Idrissi. Communication and software project management in the era of digital transformation. In *Proceedings of the International Conference on Geoinformatics and Data Analysis*, ICGDA '18, pages 22–26, Prague, Czech Republic. Association for Computing Machinery, 2018. ISBN: 9781450364454. DOI: 10.1145/3220228.3220254. URL: https://doi-org.uaccess.univie.ac.at/10.1145/3220228.3220254.

[16] D. Horváthová, V. Siládi and E. Lacková. Phobia treatment with the help of virtual reality. In *2015 IEEE 13th International Scientific Conference on Informatics*, pages 114–119, 2015. DOI: 10.1109/Informatics.2015.7377818.

[17] J. S. Hvass, O. Larsen, K. B. Vendelbo, N. C. Nilsson, R. Nordahl and S. Serafin. The effect of geometric realism on presence in a virtual reality game. In *2017 IEEE Virtual Reality (VR)*, pages 339–340, 2017. DOI: 10.1109/VR.2017.7892315.

[18] W. Hwang and G. Salvendy. Number of people required for usability evaluation: the 10±2 rule. *Commun. ACM*, 53(5):130–133, May 2010. ISSN: 0001-0782. DOI: 10.1145/1735223.1735255. URL: https://doi-org.uaccess.univie.ac.at/10.1145/1735223.1735255.

[19] L. Jadhav, K. Parekh, V. Gupta and S. Chandak. Using virtual reality for therapeutic treatment of phobia. In *2020 International Conference on Convergence to Digital World - Quo Vadis (ICCDW)*, pages 1–5, 2020. DOI: 10.1109/ICCDW45521.2020.9318727.

[20] Jeremy Thomas. Bulma, version 0.9.1. URL: https://bulma.io/.

[21] jQuery Team. Jquery, version 3.5.1. URL: https://jquery.com/.

[22] Junio Hamano. Git, version 2.30.0. URL: https://git-scm.com/.

[23] N. Khenak, J. Vézien, D. Thery and P. Bourdot. Spatial presence in real and remote immersive environments. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 1016–1017, 2019. DOI: 10.1109/VR.2019.8797801.

[24] J. J. LaViola Jr. Bringing vr and spatial 3d interaction to the masses through video games. *IEEE Computer Graphics and Applications*, 28(5):10–15, 2008. DOI: 10.1109/MCG.2008.92.

[25] J. R. Lewis. Ibm computer usability satisfaction questionnaires: psychometric evaluation and instructions for use. *International Journal of Human–Computer Interaction*, 7(1):57–78, 1995. DOI: 10.1080/10447319509526110. eprint: https://doi.org/10.1080/10447319509526110. URL: https://doi.org/10.1080/10447319509526110.

[26] E. M. Lionel, I. D. Gozali, A. N. Syakirah and W. Budiharto. Virtual reality as an alternative therapy for acrophobics. In *2020 International Conference on Information Management and Technology (ICIMTech)*, pages 365–369, 2020. DOI: 10.1109/ICIMTech50083.2020.9211189.

[27] Matthew Eernisse. Ejs, version 3.1.5. URL: https://ejs.co/.

[28] E. Mercier, S. Goldman and A. Booker. Collaborating to learn, learning to collaborate: finding the balance in a cross-disciplinary design course. In *Proceedings of the 7th International Conference on Learning Sciences*, ICLS '06, pages 467–473, Bloomington, Indiana. International Society of the Learning Sciences, 2006. ISBN: 0805861742.

[29] Microsoft. Microsoft onenote, version 16001.13328.20478.0. URL: https://www.microsoft.com/de-at/microsoft-365/onenote/digital-note-taking-app.

[30] Microsoft. Visual studio 2019, version 16.8.4. URL: https://visualstudio.microsoft.com/de/vs/.

[31] Microsoft. Visual studio code, version 1.52.1. URL: https://code.visualstudio.com/.

[32] MongoDB Inc. Mongodb, version 4.4.3. URL: https://www.mongodb.com/.

[33] MongoDB Inc. Mongodb compass, version 1.25.0. URL: https://www.mongodb.com/products/compass.

[34] Natalie Weizenbaum, Chris Eppstein. Sass, version 4.14.1. URL: https://sass-lang.com/.

[35] Nicola Del Gobbo. Bcrypt, version 4.0.1. URL: https://github.com/kelektiv/node.bcrypt.js#readme.

[36] Node.js Foundation. Node.js, version 14.2.0. URL: https://nodejs.org/.

[37] npm, Inc. Npm, version 6.14.4. URL: https://www.npmjs.com/.

[38] Oracle Corporation. Mysql. URL: https://www.mysql.com/.

[39] R Foundation. R. URL: https://www.r-project.org.

[40] Scott Motte. Dotenv, version 8.2.0. URL: https://github.com/motdotla/dotenv#readme.

[41] Strongloop. Express.js, version 4.17.1. URL: https://expressjs.com/.

[42] R. Tadeusiewicz. Computers in psychology and psychology in computer science. In *2010 International Conference on Computer Information Systems and Industrial Management Applications (CISIM)*, pages 34–38, 2010. DOI: 10.1109/CISIM.2010.5643696.

[43] TypeStack. Class-transformer, version 0.2.3. URL: https://github.com/typestack/class-transformer#readme.

[44] Unity Technologies. Unity, version 2020.2.2. URL: https://unity.com.

[45] Valeri Karpov, Kathryn Radovan. Mongoose, version 5.10.8. URL: https://mongoosejs.com/.

[46] Valve Corporation. Steam. URL: https://store.steampowered.com/about.

[47] M. Vasser, M. Kängsepp, M. Magomedkerimov, K. Kilvits, V. Stafinjak, T. Kivisik, R. Vicente and J. Aru. Vrex: an open-source toolbox for creating 3d virtual reality experiments. *BMC Psychology*, 5, Dec. 2017. DOI: 10.1186/s40359-017-0173-4.

[48] Webpack Team. Webpack, version 4.44.2. URL: https://webpack.js.org/.

[49] A. Williams. Reality check [consumer electronicsvirtual reality 3d]. *Engineering Technology*, 10(2):52–55, 2015. DOI: 10.1049/et.2015.0204.

[50] C. Zhang. Investigation on motion sickness in virtual reality environment from the perspective of user experience. In *2020 IEEE 3rd International Conference on Information Systems and Computer Aided Education (ICISCAE)*, pages 393–396, 2020. DOI: 10.1109/ICISCAE51034.2020.9236907.

Appendix A Questionnaires

A.1 After-Scenario Questionnaire

1. Overall, I am satisfied with the ease of completing this task

	1	2	3	4	5	6	7	
Strongly Agree	o	o	o	o	o	o	o	Strongly Disagree

2. Overall, I am satisfied with the amount of time it took to complete this task

	1	2	3	4	5	6	7	
Strongly Agree	o	o	o	o	o	o	o	Strongly Disagree

3. Overall, I am satisfied with the support information (online help, messages, documentation) when completing this task

	1	2	3	4	5	6	7	
Strongly Agree	o	o	o	o	o	o	o	Strongly Disagree

A.2 Post-Study Usability Questionnaire

1. Overall, I am satisfied with how easy it is to use this system

	1	2	3	4	5	6	7	
Strongly Agree	o	o	o	o	o	o	o	Strongly Disagree

2. It is simple to use this system

	1	2	3	4	5	6	7	
Strongly Agree	o	o	o	o	o	o	o	Strongly Disagree

3. I could effectively complete the tasks and scenarios using this system

	1	2	3	4	5	6	7	
Strongly Agree	o	o	o	o	o	o	o	Strongly Disagree

4. I was able to complete the tasks and scenarios quickly using this system

	1	2	3	4	5	6	7	
Strongly Agree	o	o	o	o	o	o	o	Strongly Disagree

5. I was able to efficiently complete the tasks and scenarios using this system

	1	2	3	4	5	6	7	
Strongly Agree	o	o	o	o	o	o	o	Strongly Disagree

6. I felt comfortable using this system

	1	2	3	4	5	6	7	
Strongly Agree	o	o	o	o	o	o	o	Strongly Disagree

7. It was easy to learn to use this system

	1	2	3	4	5	6	7	
Strongly Agree	o	o	o	o	o	o	o	Strongly Disagree

8. I believe I could become productive quickly using this system

	1	2	3	4	5	6	7	
Strongly Agree	o	o	o	o	o	o	o	Strongly Disagree

9. The system gave error messages that clearly told me how to fix problems (do not answer if you had no problem to fix)

	1	2	3	4	5	6	7	
Strongly Agree	o	o	o	o	o	o	o	Strongly Disagree

10. Whenever I made a mistake using the system, I could recover easily and quickly (do not answer if you made no mistake)

	1	2	3	4	5	6	7	
Strongly Agree	o	o	o	o	o	o	o	Strongly Disagree

11. The information (such as online help, on-screen messages and other documentation) provided with this system was clear

	1	2	3	4	5	6	7	
Strongly Agree	o	o	o	o	o	o	o	Strongly Disagree

12. It was easy to find the information I needed

	1	2	3	4	5	6	7	
Strongly Agree	o	o	o	o	o	o	o	Strongly Disagree

13. The information provided for the system was easy to understand

	1	2	3	4	5	6	7	
Strongly Agree	o	o	o	o	o	o	o	Strongly Disagree

14. The information was effective in helping me complete the tasks and scenarios

	1	2	3	4	5	6	7	
Strongly Agree	o	o	o	o	o	o	o	Strongly Disagree

15. The organization of information on the system screens was clear

	1	2	3	4	5	6	7	
Strongly Agree	o	o	o	o	o	o	o	Strongly Disagree

Note: The interface includes those items that you use to interact with the system. For example, some components of the interface are the keyboard, the mouse, the screens (including their use of graphics and language).

16. The interface of this system was pleasant

	1	2	3	4	5	6	7	
Strongly Agree	o	o	o	o	o	o	o	Strongly Disagree

17. I liked using the interface of this system

	1	2	3	4	5	6	7	
Strongly Agree	o	o	o	o	o	o	o	Strongly Disagree

18. This system has all the functions and capabilities I expect it to have

	1	2	3	4	5	6	7	
Strongly Agree	o	o	o	o	o	o	o	Strongly Disagree

19. Overall, I am satisfied with this system

	1	2	3	4	5	6	7	
Strongly Agree	o	o	o	o	o	o	o	Strongly Disagree

Other comments

A.3 Post-Study Usability Questionnaire Metrics

Score Name	Average of
OVERALL	Items 1 through 19
SYSUSE	Items 1 through 8
INFOQUAL	Items 9 through 15
INTERQUAL	Items 16 through 18