Politechnika Łódzka
Instytut Informatyki

PRACA DYPLOMOWA INŻYNIERSKA

Using the Unity game engine for aiding psychological research

Wydział Fizyki Technicznej, Informatyki i Matematyki Stosowanej
Promotor: dr inż. Dominik Szajerman
Dyplomant: Anna Aleksandra Dominiak
Nr albumu: 165389
Kierunek: Informatyka
Specjalność: Inżynieria Oprogramowania i Analiza Danych

Łódź, 2014

Instytut Informatyki
90-924 Łódź, ul. Wólczańska 215, budynek B9
tel. 042 631 27 97, 042 632 97 57, fax 042 630 34 14 email: office@ics.p.lodz.pl

The idea for the topic of this paper came from Hannes Högni Vilhjálmsson Ph.D., who was my supervisor in the CiSuUs project during my internship at CADIA, Reykjavik University.

1 Introduction

1.1 About

1.1.1 CiSuUs Project

With the inevitable growth of the capital of Iceland, new workplaces and housing must be provided for the companies and citizens. Rather than expand the city further out, plans have been made to increase the density in already populated areas and revitalize certain sectors of the city. However the authorities are concerned with what effect the changes will have on the people and are increasing effort to find the best solutions, making Reykjavik a friendly place to live and work in. The University of Reykjavik has started a project to explore new ways of testing the effect the architecture type has on people with the use of Virtual Reality. The Cities that sustain Us project aims to develop a methodology and complete lab for testing the restorative effect of environments, in order to determine what factors impact how people recover from stress and what is the most beneficial set of parameters.

The first part of the project includes setting up a Virtual Environment lab together with the Oculus Rift, controllers and other equipment, as well as developing software for creating and conducting future experiments. The first experiment shall be a reproduction of a previously done study, where the subjects will be exposed to an artificial urban environment through the Oculus headset. The next experiment shall build on the first and previous work of Páll Jakob Líndal [2], and will include a virtual version of an existing part of Reykjavik. The next phase is to compare the experience in the virtual environment with that of a real one. Further plans include creating virtual representations of future urban environments based on real locations, measuring their restorative potential and gathering results to present to the City.

This paper is concerned with the initial phases that is developing the software and reproducing a previous study. The team during this time consisted of 5 people, Páll Jakob Líndal Ph.D. and Kamilla Jóhannsdóttir Ph.D., psychologist, Hannes Högni Vilhjálmsson Ph.D. from the CADIA Institute, a 3D computer graphic artist - Sonja Bjork Ragnarsdóttir, and myself, Anna Dominiak, the Unity developer. Additional consulting in psychology was provided by Terry Hartig Ph.D., Professor of Environmental Psychology, (Institute for Housing and Urban Research and Department of Psychology, Uppsala University, Sweden).

The aim of replicating the earlier study is to validate the restorative value of specific urban design features. The study shall use two virtual neighbourhoods with differents features. The two neighbourhoods have previously been shown to be widely distinct in terms of judged restorative potential (one has a high potential, and the other low). The participants will be asked to take a virtual tour through one of the environments. To test the restorative potential, the procedure includes the initial phase of inducing fatigue on the subjects.

The features of the environment, the restorative impact of which is measured in this study include:

- building height (number of floors: 1 or 3),
- roof type (flat or steep),
- fasade complexity (with or without decorations),
- presence of greenery (grass patches on the pavement).

Some features, for example having greenery in the street, were proven to have higher restorative potential. Apart from the impact of individual features, the study tried to measure the impact of the entropy of the entire environment [1]. The entropy equals 0 when all structures in the environment are the same (for example all the houses are 3 floor, flat-roofed without decoration), and grows as the variety of the environment increases.

The software created in this project differs significantly from what was used in the previous study. Unlike before, the participants shall be using the Oculus Rift. What is more, the software allows for the dynamic creation of environments (rather than importing pre made models of entire streets) and planning of the test choreography.

1.2 The Aim

This paper aims to explore the possibilities of using the Unity game engine for aiding psychological research. It will examine how certain functionalities of the engine can be utilized for this purpose and will propose a system architecture developed with this type of research in mind. The practical part of the work was part of the “Cities that Sustain Us” project carried out at the University of Reykjavik under the auspices of the Reykjavik City, the aim of which was to develop software for conducting a series of tests on the restorative potential of urban environments.

1.3 Assumptions

The project shall focus experiments which are conducted on individual participants (rather than pairs or larger groups) and can be fully automated. What further distinguishes the tests that this thesis concerns is the virtual environment element. The paper concentrates on the use of Unity Pro for building PC standalone applications with the possibility of connecting to a remote server. Additionally, the Oculus Rift has been chosen to be used by the participants in order to experience the virtual environment.

1.4 Outline

1.4.1 Technology

Introduces the Unity game engine and Oculus Rift technology, presenting some their features, with particular emphasis on their usability for psychological research.

1.4.2 Computer games technology in psychology

This chapter takes a brief look at how computer games and engines have been used before in psychology. It also aims to specify what this kind of technology - together with virtual reality - can bring to the field and how it can help, listing both the advantages and disadvantages.

1.4.3 Design

This chapter goes over a typical experiment and proposes a way to structure it, characterising the different stages. It also tries to specify how an application aiding the creation and conducting of such tests should work.

1.4.4 Implementation

This chapter addresses the technical issues connected development in Unity and the implementation of the CiSuUs application. It contains code samples and refers to solutions specific for Unity and the Oculus Rift.

2 Technology

2.1 Unity

The Unity website defines it as an ecosystem for the creation of games and interactive content. It consists of a game engine and a set of tools and services that aid the development process.

Developed by Unity Technologies, Unity was first released in 2005 and as of September 2014 is at version 4.5, with a 5.0 announced and available for preorder. It is estimated to take over 45% of the full feature game engine market shares and has over 3.3 million registered developers.

Unity offers different licence plans making it accessible to both professional studios and indie developers. Available are: Unity (free), Unity Pro, iOS Pro, Android Pro and Team License. The Unity Pro version, giving access to more features, as of 2014 costs $1,500 or $75/month. What is important, is that upgrading to the Pro version can be done at any moment, without any loss. Therefore it is possible to start working on a game without any initial charges (but limited functionality). This is helpful especially for small teams and beginning developers, for whom the Unity was first created. Unity is a great alternative for larger engines and more expensive engines such as Unreal or the Cry Engine, not only because of the price, but also because it is friendly to new users. It was specially designed to be simple to start with, but at the same time offer powerful capabilities to the more experienced - easy to learn, hard to master. Unity offers the possibility to easily create build for different platforms: Windows, Mac, iOS, Linux, Web Player (browser plug-in), Android, BlackBerry, Windows Phone 8, PlayStation 3 and 4, PlayStation Vita, Xbox 360, Xbox One, Wii U, Tizen and others.

Unity consists of several tools. Some provided by the creators are an integral part of the product, others are paid extras. Unity also makes it possible for its users to create new extensions, tools and content for others to use. These can be easily bought and downloaded from the Asset Store - an integrated marketplace for Unity. Below is the list of some of the tools and features that build up Unity.

I. Provided, built-in tools:

A. Editor

It is the fundamental tool for developing in Unity. Connects together the different tools and features of the engine.

B. MonoDevelop

An IDE for scripting.

C. Mecanim

An animation system included in the engine . Offers the tools and workflows needed for creating and building muscle clips, blend trees, state machines and controllers directly in Unity.

D. Graphics engine

Unity provides standard renderers (forward, deferred), an extensive particle system, post-processing, realtime shadows, baked lightmaps and many pre-made shaders.

E. Physics engine

Unity uses a physics engine based on the solutions known from the nVidia Physix engine.

F. Audio

Allows importing standard files, and playing them at run time with various effects.

G. NavMesh

NavMesh baking is useful when scripting AI.

H. 2D

Unity offers a number of facilities for 2D development including a special 2D view, easy sprite management, and a two-dimensional physics engine.

I. Profiler and optimization

Unity comes with easy to use profiler tool which can be used to monitor the performance of the application. On top of that Unity brings us some helpful optimizing tools and algorithms such as Umbra Occlusion Culling, LOD and shader language optimizers.

II. Additional tools (examples):

A. NGui

An extension for graphical interfaces.

B. iTween

A tool for animating objects, useful for moving, rotating and scaling in time.

C. PlayMaker

A tool for visual scripting in Unity.

D. FingerGestures

A package that adds common mouse and touch screen input functionalities.

E. Shatter Toolkit

An addition allowing for shattering objects into pieces,

F. Mega-Fiers

A system for deforming meshes.

2.1.1 Why Unity?

There are many features of the Unity environment that can make it especially appealing to teams working on psychological tests. As it is not focused on any particular game genre, its flexibility can be used to adapt to the creation of non-game content. The possibility to extend the editor makes it easier to create new, specialized tools within the same environment. Since the team shall include psychologists (the majority of which probably have little or no experience with programming and software development), the visual editor is a valuable tool. It allows them to have an insight into the application, monitor the development process and even take part in creating and making modifications. Additionally, the Unity Asset Store makes it easy to obtain game-ready content to plug into the project. This minimises the amount of technical knowledge needed for developing a product. What is more, Unity has good support and an active community, making it easy to find tutorials and help. An additional advantage for psychological tests is the fact that the Oculus Rift is easily integrated into Unity, which gives numerous possibilities of using it to provide stimulus to the participants.

2.2 Oculus Rift

Figure 2.1
Oculus Rift Development Kit 1 (source: https://www.oculus.com/order/dk1/)

The Oculus Rift (figure 2.1) is a virtual reality headset. It is developed by Oculus VR, a company founded in 2012 by Palmer Luckey. After launching a Kickstarter campaign, the company raised over $2.2 million and released the first development version of the Oculus Rift. Development Kit 2 started shipment in July 2014. Oculus VR has support from major gaming companies such as Valve, Epic Games and Unity and has also been collaborating with Samsung to create the Samsung Gear VR for mobile. The aim of the Oculus project was to create immersive virtual reality technology that would be wearable and affordable, making it suitable for home use. The Oculus is not the only device of this type, Project Morpheus announced by Sony has similar goals.

2.2.1 Features

The Oculus offers low latency, 360° head tracking, allowing the users to look around the environment as one would in real life - by turning the head. By presenting unique parallel images for each eye, it creates a 3D, stereoscopic view that give the users a sense of depth.

2.2.2 The Oculus Rift in psychological tests

Virtual reality headsets such as the Oculus rift are currently probably the closest to a real world experience. Their advantage over testing in the real world is most visible in situations where the latter would prove to be dangerous or difficult and expensive to set up. Testing the reaction to not yet existing structure to improve their design, like in the CiSuUs project is a good example of an experience that would be otherwise difficult to achieve. This solution does however have its disadvantages. Due to the quality of the display, the weight of the headset and lack of tactile feedback, the VR cannot be treated as an equivalent to reality. These differences can have a major impact on the subject's experience and influence their responses and performance. What is more, wearing the headset practically makes it impossible to observe the subjects facial expressions, which could otherwise be used as a source of information on the subjects emotions. The probability of experiencing sickness when using the Oculus Rift additionally decrease its value as a testing tool, though several measures can be taken to minimise this side effect.

3 Computer games technology in psychology

3.1 Previous use of games and game engines

The first connections between computer games and psychology date back to the 1970s [7]. The combination of these two fields opens up wide range of possibilities for research, which can include the studies on the gaming experience (how players feel and react in games, what emotions they experience and how this can be used to improve games), instructional values and the impact of games on other aspects (for example the increase of spatial awareness), the differences between games and real life (for example issues connected with violence and morality), but there is also the possibility of using game related technology as a tool (to simulate real life scenarios). A subarea of the last, that is using game creation tools to replace other more traditional methods, and to test their usability is the concern of this thesis. Game engines have been used for research in psychology related fields before. For example, the Source Engine was used, among other studies, in research connected with behaviour during evacuations [5] (the author considers other engines, however Unity is not listed) and later examined in terms of automatically providing data output to the researcher [4]. S.T. Koenig [13] mentions Unity as the engine of choice for psychological research, using it in numerous studies, including work on rehabilitation after brain injuries [14]. It was also the tool used for creating the first study, which CiSuUs is based on.

3.2 Advantages and disadvantages of using game-like tasks and VR in psychology

3.2.1 Advantages

Repeatability - one of the problems with many psychological tests that are conducted is that they are difficult to replicate. If there are any doubts as to the

credibility of the results, or if someone wants to study the same topic on a different group of participants, the possibility of easily repeating the test is highly desired. Computer applications offer a way to quickly and effectively port to other machines. This would enable to conduct experiment in similar conditions independently of the location. (Of course, in cases where the Oculus is used, all institutions hosting the test would have to have the equipment, however by fulfilling that condition, they would open up a wide range of possibilities).

Simulating conditions difficult to achieve in the real world.

Strict control over the conditions - in real environments, many conditions are difficult to control (for example: the weather, the amount of people present in a particular place at a certain time, etc.). Using Virtual Reality lets the designers specify many of these parameters. This means that the conditions can be exactly as planned and remain the same for all participants.

3.2.2 Disadvantages

Complexity of stimuli. As game-like tasks will provide a large amount of complex stimuli, causing several reactions - emotional and other - it may be difficult (more than in sterile laboratory conditions), to isolate what was the cause of individual responses.

Costs and time of creation - the creation of entire virtual environments, and game-like tasks has often to this point required more resources than a simple paper and pencil tests or even smaller computerized tasks.

Retrieving data - although many larger games do have some system of tracking the users performance (such logs are used during beta testing of games or even after release in MMO to improve the experience and track down issues), the right data may still prove to be difficult to extract for the researcher.

3.3 Aiding psychologists in tests

To analyze how technology in general can help psychologists, it is good to distinguish 3 different stages of their work.

Planning and creating the test - here, the psychologists must decide on the methods they want to use, think of the different tasks the subjects will have to perform, the questions they will have to answer, the stimuli they will be exposed to and the order in which everything shall be executed. This is connected with a lot of research and at this stage, the ideas change often.

Performing the test - at this point the test is set. Here the subjects are asked to take the test and data connected with their performance is collected.

Analysing the results - this stage may start during the previous one, or after if finishes. Here the collected data is processed and analysed. The statistics can then lead to formulating some general rules.

This last stage in itself is a very complex topic. Numerous tools already exist that help with the processing of raw data, their visualization or finding patterns. This paper shall mainly focus on the first two stages as the area, where the Unity game engine can be most beneficial.

3.4 Psychological tests as opposed to games

Although the technology we are proposing to use for aiding psychological tests is the same as used for games, there are a number of differences between these two things that need to be considered in the development process.

3.4.1 Target group

Depending on the type and purpose of the test, the target group can vary. It might be men, women, children under 5, factory workers, surgeons, pensioners, prisoners or any other. What is important, that the people might have experience with computers, or even not use them at all. One cannot in such a case rely on the users being already accustomed to certain mechanics or patterns that would normally be obvious to gamers.

3.4.2 Time

While games usually are designed to be played for a longer period of time and often are replayable, allowing the users to get used to the mechanics and improve, psychological tests are in many cases performed only once for each participant and

take a relatively short amount of time. What is more, the pace is usually set, so the participant can't take his/her time to learn, go back and retry.

3.4.3 Purpose

Lastly, an important difference is the purpose. A game's main purpose is usually to provide entertainment. A psychological test on the other hand is to provide data for research and gaining knowledge. The results of such test are often later used for purposes that can have an effect on peoples health and lives. The consequences of mistakes and misunderstandings in such a case are therefore far greater than in case of regular games.

The conclusion is therefore, that in case of psychological tests, a great emphasis must be put on creating an intuitive and seamless experience. The users must at all times know exactly of what they have to do and how to do it. Instructions must be clear and the interface unambiguous. The design should be aesthetic and look professional to help the users feel safe, but be simple to avoid distractions. Clarity and functionality should never be in such a case sacrificed for the cost of artistic value or creativity.

Unless on purpose, the design should avoid exposing the user to stimuli that could have an emotional or other unpredictable effect on him/her, such as music, images or videos or even specific colour combinations. Such stimuli might have a strong and very individual effect on the user and could prove difficult in the analysis of the results.

3.5 Requirements for an application

3.5.1 Intuitiveness

As the users of the application might not be experienced with computers, it is crucial for it to be intuitive. The parts that shall be used by the psychologists to create tests can be more complex. As the test designers will spend some time with the application and are not under time constraints, it is acceptable for the product to require some sort of training in use. This is not however the case with the

participants, therefore a good interface design of the sections dedicated to them is an important part of the development.

3.5.2 Parametrization

As mentioned before, during the designing phase and initial pilot tests, many parameters, such as the length of the test, some of the questionnaires or the wording of the instructions may change. It is therefore very beneficial if the variable parts can be modified easily by the designer (rather than hard-coded into the scripts).

3.5.3 Language

When conducting psychological experiments, especially when gathering feedback in the form form questionnaires, it is important for the participants to understand exactly what is being said to them. Unless linguistic associations are in themselves the subject research, it is usually desirable for the participants to have a similar understanding of the concepts presented to them, so as to later be able to compare the answers. It is therefore advisable for the questions to be clear and if possible, in the native language of the participant. Using languages other than the native might put into question the findings of such research.

3.5.4 Safety

Psychological test are usually performed on a group of voluntary subjects. In many cases, the tests use tool or methods that have not been researched. These may have unexpected effects on the subjects posing a threat to their health. The dangerous effects might themselves be the subject of the tests, in that case they cannot be eliminated. Nevertheless, measures must be taken to minimise any unnecessary risks.

The participants should be informed of the possible threats and willingly agree to take the risk. In some cases, the exact knowledge of the possible effects could bias the participant and invalidate the results. In such situations, some details may need to be concealed from the participant for the time of the test. Such cases usually need an official approval of an ethics committee (depending on the prevailing laws and regulations).

Apart from standard safety issues that must be addressed when working with computers, the Oculus Rift presents additional requirements that should be fulfilled to ensure the safety of the subjects.

3.5.4.1 The test station

Since a person wearing the Oculus Rift cannot see his/her real surroundings, it is crucial to ensure that they cannot harm themselves. The station should ideally be kept far from stairs, open windows, walls and other obstacles one can walk into (unless they have a visual representation in the virtual environment), sharp objects, fire and other potentially dangerous objects such as hot pipes, bulbs or electric sockets.

As the Oculus itself is not wireless, if any other devices are used, such as headphones and controllers developers should consider using wireless versions, to avoid the user getting entangled in the cable cords or overwhelmed by their amount.

3.5.4.2 Simulator sickness

Oculus Rift users may experience a sickness similar to motion sickness caused by riding in a car or traveling by boat.

The Oculus Best Practise Guide [6] suggests steps that should be taken to test and minimise this effect:

- testing with a large number of unbiased users,
- implementing mechanisms that allow the user to adjust the intensity of the VR experience (for example: movement speed, field of view) and set to minimum by default,
- allowing user-adjustable parameters to be modified on-the-fly, without restarting,
- using an independent background such as a skybox which would match the user's real-world inertial reference frame,
- minimising the use of high spatial frequency imagery and instead using flatter textures.

4 Design

This chapter shall present the design and concepts of the Cissus application. It will give detailed descriptions of what was done to create an aid for psychologist in their test. The concepts here are generally independent from the Unity implementation and can be used in other environments. The proposed structure can serve as a template for other such projects. Technical details of the Unity implementation shall be addressed in the following chapter.

4.1 The application

Following the goal set out in the previous chapter, that is to create a tool that would aid in the first two stages of the test process, the application is divided into two parts - the Editor and the Executor. Although they are both part of the same program, the idea was to keep them independent of each other, so that working in the editor would not require entering the Executor and analogously, running the test in The Executor would not need communicating with the Editor. All the systems which are common for both, should ideally be independent of both and autonomous. In this way, should there be a need to separate the two into two autonomous programs, it could be done without having to redesign the entire solution. This could prove to be useful in creating an autonomous Executor in order to prevent the subject from entering the Editor mode and changing any settings.

4.1.1 The Editor

The Editor is that section of the application which aims to aid the first phase of the testing process, that is the creation stage. This is the part psychologists and other researches shall be working with. It serves two main purposes:

1. Planning of the tests structure. Deciding what parts the test shall consist of and what order they shall be executed in.

2. Setting the parameters. This includes how it is presented, when it stops and other things, specific for the test.

The editor has other functions, which include managing the language of the displayed texts and how the results shall be stored in the database.

All the specifications of the experiment are saved externally by the editor, and can be later on retrieved by the executor.

4.1.2 The Executor

The executor is responsible for carrying out the second phase of the testing process, that is the actual execution of the test with the subjects. It serves two purposes:

1. Carrying out the test according to the specifications.
2. Gathering data and saving it.

4.2 The Experiment

The test, (which in the application we shall call the experiment to avoid confusion with software tests), can consist of several parts, which in the project are called stages. Each stage has its specific purpose, as explained later. Stages are independent of each other and can be arranged in any order (although some configurations make more sense than others). One stage type can also appear multiple times throughout the experiment. There are 5 main stage types.

4.2.1 Information Stage

This stage has been created to provide information to the user, additional instructions or stimuli. In this stage, no input or action is required from the participant (except in some cases, confirming in order to continue). The information is presented in the form of text or images. A possible extension could additionally support audio recordings.

The information stage can consist of several pages. Each page is an image or text. The designer can modify the following parameters, which are common for all pages within a stage:

- the condition for continuing to the next page/stage. The designer can choose for the pages to change automaticly after a certain period of time(the time is adjustable), like in a slideshow, or require the user to provide input in order to proceed (or a mixture of both - the page changes after a user action, or in case of it's lack, after a set time limit),

- the background colour,

- the font colour and size (for texts),

- the text/image orientation/alignment (left, right or centered).

Figure 4.1 shows how some of these settings are modified in the CiSuUs Editor.

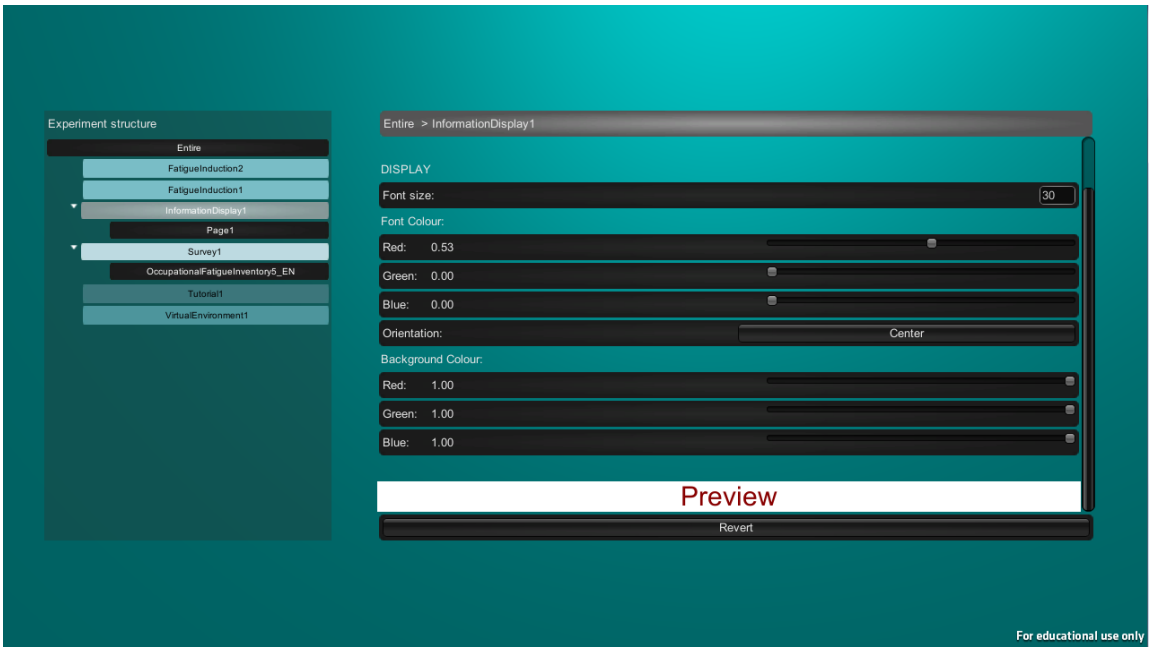

Figure 4.1
The Information Stage Editor in the CiSuUs application

4.2.2 Fatigue Induction / (Measurement) Stage

In this stage, the participants are asked to perform certain tasks. There are two main goals: to induce fatigue, that is tire the subject, and to measure the performance. The participants input during this stage is parsed and saved, to allow analysing their reactions. Some test will not require any fatigue induction and will use this stage solely for the second purpose.

There are several standard tests know in psychology used in such cases. The application implements two: SART (Sustained Attention to Response Task) [16] and Eriksen flanker variant test [16]. The test come with instructions, so they do not have to be additionally preceded by Information Stages.

4.2.2.1 SART

In this test, the subject must react to digits which appear on the screen. Only one digit, from 0 to 9, is displayed at a time. It stays on the screen for a certain amount of time and disappears. The user's task is to react to the appearance of a digit as quickly as possible, by pressing a button, except when the digit is a certain target digit (usually 3). In that case they should wait for the next digit. The instructions inform the subject what digit is the target and how they should react.

The data collected during the SART test are:

- the reaction time (how quickly after the appearance of a digit the user presses the button),
- accuracy (does the user react to the target digit).

Adjustable parameters:

- target digit (from 0 to 9),
- total amount of digits displayed,
- target fraction (amount of target/total amount of digits),
- time for which the digit is displayed,
- time between two digits (or a time range, if a random time is to be used).

The designer should also be able to adjust parameters connected with the visual display of the stimuli, that is:

- font size,
- font colour,
- background colour

as these may also affect the users performance. Additional parameters, such as the font type itself could also be considered.

4.2.2.2 Eriksen Test

In this test, the user is presented with sets of 3 letters, displayed in a row. Two letters from the alphabet and two colours are used in different combinations. The two outer letters are always identical (are the same letter and have the same colour), but the middle one can be different. The users should react based on the

middle letter, the two other ones serve only as distractors. Depending on the instructions, the user must press the left, right or no button when they see a letter of a specific type. For example they should press the right if the middle letter is an “H” and the left if the letter is a “S”. Another example is that the user must press the right button only if the middle letter is a red “H”, press the left button when it is a green “S”, and do nothing in all other cases. Each set of letters is preceded with a cue in the form of a word spelling “left”, “right”, “red” or “green” (or other colour names, depending on what colours are used). This cue is then followed by a blank screen, after which the letters appear. The letters are only visible for a certain amount of time, and disappear when the user reacts. They are then followed by a blank page, after which a new cue is displayed. Figure 4.2 shows 2 screens of the test with the default settings.

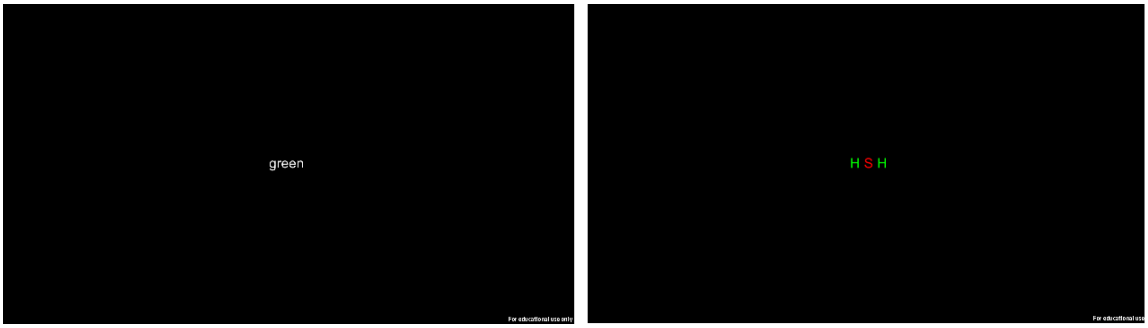

Figure 4.2
Screenshots from the Ericksen test

The user’s response to each set is registered (both the response type and the correctness). In case of a wrong answer, if the user changes their response (presses another button), their second answer is also noted (there is a set time after the first answer during which the second answer counts. During this time the set is no longer visible). In addition to user input and correctness, the displayed cue for each set and the distractor letters type (whether they were identical to the middle letter, had the same colour as it, were the same letter or were completely different) as also registered.

The adjustable parameters are:

- the two letters displayed,

- the two colours,
- the cue texts,
- the time of the entire test,
- the time for which the letters are visible,
- the time for which the cues are visible,
- time between the sets (or time frame, if it is to be randomised),
- time between the cues and the letters,
- time for which the user's corrected answer still counts.

The designer may choose, what reactions are considered correct for what type of letters. Additionally, as in the SART test, the display parameters such as the background colour and the colour of the cue texts should be adjustable.

These two examples show what type of tasks might need implementation and what parameters should be adjustable. There are numerous other tests that may be considered, each requiring its own implementation. The main features of this stage is to present stimuli, receive input from the user and record the run of the test.

4.2.3 Survey Stage

The purpose of this stage is to collect subjective opinions and statistical data from the participants in the form of surveys and questionnaires. It can be used to find out for example how the person would describe their condition, what they thought of the experiment or to note down the age and gender of each person. The participant, as in common surveys, might be asked to write a short answer in their own words, or to rate some aspect in a set scale, provide a number.

Since a participant may often be asked to fill out several different questionnaires in a row, a Survey Stage can be divided into smaller parts, which we shall call survey pages. Such a page consists of one questionnaire. Each questionnaire shall usually have it's own title and instructions. It is common to give the participant an indication of how far they are in through the stage. This can be done by displaying the amount of pages finished out of all the pages, or by a progress bar. The convention is for the number or bar to start at 0 and increase after a page is

finished. Figure 4.3 displays how a sample survey looks like in the CiSuUs application.

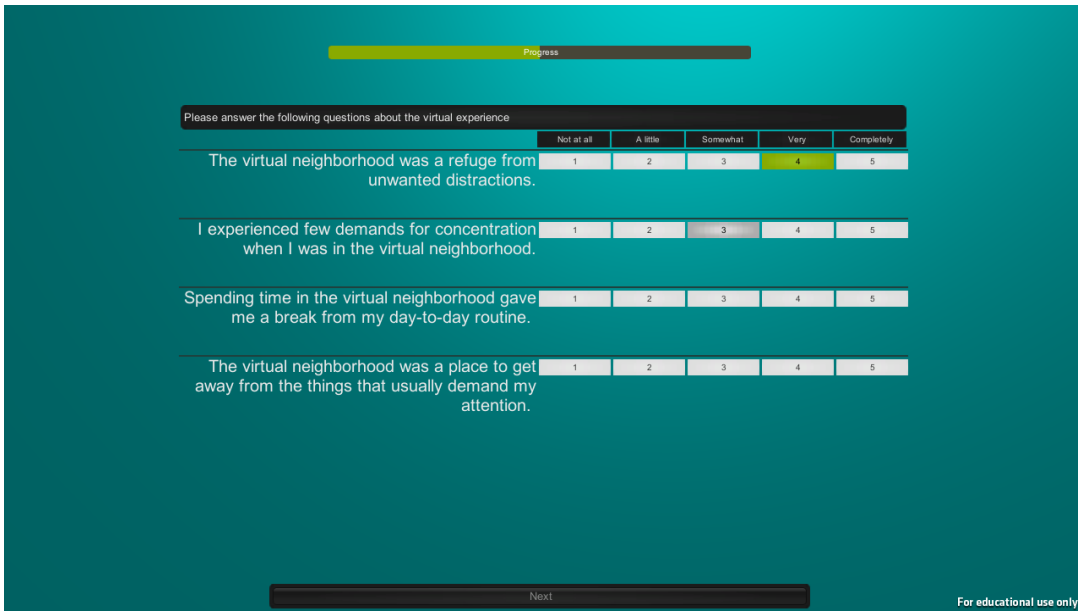

Figure 4.3
QuestionGrid survey as presented to the participant

Pages which are too long to fit on the screen can use a scroll bar or be further divided into smaller portions (screens). The first case allows the user to have all the questions in one place, however scrolling up and down big sections can be tiresome and the user may easily omit some part by accident when scrolling too fast. There is also the danger of an inexperienced user not noticing the scroll bar at all, and failing to find the further questions. The second approach splits otherwise connected questions into separate screens. It is therefore advisable to allow the user to move back and forth between screens freely within one page. If pages consist of many screens or questions, another progress indicator can be used to shows the amount of finished percentage of the page.

Survey questions come in different forms. Below are some of the most common.

4.2.3.1 Basic Fields

1. Short text questions, where the participant is expected to answer in one or few words.
2. Long text questions, where the answer requires a longer explanation,
3. Integer questions, where the answer is an integer number.
4. Float questions, where the answer is a real number, for example a fraction from 0 to 1.
5. Multiple choice - allows the user to choose any amount of answers from a given set (some restrictions may be added, like the minimum and maximum amount of chosen options).
6. One of many - usually implemented as a radio button or drop down list, allows the user to choose one option out of many. A subtype of these are Yes/No questions where the participant only has two options to choose from and must declare themselves either in the affirmative or negative.

4.2.3.2 QuestionGrid

This type requires the user to rate some statements in a set scale. The scale is common for all the questions in a grid, and numbers can have corresponding text descriptions. Figure 4.4 shows how a question grid might look like.

Answer the questions

	No	I don't know	Yes
Did you feel stressed during the test?	-1	0	1
Did you think the test was too long?	-1	0	1

Figure 4.4
Sample QuestionGrid survey

The designer can set the amount of questions and their content. The size of the scale and the number it starts from should also be adjustable, with a possibility to add labels to each scale point. The participants are required to answer all the questions, though the designers are free to create a “no answer”/”don’t know” option in the scale.

4.2.3.3 MultiGrid

This type, like the QuestionGrid, is based on a scale. The scale has the same size for all the questions, however, there are no common labels. Each question consists of two opposite statements. The participant must then choose where between those two they think they are.

The designer determines the number of questions, the scale size and inserts the statements. A sample MultiGrid is presented in figure 4.5

Core affect scale
Rate how you feel:

happy	-1	0	1	sad
calm	-1	0	1	nervous
awake	-1	0	1	sleepy

Figure 4.5
Sample MultiGrid survey

4.2.4 Virtual environment Stage

Although this element is not an obligatory part of all experiments, it shall be the key stage of those based on using virtual reality. In this stage the participant is placed in a virtual environment, in which they remain for a set amount of time, or until reaching some point. In the CiSuUs application, no data was collected during this stage nor was the users performance monitored in any way (the effects were measured at a later point), however it may in some cases be desirable to extend this stage in that way.

The setting, that is the content of the world, are determined by the topic of the psychological research. But the method of exploring the environment can also have a strong influence on how the participant experiences this stage. The designers might want to consider a few possibilities:

- **free walking**, where the user is free to move in any direction and stop at any moment,

- **fixed position**, where the user is placed in one spot, and can look round, but not move. This can be extended to multiple positions, where the user gets “teleported” between different locations,

- **following a track**, where the user is moved along a certain predefined path at a set speed,

- **walking along a path**, where the user may only move along a predefined path, but is free to stop at any given moment, and movement is applied only at the user’s direct input.

Figure 4.6 shows how the CiSuUs scene Editor allows for the setting of paths.

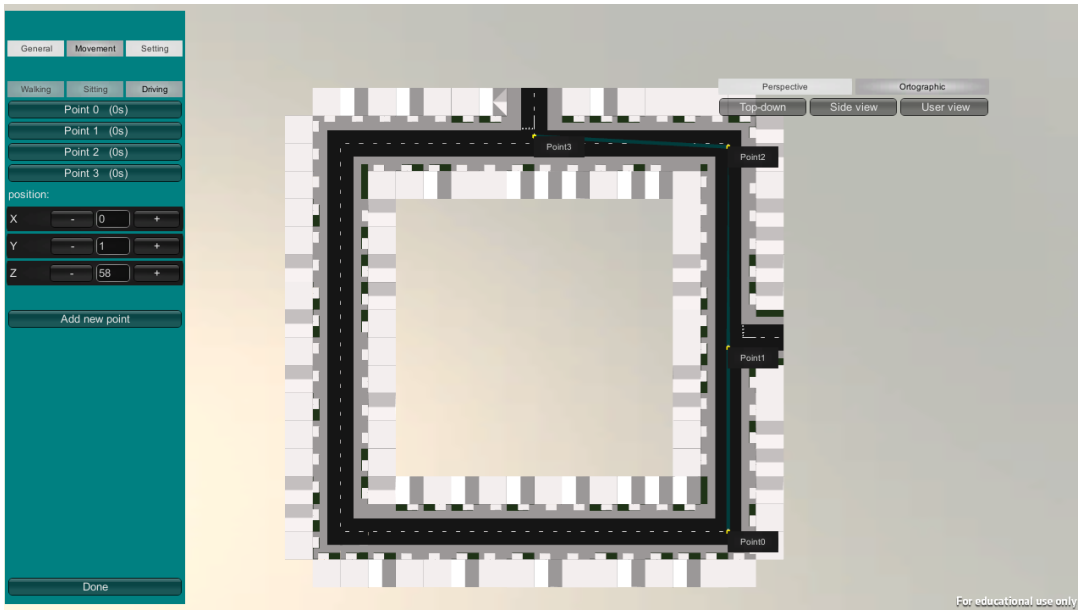

Figure 4.6
Movement settings in the scene editor of the CiSuUs application

Apart from the movement type, one should also decide on the ways of interacting with the environment. It is generally desirable for the experience to be

based on rules the users are used to in the real world, that is for objects to have collisions (rather than allowing for walking through walls), react to gravitation and so on. Apart from these standard psychic based interaction, some experiments may require more advanced actions, such as picking objects up or opening doors, which should be implemented according to needs.

While giving the users the possibility to move around on their own at their will may give the users the feeling of greater freedom, it might be overwhelming for those who are inexperienced with such technology (for example elderly people). Great care must be therefore taken to make the controls intuitive and the whole experience suited to the target users.

The editor for this stage might consider creating such adjustable parameters:

- the movement type,
- movement speed,
- time inside the environment (or other stop condition).

4.2.4.1 Scene

The scene can either have its own editing tool within the Editor, for making adjustments and setting objects (as in the CiSuUs application), or be created with a different tool, for example the Unity Editor, and then loaded in.

The purpose of the CisSuUs scene editor was to easily modify the features of an environment. Each element with variable features - in this case the buildings and pavement - should have the option of changing its settings. For example, the designer should be able to pick a house and change it from a 3floor building to a 2floor one. The editor should also have the possibility of modifying weather conditions, lighting, shadows etc. (figure 4.7).

Figure 4.7
Environmental settings in the scene editor of the CiSuUs application

4.2.4.2 Distress Button

As this stage is connected with the use of the Oculus Rift, it is important for the participants to be able to stop the experiment at any moment if start feeling sick, in order to prevent them from getting worse and/or destroying the equipment. This is especially true if they are expected to stay in the environment for a long period of time. A “distress button” should be available for them to press at any given moment, stopping the simulation. This should be followed by a screen asking them to wait for assistance (handling the equipment themselves in such a condition could be dangerous). The person assisting the experiment should have a clear indication that the simulations was terminated by the participant. The simplest way would be a visible indicator on the screen (sound effect may not work, if the participant is wearing headphones and they are the only audio output). Other, more complex possibilities may consider the application sending an alert to another device which would be monitored by the assistant (this could be especially beneficial if the assistant were to remain in a different room or have more than one participant at a time).

4.2.5 Tutorial Stage

The tutorial stage is suggested as an additional stage before entering the virtual environment, to provide a possibility for the users to become familiar with the controls and equipment such as the Oculus Rift. This way, the user will feel more confident in the actual environment stage. This can be helpful especially if performance is measured during the virtual environment stage, as it can minimise the initial confusion and will give clearer readings (otherwise bad results could be caused by the inability to use the controls or shock from a first use of the headset).

The scene should be simple, and the tasks straight forward. The proposed scene consists of a room with a box. The user is asked perform some or all of the following tasks (depending on the movement mode chosen and Oculus Rift usage):

- to turn his/her head to face the box,
- to rotate the avatar to face the box,
- to make the avatar walk up to the box,
- to try out the ‘distress’ button.

The user should receive clear instructions for each task, and a confirmation after completing each one. During this stage, the instructions for the current task should be visible all the time until completion, or there should be a possibility of looking them up at any moment (in the second case the user should be instructed on how this can be done). See figure 4.8 for a view of the first task from the CiSuUs application.

Figure 4.8
Execution of the tutorial stage

4.3 Localization

As mentioned in chapter 3, it is highly desirable to use the native language of the users. When creating a localized version of some test, there are many places where language is used, that should be considered:

- instruction texts,
- surveys questions and answer options,
- navigation buttons and labels,
- tooltips and other GUI texts,
- texts on images,
- texts on textures used in the scenes,
- audio recordings with speech.

It is therefore understandable, that these elements must be replaceable.

To solve the problem of localization, one can take one of two approaches. One approach is to create separate experiments for each language. In this case a single experiment will be smaller, require less work for creating a single version and, if so desired,

there is a possibility to create an experiment where part of it is in one language and part in another. The other approach is for each experiment to have a complete definition of all the possible languages that can be chosen, and for the language to be chosen during the run. This solution has the advantage of adding a possibility to switch languages after the start.

Another design problem connecting with localization is where to place the option for replacing the texts, images and audio files. On one hand, some stages like the Information Stage, are closely connecting with the text and images. These texts are usually specific for that experiment, and are only used once. It therefore makes sense for them to be inserted and edited within the stage. On the other hand, some text, like the “next” button might be used multiple times during the experiment, even if they are only connected with one stage (though they do not have to be). In that case it makes little sense to have to set these values each time separately. Instead they can be set once for the entire experiment, with only a possibility to access these common settings from each place they are used in. Using a combination of these two may be the most intuitive for the user and is a good solution if there are plans to make only a few language versions or if the all languages are implemented at the same time. However, with the growing number of languages to change and add, it may prove tedious and error prone to have to change content in many different places in the editor. In such situations, it is better to have all exchangeable content together. This decreases the possibility of omitting some part during the translation process.

4.4 Saving Data

Saving data is important in two aspects: saving the results and feedback from the participants, and saving experiment settings and presets by the designers.

4.4.1 Presets

Some sets of settings might be used often in different experiments, or many times during one. In such cases, presets can be created and saved for further use. This includes:

- standard surveys. Some surveys are commonly used and well defined. Typing in the questions each time would be tedious and could lead to errors. A survey can be created once, and then saved as a “standard survey”. From that point, it

should be possible to insert it as a survey page with a single insert action, without the need to define the content a second time,

- language presets - any localization data (texts, images, audio), that is not specifically connected with one particular case, for example navigation labels, can be saved together as a language preset, and then loaded into any new experiment to speed up the translation process,

- test settings - if a designer decides on a set of parameters for a test (for example SART), that they believe works well or is suitable for some particular situation, they might want to save these settings for further use in other experiments,

- environment settings - presets defining a certain atmosphere/weather/time of day or year in the environment can be saved separately from the scene and used across different environments.

5 Unity implementation

This chapter shall focus on the practical implementation in Unity of the ideas presented previously.

5.1 Technology used

The technology used in this project consist of:

- Unity Pro version (Scripting in C#),
- Oculus Rift Development Kit 1,
- MySql database + php scripting.

5.2 Developing in Unity

5.2.1 The Editor

The Unity development comes with an editor, which allows to create and manage a large portion of the product visually. The interface is divided into views, which are connected with different functionalities.

5.2.1.1 Project view and importing assets

The project view shows the assets belonging to the project. The structure represents the actual file system hierarchy on the disk. Importing new assets, for example models or scripts requires only moving the appropriate files into the Assets folder of the Project. Unity, if possible, takes care of importing the models. In some cases, it makes use of outside programs such as 3DsMax for importing from their native format, if they are installed on the machine.

5.2.1.2 Scene View and Scenes

The scene view gives an insight into the in-game world. It allows placing and arranging object in the scene. The scene can be thought of as a representation of a level, a part of the game that is loaded and active at

a certain time. A project can consist of many scenes which appear as assets in the project view and can be switched between at run time.

5.2.1.3 Hierarchy and GameObjects

The hierarchy panel gives an overview of all the objects in the scene. Basically everything in the scene is considered a GameObject - cameras, lights, characters and pieces of the environment. GameObjects can have one parent and multiple child GameObjects. These dependencies are represented in a tree structure in the hierarchy panel.

5.2.1.4 Inspector and Components

Apart from having child objects, the GameObject itself consists of Components. These define the properties of the object and can be looked up and modified in the Inspector. All GameObjects have a transform component, which defines the position, rotation and scale. (the child's transform values are relative to the parent's). Other components include mesh filters and renderers (for objects which have meshes), colliders, light components (defining options like the colour and intensity) and animators. Scripts are also considered components and attached to GameObjects.

5.2.1.5 Game View and the Play option

The game view enables viewing the scene as it would appear to the player. Unity allows testing the game basically at any time during the development (except when there are unresolved errors in the scripts). The Play button allows to enter the Play mode and run the game inside the editor.

5.2.1.6 Toolbar

This small view contains controls used with other views, such as the Play/Pause/Step buttons and the transform modifications

5.2.1.7 Console

The console is used for displaying warnings and errors as well as debug logs and print statements.

The other views are: the Animation View, the Profiler, the Asset Server View, the Lightmapping View and the Occlusion Culling View.

5.2.2 Scripting

One of the crucial features of Unity is scripting. The Unity game engine supports scripting in 3 languages

- C#,
- JavaScript,
- Boo.

All code samples provided in this paper shall be in C#, however most tasks can be also accomplished in JavaScript. The Boo language, although supported and documented in the official Unity references, is rarely used by the community.

Scripts are attached to game objects in the Scene and cannot be used independently. They appear as components in the Inspector View, where their public variables can be set and modified. They may also be attached to objects during runtime like other components with the AddComponent<scriptname>(); method. It is important to note that typical scripts (derived from MonoBehaviour) should not be created with constructors (that part is handled by the engine).

5.2.3.2 Script Editor

MonoDevelop is the standard editor for Unity scripting. However, the scripts can be modified in Notepad or any other program. Saving them in the Assets folder of the Unity project will cause their compilation on returning to the Unity Editor.

5.2.3.3 MonoBehavior and writing scripts

MonoBehavior is the default base class for Unity scripts. It grants access to many basic functionalities. MonoBehavior classes have key functions which are called at specific times during runtime. These include:

- Update() - called every frame,
- Start() - called once before any Update is performed on the object (can be used for initialization),
- FixedUpdate() - called before each physics update,

- LateUpdate() - called after all Update, FixedUpdate and animation calculations,

- OnGUI() - to be used for displaying GUI elements on the screen (over other elements from the scene),

- collision functions such as OnCollisionEnter, OnCollisionStay, OnCollisionExit and OnTriggerEnter, OnTriggerStay, OnTriggerExit for objects with collider components,

- mouse functions such as OnMouseDown, OnMouseUp, OnMouseUpAsButton, OnMouseEnter, OnMouseExit and others.

5.2.3 Consistent data

In many cases, the course of the experiment must be modified depending on some characteristics of the subject, for example their age or gender. If this must happen more than once, it would be desirable to have the subject enter the necessary information only once, and then store it for looking up whenever needed. Although querying the database or checking in a file is an option, keeping the data in memory should prove much more efficient.

In Unity, we can define a class and have an instance of that class attached to a gameobject in the scene via a script component. It is therefore easy to create a user profile class for storing any data provided by the user. However, by default, gameobjects are destroyed when a new scene is loaded, and so all the data stored in it is lost. Quite often, the data shall be needed in more than one scene. There are two ways of keeping consistent data.

5.2.3.1 Destroy Prevention

An object can be omitted in the process of destroying the scene and loading a new one by passing it as a parameter to the DontDestroyOnLoad() method. In case of gameobjects and components, the whole transform hierarchy is affected, so child objects shall also not be destroyed. This can be called for example in the Start() or Awake() function of a script attached to the object as follows:

```
void Start() {DontDestroyOnLoad(transform.gameObject);}
```

The object can also be passed as an argument from another script.

When using this method, it is important to remember to destroy the object manually when it is no longer needed and to avoid accidentally creating multiple instances of the object. This could happen for example when the target object is part of a scene which gets reloaded multiple times during the runtime of the application. In such a case each time the scene is loaded, a new instance of the target object will be created, while the others will still remain in memory and in the scene until destroyed. An object can be destroyed by calling

Destroy(Object target), either from a script attached to the object itself or from another object. An optional parameter may be passed to the function to determine the time in seconds after which the destruction should occur.

5.2.3.2 Static members

The other way of storing data throughout different scenes is to make use of static members of classes. Static variables can be assigned from a script and are not lost even when the object is destroyed. It is therefore possible to initialize the variables with user input data at some stage of the experiment, and then access them at a later point from within a different object in a different scene. Static variables are bound to a class rather than a specific instance, so there remains a single copy of the data. This means that writing to such a variable will override it's previous value, even when we are accessing it from another object. This is different than in the first method, where one can choose to store multiple data objects for different users, and should be considered when designing the application.

5.2.4 Saving permanent data

The CiSuUs application uses the local computer to store settings and all information needed for performing the experiment, however the users responses and performance measures are sent to a database on a server. Unity, with certain restrictions, allows saving both to the local machine and sending data to a server.

As saving data, especially to a server, might take longer than a single frame should, the saving functions can be called as coroutines with MonoBehaviour's StartCoroutine.

5.2.4.1 Saving locally

This solution, for security reasons is not possible in the Web build. A Unity Web application does not have direct access to the file system. Otherwise, the developer can use standard .NET libraries. (System.IO, System.Runtime.Serialization.Formatters.Binary, System.Xml), which will allow for IO operations, XML and binary serialization.

5.2.4.2 Saving to a server

When sending data to a server, the WWW and WWWForm classes from the UnityEngine namespace can be used as shown:

```
public IEnumerator test( )
    {
            WWWForm form = new WWWForm();
            form.AddField("action","Test");
            form.AddField("database",databaseName);
            form.AddField("table","TestTable");
            form.AddField("index",1);

           string url="http://example.com/script.php";
            WWW w = new WWW(url,form);
            yield return w;
            if(w.error != null)
            {
                 print("error");
            }
            else
           {
                if(w.text == "TestOk")
                {
```

```
                print("test success");
            }
            else
            {
                print("test fail");
            }
        }
    }
```

Fields can be added to the form, containing strings or integer values. These fields can be then read on the server side in a php script.

5.2.5 Extending the Editor

The CiSuUs application includes an entire section for editing the experiment settings. This approach was chosen, as it will enable building a standalone product, so that the experiment can be modified without Unity. It also gives a lot of flexibility in how the settings are modified and the visual display. However, Unity also offers a way to extend its editor, giving more control and additional functionalities.

One of the basic modifications is creating a custom inspector. In order to do this, a new script should be created. All editor scripts must be placed in a folder named "Editor" somewhere in the project's Assets. Such a script should inherit from the Editor, have the script it is connected to defined and override the OnInspectorGUI function like so:

```
using UnityEngine;
using System.Collections;
using UnityEditor;

[CustomEditor(typeof(NameOfConnectedScript))]
public class ScriptEditor : Editor
{
    public override void OnInspectorGUI()
    {}
}
```

Inside the OnInspectorGUI the content of the inspector is defined. By default, public variables are visible in the inspector, however overriding this function invalidates that setting. To restore this functionality, the DrawDefaultInspector() function can be called. It is also possible to add other visible fields, helpboxes or buttons. Buttons can be used to call functions from the connected scripts. For example, the custom inspector can have the following function:

```
public override void OnInspectorGUI()
{
DrawDefaultInspector();
EditorGUILayout.HelpBox("This is a help box",
MessageType.Info);
ConnectedScript myScript = (ConnectedScript)target;
EditorGUILayout.LabelField("Text", myScript.SomeText);
        if(GUILayout.Button("Build Object"))
        {
            myScript.SomeFunction();
        }
}
```

where SomeText is a public property returning a string. SomeFunction can for example be used to instantiate objects in the scene, move them around or make other changes.

5.2.6 GUI

5.2.6.1 Fitting the screen

As mentioned in the survey descriptions, it is important to be able to handle cases when the content is too large for the screen. There are a few options to consider.

5.2.6.1.1 Resizing GUI elements

With the standard GUI approach, each element should be passed its position and size. This gives great control over the layout.

```
GUI.Label(Rect position, string text);
```

```
Rect(float left, float top, float width, float height);
```

The screen coordinates in Unity start with 0,0 in the upper left corner and increase to the bottom right.

for example

```
GUI.Label(new Rect(0,0,100,100),"Hello world");
```

will create a label with the text "Hello world" in the upper left corner.

As screens may have different sizes (the value of the bottom right coordinate depends on the size), Unity allows us to access display information via the Screen class static variables. This allows setting the size and position based on screen attributes.

for example

```
GUI.Label(new Rect(Screen.width/4, Screen.height/4, Screen.width/2, Screen.height/2),"Hello world");
```

will give a label the width of half the screen and height of half the screen, starting in ¼ of the screens width and height (to center the label in the screen, as the GUI elements coordinates also start in their upper left corner. The text may not appear centered, as by default, the alignment is to the left side of the label rectangle).

Setting all the sizes manually might not be an efficient way to design the interface, therefore Unity offers another option - the GUILayout class. GUILayout offers similar elements as the GUI class, but their creation does not require passing a position rectangle. Instead it automatically chooses the positioning of the elements to fit the given space. GUILayout elements can be declared inside an Area, to ensure they stay within some set boundaries.

```
GUILayout.BeginArea();
…
GUILayout.EndArea();
```

The sizes of the elements can optionally be set manually by passing GUILayoutOptions as parameters in the constructor.

For example :

```
GUILayout.Label("Hello world",
GUILayout.Width(100));
```

which will ensure that this specific label will have the width of 100. The other possible options are: Height, MinWidth, MaxWidth, MinHeight, MaxHeight, ExpandWidth, ExpandHeight, which accordingly to their names set the height, minimum and maximum widths and height, and declare whether or not to allow vertical and horizontal expansions.

These are powerful tools, however they only go so far. With the growing number of elements and small screen sizes, one cannot shrink the elements forever, as they will become unreadable. Defining too many elements will either cause them to be crowded, or if their set size exceeds the available space, they may get cut off. This calls for other solutions.

5.2.6.1.2 ScrollView

Both the GUI and GUILayout offer a ScrollView. This helpful element automaticly adds horizontal and vertical scroll bars if needed, and takes care of their handling.

```
public class ScrollExample: MonoBehaviour {
public Vector2 scrollPosition;
void OnGUI() {
      scrollPosition =
      GUILayout.BeginScrollView(scrollPosition);
      // All GUILayout elements to be included in the
      // scrollview go here
      GUILayout.EndScrollView();
}}
```

5.2.6.1.3 Calculating content size

When scrollviews are to be for any reason avoided, content can be planned out and split into parts that fit the page. This approach is more complex than the previous and requires additional calculation.

The contents size should be calculated, and when it exceeds the given space, the extras should be somehow stored for displaying later.

The storage method and way of proceeding to the next part shall depend on the content. The CiSuUs application uses this approach to split survey pages into screens. The sizes of the questions are calculated and based on that the questions are divided into lists that fit on a screen. Screens can be navigated between with buttons, and the application keeps track of the current screen index. For details on the implementation see the MultiGrid, QuestionGrid and FieldInput Executor files.

Unity has a couple of functions that allow this kind of operations.

1. GUIStyle.CalcSize(GUIContent content);

This function allows to calculate the size of an element without drawing it. It is based on the GUIStyle that shall be passed into the constructor. If no GUIStyle is passed, the default style is used. The default style can be used for the calculations as so:

```
GUIStyle.label.CalcSize("Hello world");
```

This method does not take into account word wrapping.

2. GUIStyle.CalcHeight (GUIContent content, float width);

If the width of an element is to be set, and word wrapping used, this function can be used to determine the height of an element. As with CalcSize, this is a GUIStyle function and should be used with the GUIStyle of the element to be drawn.

3. GUILayoutUtility.GetRect

This function reserves layout space for content and returns the Rect, that would be used for it's display.

4. GUILayoutUtility.GetLastRect

This returns the Rect of the last drawn GUILayout element.

Note: Use of this function may lead to unexpected behaviour. If the result is used for determining the size of an element in another call of the

OnGUI function it may cause flickering. This is because the OnGUI function is called more than once during a frame (it gets called once for every event) and has different modes. Not all values are set during certain modes. GetLastRect().width and GetLastRect().height may therefore return 1 during certain calls, independently of the elements actual size. This should be held in mind then making use of the GetLastRect function.

5.2.6.2 GUI Controls

5.2.6.2.1 Standard controls

The Unity GUI and GUILayout come with a set of standard controls, which can be used for creating the user interface.

1. Label

   This is an element for displaying non-interactive content, such as text or images.

2. Button

   This element is interactive and returns true when clicked. It only reacts to a click once.

```
if(GUILayout.Button("Click me"))
{
// do something
}
```

3. Repeat Button

   An interactive element that returns true as long as the mouse remains pressed down on it.

4. TextField

   A single line of text. The text is editable. This can be used as an input field. It takes a string as parameter and returns the (possibly modified) string back.

5. TextArea

   Similar to the text field, but multi-line.

6. Toggle

A checkbox-like element. Is either on or off, and return a boolean value based on it's state. Clicking it changes the state to the opposite.

7. Toolbar

   A row of buttons, where only one is selected at a time (returns the selected index).

8. SelectionGrid

   Similar to the toolbar, but can have multiple rows.

9. HorizontalSlider and VerticalSlider

   Sliding knobs that can be dragged to change their value between a predefined minimum and maximum.

10. HorizontalScrollbar and VerticalScrollbar

    Used for ScrollViews

11. Window

    Draggable container of Controls.

5.2.6.2.2 Progress bar

A progress bar is not included in the standard controls. It is however simple to implement using the Box elements. For example:

```
GUI.BeginGroup (new Rect(Screen.width/2-barSize.x/2, Screen.height/2 - barsize.y/2, barSize.x, barSize.y));
GUI.Box (new Rect(0, 0, barSize.x, barSize.y), barBack, barStyle);
GUI.BeginGroup (new Rect(0, 0, barSize.x*fractionDone, barSize.y));
GUI.Box (new Rect(0, 0, barSize.x, barSize.y), barFront, barStyle);
GUI.EndGroup();
GUI.EndGroup();
```

GUI.BeginGroup() and GUI.EndGroup() together define a region in which other GUI elements can be placed. The inner elements coordinates will

be relative to the group position, and if the element exceeds the boundaries, only the part inside will be visible. For this reason, only part of the second box will be visible (the fraction indicated by the value of fractionDone).

There is no simple way to fill a GUI.Box with a colour, other than assigning it a texture. The texture can either be passed in as GUIContent, or else can be the background texture of the GUIStyle. Textures can be imported images, assigned in the Unity inspector. However they can also be created manually. For a single colour background this can be done as follows (done once, for example in Start):

```
Color color = new Color(0,0,0);
Texture2D backgroundTexture = new Texture2D(1, 1);
backgroundTexture.SetPixel(0,0,color);
backgroundTexture.Apply();
```

This texture can then be passed on as the background of a GUIStyle.

```
GUI.skin.box.normal.background = backgroundTexture;
GUI.Box(position, GUIContent.none);
```

The texture shall be used to fill box as a background. In this case the default style is modified.

5.2.7 Time in Unity

To keep track of time, the Unity offers a Time class, which gives access to values such as timeSinceLevelLoad, timeScale and deltaTime.

deltaTime can be used in Updates which rely on time. It is the length of the previous frame multiplied by a timeScale factor. This factor, by default 1, can be changed, for example to 0 when pausing the game (though it is important to note, that this shall not stop the entire application and all Update functions shall be called normally. It will only affect operations dependant on the deltaTime, as it shall return 0).

5.2.8 Scene and resources

Creating a scene within the Unity editor is simple - assets or prefabs can be dragged from the project browser into the Scene view, causing them to appear in the scene. This is done before building the application and with assets that have already been imported into Unity.

Loading in assets during runtime can be done, but has its limitations. During building, for optimisation, Unity does not include assets that are not connected to any gameObject in the scene. To instantiate a certain object in the scene during runtime, it can either be stored as a prefab and attached to a script of a GameObject in the scene. The script then has access to the original, and can instantiate using `Instantiate(Object original, Vector3 position, Quaternion rotation)`

Objects that are not attached to any script or object, but are to be included in the build should be placed inside a folder called "Resources". They can then be loaded in using the Resources class

```
Resources.Load(string path, Type systemTypeInstance);
```

Another option that is available in the Pro version are AssetBundles. These can contain any assets that are recognisable by Unity. Such bundles can be built from assets in the scene, put in a server, downloaded and imported into an application during runtime.

What is important to note, that even dynamically loaded assets have to first be processed by Unity. There is not simple way to include raw assets such as models in .fbx or other formats into the application. This would include building (or buying) a specialized importer.

5.2.9 Differences between editor and build

Although Unity allows for the testing of the application inside the editor (with the "Play" button), the editor play mode differs from the final build and unless precautions are taken, the build may lack some functionalities or demonstrate unexpected behavior.

First of all, the “Play” button will run the currently open scene. However, in the build version, the scene that shall be loaded first, is the one with index 0. All scenes that are to appear in the build must be included in the list, otherwise they shall not be accessible (even if the wanted scene was open during build time).

Another difference is connected with file paths and resource management. As has already been mentioned, all unconnected assets outside of the Resources folders are omitted. The rest gets packed into “.assets” files. Therefore, if any resource was omitted, but the application tries to access it (usually via a relative path or an absolute path on another machine, where the resource is absent), it shall not be available. What is more, even if the resource was not omitted, and was packed into the assets package, it shall also no longer be available via system functions such as System.IO.File.ReadAllLines, since the build version does not preserve the folder structure of the resources as seen by the file system. They can however still be reached with their relative path via the Resources class as shown before.

Classes from the UnityEditor namespace also cannot be used in the build version (this shall in fact cause errors preventing successfully finishing building). While this is obvious when creating Editor extensions, it is important not to use these classes in other parts of the code, for example to list out the scenes included in the build list (UnityEditor.EditorBuildSettings.scenes).

5.3 Oculus Rift

The Oculus Rift is currently available for developers with the Development Kits. The SDK can be downloaded from the Oculus Developer website[11]. It comes with example applications, configuration tools and instructions. Unity integration is supported by default, and an integration package, demo and guide can be obtained from Oculus. The package is a standard .unitypackage which can be imported into Unity Pro (double-clicking should import the package straight into the currently open project), coming close to plug-and-play functionality. The package includes standard controller prefabs that can be dragged onto the scene without further modifications. The Guide explains in details the steps to be taken for the correct functioning of the Oculus Rift in Unity and preparations for a standalone build.

5.3.1 Configuring the Oculus Rift

Although the Oculus Rift has some default settings for parameters such as the field of view (FOV) and Inter-Pupillary Distance (IPD), these values in reality differ slightly for every person. Therefore, for the best effects, they should be measured and set for each user. The SDK offers a configuration tool which, through a test, determines the correct values and allows the creation of user profiles. A user can be made default and their setting shall be available to other Oculus applications.

5.4 The CiSuUs Application Unity architecture

As suggested in a previous chapter, the experiment itself is seen as a list of different stages. Each stage has its own C# class (not derived from MonoBehaviour) which holds its definition and settings for that stage. These stages are held in a list in the experiment object. On entering a mode - Editor or Executor, the application instantiates an ExperimentEditor or ExperimentExecutor object, which are Unity prefabs.

The ExperimentEditor consist of several scripts, which serve as editors for different stages, for example “TutorialEditor”, “SurveyEditor”, “MultiGridEditor”. These scripts are responsible for displaying controls which enable changing settings in the stages. Each editor has an Init function which takes an instance of that stage (or page) as a parameter. There is one instance of each editor in the ExperimentEditor, even if the experiment consists of multiple stages of the same type. In such a case the Init function is used to initialize the editor with each individual stage.

The ExperimentExecutor object has a similar structure. It consists of executors for different stages (although executors for pages are added and removed additionally at run time). The main ExperimentExecutor is initialized with the experiment, which is passed on as a parameter, and then for every stage in turn, enables and initializes the corresponding executor.

The experiment is stored in an xml file, in a human readable format, allowing it to be read and modified outside the application. Standard surveys, language presets,

environment setting presets and created scenes are also stored in separate xml files. Each stage and page, as well as the localization texts classes have ToXML and FromXML functions allowing them to be easily parsed.

5.4.1 Scene

The urban environment consists of different structures. In CiSuUs there are 9 structure types (these include the several building and pavement types such as regular and corner versions). Each structure type, has its unique name and a set of variable features and their possible values. These are defined in a xml file. The Resource folder contains a subfolder corresponding to each structure type. These subfolders contain the xml definition prefabs of all the possible variations of that structure type. For example, a regular pavement - HS - has one variable feature - the existence of grass, which in turn has two possible values - with grass and without. This gives us two variations of this structure. A regular house will have three variable features - decoration amount (see figure 5.1), floor count and roof type, which in turn have respectively three, two and two possible values. This gives us twelve (3*2*2) possible variations. The amount of variations can be counted by formula 5.1:

$$\prod_{i=1}^{n} v_i \tag{5.1}$$

where n is the amount of variable features and v_i is the amount of possible values for the i-th feature.

Figure 5.1
Decoration types: low (left), medium (center) and extra (right)

To simplify the arrangement of different buildings and pavement tiles, the CiSuUs scene editor loads in a layout created in an external 3D modeling program. Such a layout contains the positions and rotation of all the structures in the environment represented by nodes with names corresponding to the structure type. The names are used to fill the layout with the correct type. By default all nodes of the same type are filled with the same variation of that type, which can then be modified by the user. After clicking on a structure, the designer is presented with a list of variable features which they can modify. On changing a feature, the ResourceManager loads a new model corresponding to the new values and replaces it in place of the old.

5.4.1.1 Entropy counting

The entropy of the environment, that is the diversity is counted based on the amount of different variations of the structures. When all structures in each type are of the same variation, the entropy equals 0. All structures are held in a container, allowing the different types and variations to be counted. The entropy is counted according to formula 5.2 [1].

$$E = -\sum_{i=1}^{n} p_i \cdot log_2 p_i \tag{5.2}$$

where n is the amount of all the variations of all structure types together and p_i is the frequency of the i-th variation counted from formula 5.3.

$$p_i = \frac{j_i}{m} \tag{5.3}$$

where *j* is the amount of structures in the current environment of that variation and m is the amount of structure in the environment of the parent type of that variation. Figure 5.2 presents two environments with different entropy levels.

Figure 5.2
Entropy levels. High (left) and low (right) entropy environments

5.4.2 Database

As the experiments do not have a set structure (it is defined separately for each experiment by the designer), the corresponding database also cannot be created beforehand. The amount of tables and their fields results directly from the experiment definition, therefore it is best for the database to be created after the whole experiment is designed. The name of the database and the tables within can be changed so that the data is easier to analyse later on. By default the database name is the same as the experiment itself, and the table names come from stage/page names (with numbers). The application makes sure that all table names are unique. When new parts of the experiment are added which require tables,, they are given the default name, for example: SART1, SART2, SART3, Fascination_EN1 and so on. When the user tries to change the name, the new name is checked, to see if it is unique within the experiment. If not, the user is informed of the fact and asked to choose a different name. To check the uniqueness, a HashSet<string> is used, as this container stores unique values. The Add(string name) method of the HashSet returns a bool indicating if the new value was added, or not (if it was already in the set). [12]

Connection with the database is managed by one DataBaseConnection script. Data is passed as function parameters and then prepared for sending. In the process of creating the database, the name is sent to the server. Then, for each stage/page which requires a table (that is the fatigue induction stages and the surveys), a separate form is created, which is filled with the name of the table, and the names and types of each field. This data is sent to the server which then handles the creation of tables.

5.4.3 Oculus Rift

The Oculus Rift Unity integration package comes with a ready standard controller, however, to switch between the Oculus (during the environment and tutorial) and a normal display (in other stages), as well as decide whether or not to use the Oculus at all (the environment can also be explored with a traditional view), it is not enough to place the controller in the scene. When it is time for the user to enter one of the two stages where they control an avatar, a player controllers are instantiated. There are two controllers - a standard one and an Oculus one, but only one of them should ever be active at a given time. The application activates the Oculus controller if it detects the presence of the Oculus (See figure 5.3 for a display prepared for the Oculus - a separate view for each eye). This is done with the help of OVRDevice (this script should be added to one object in the scene).

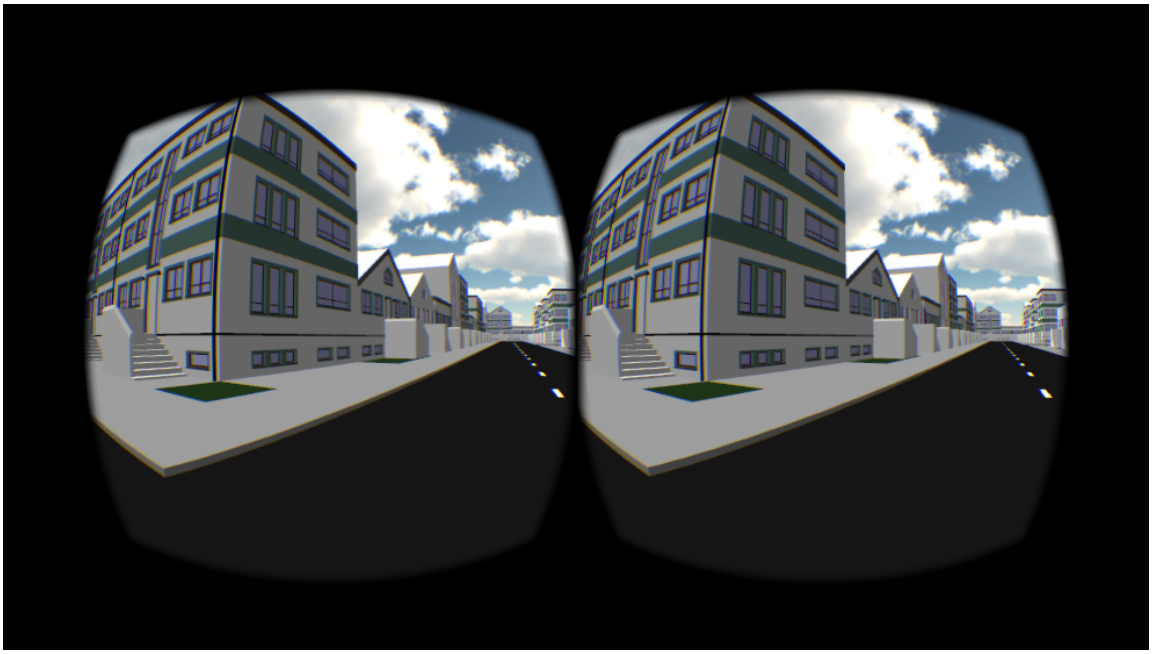

Figure 5.3
Display for the Oculus

The Oculus is both an input and output device. To detect if Oculus input is present, one can use:

```
OVRDevice.IsSensorPresent()
```

which returns a bool. This can be checked to choose the correct controller. The returned value will change if the Oculus USB input is removed or inserted, so it can be used to adapt and switch the controller. If this method is called in the Start function before the state of the Oculus is actually checked, it might return incorrect information.

```
OVRDevice.IsHMDPresent()
```

should return whether the Oculus HMD output is connected. However, this method has proven to be unreliable, returning true even after the output was disconnected from the running machine.

6 Summary and Conclusions

It was possible to create a program that could aid the process of designing and running psychological test. The final product is a standalone application which fulfills the main goals set for it, allowing for:

- designing the order of stages
- adjusting of parameters
- designing virtual neighbourhoods
- running experiments, including:
    - displaying information to users
    - running surveys of different types
    - teaching of the controls by a tutorial
    - exposing participants to virtual environments
    - running simple cognitive test like SART and Eriksen
    - collecting and saving data to a remote database

The constraints of Unity made it however impossible for the application to import new types of buildings and environments after the build, making it less reusable. Any new models and layouts that might be used for other experiments or even in the later phases of CiSuUs must be imported in the Unity environment. This means the necessity of having Unity installed (preferably the Pro version for full functionality such as the Oculus Rift) as well as basic knowledge of the engine and editor.

Unity's workflow, its visual editor and numerous tools make it possible for psychologists without background in computer science to take part in the development process. However, although they can be users of applications such as the one presented in this thesis paper, making changes to the product still requires programmer, as scripting remains an inevitable part of any, but the simplest Unity project. That said, the community offer numerous easy to follow tutorials, manuals and support for anyone willing to learn. This makes Unity a good place to start for any teams thinking seriously about research in connection with virtual reality. New releases together with the growing number of features and tools promise new possibilities in the future and its support for the Oculus Rift make it an especially attractive option for further use in research.

Bibliography

[1] A. E. Stamps III, "Advances in visual diversity and entropy", *Environment and Planning B: Planning and Design* 30(3) 449 – 463 , 2003

[2] P. J. Lindal , “Restorative Environmental Design for Densifying Cities”, The University of Sydney, 2013

[3] G. H. Crampton, “Motion and Space Sickness” Chapter 15 “Simulator Sickness”, CRC Press, 1990

[4] S.L. Trawley, A.S. Law, M.R. Logie, R.H. Logie, “Desktop virtual reality in psychological research: a case study using the Source 3D game engine”, http://www.psy.ed.ac.uk/resgroup/MT/documents/SourceEngine_2013.pdf retrieved on 11.09.2014

[5] S.P. Smith & D. Trenholme, “Rapid prototyping a virtual fire drill environment using computer game technology.”, Fire safety journal 44 (4), 2009

[6] “Occulus Best Practises Guide” , http://static.oculusvr.com/sdk-downloads/documents/OculusBestPractices.pdf, retrieved on 30.06.2014

[7] D. A. Washburn, “The games psychologists play (and the data they provide)”, Behavior Research Methods Instruments and Computers, Vol 35, 2003

[8] Official Unity3d website: http://unity3d.com/ , as of 10.09.2014

[9] Official Unity3d documentation: http://docs.unity3d.com/Manual/ retrieved on 14.06.2014

[10] Official Occulus website: http://www.oculusvr.com/ as of 10.09.2014

[11] Occulus Developer website: https://developer.oculusvr.com/ as of 14.06.2014

[12] Microsoft Developer Network, MSDN Library http://msdn.microsoft.com/library/ retrieved on 17.06.2014

[13] Virtual Game Lab, S.T. Koenig, http://www.virtualgamelab.com/blog/unity-for-research, retrieved on 14.06.2014

[14] S.T. Koenig "*Individualized Virtual Reality Rehabilitation after Brain Injuries."* PhD dissertation, Human Interface Technology Lab New Zealand, College of Engineering, University of Canterbury, Christchurch, 2012
[15] B. A. Eriksen & C. W. Eriksen, "Effects of noise letters upon the identification of a target letter in a nonsearch task.", Perception & Psychophysics, 1974
[16] I. H Robertson, T. Manly, J. Andrade, B.T. Baddeley & J. Yiend, (1997). "'Oops!': Performance correlates of everyday attentional failures in traumatic brain injured and normal subjects", Neuropsychologia, 1997

Streszczenie pracy dyplomowej
Wykorzystanie silnika gier Unity do wspomagania badań psychologicznych

1. Wstęp

Praca skupia się na zbadaniu możliwości wykorzystania silnika gier Unity do wspomagania badań z zakresu psychologii, poprzez analizę wymagań stawianych aplikacjom które mają spełniać taką rolę oraz przegląd funkcjonalności silnika Unity. Część praktyczna powstała jako część projektu Cities That Sustain Us (CiSuUs), realizowanego przez instytut CADIA przy Uniwersytecie Reykjaviku, którego celem jest sprawdzanie wpływu rodzaju zabudowy miejskiej na regenerację ze stresu. W ramach projektu stworzony miał być system pozwalający na tworzenie i przeprowadzanie testów z wykorzystaniem wirtualnej rzeczywistości.

2. Technologie

Wykorzystane technologie obejmują silnik gier Unity oraz Oculus Rift. Unity jest popularnym narzędziem do tworzenia gier komputerowych. Jedną z jego głównych zalet jest łatwość nauki, którą umożliwiają intuicyjny edytor graficzny, dokumentacja, liczne samouczki i duża społeczność. Unity dostarcza licznych narzędzi wspomagających proces tworzenia gier, a także daje możliwość rozszerzania funkcjonalności poprzez skryptowanie edytora lub dodatki zakupione z Asset Store'a. Kolejną istotną z punktu widzenia projektu zaletą jest wsparcie dla Oculus Rifta (dostępne jedynie w płatnej wersji - Unity Pro).

Oculus Rift jest urządzeniem pozwalającym na doświadczanie wirtualnej rzeczywistości poprzez zakładane na oczy okulary wyświetlające stereoskopowy obraz. Urządzenie to śledzi ruchy głowy 360°, pozwalając użytkownikowi na rozglądanie się po wirtualnym środowisku. Oculus jest obecnie nadal w fazie tworzenia, choć dostępne są zestawy dla developerów. Jego zaletą w porównaniu z wieloma wcześniejszymi rozwiązaniami ma być lekkość i przystępna cena.

3. Technologia gier w psychologii

Informatyka i psychologia to dziedziny, które już od lat są ze sobą łączone. Komputery są w stanie znacznie usprawnić działania psychologów, poprzez dostarczanie narzędzi do planowania i przeprowadzania testów, zbierania wyników a także analizowania danych. Technologie gier i wirtualna rzeczywistość stwarzają nowe możliwości dla badań. Zaletami takich rozwiązań są: powtarzalność testów, dokładna kontrola nad warunkami w jakich przebiega badanie, a także możliwość symulowania warunków trudnych (lub niebezpiecznych) do uzyskania w rzeczywistości. Choć stwarzanie realistycznych

środowisk może pomóc w uzyskaniu bardziej wiarygodnych reakcji od uczestników, zachodzi jednak niebezpieczeństwo, że złożoność takiej symulacji może znacznie utrudnić analizę wyników. Przeciwko takim metodom mogą również przemawiać nakład pracy jaki musi zostać włożony w stworzenie wirtualnych środowisk oraz problemy ze zbieraniem wyników.

Tworząc aplikacje wspomagające przeprowadzanie testów psychologicznych, należy zwrócić uwagę na szereg różnych aspektów, które odróżniają je od gier. Po pierwsze, grupą docelową mogą być osoby nie mające na codzień styczności z komputerem, po drugie, w większości przypadków nie będą miały one czasu na nauczenie się aplikacji, gdyż testy często wykonywane są jednokrotnie, przy określonym limicie czasowym. Równocześnie, błąd użytkownika jest dużo większej wagi niż w przypadku gier, gdyż zaważa na wynikach badań. Stąd szczególny nacisk musi zostać położony na intuicyjność aplikacji i jasność instrukcji. Dodatkowo, dla aplikacji wspomagających projektowanie, istotna jest parametryzacja i możliwość dostosowania języka do pochodzenia uczestników.

Przy przeprowadzaniu testów nie można zapominać o bezpieczeństwie użytkowników. W przypadku aplikacji komputerowych z wykorzystaniem wirtualnego środowiska oznacza to przede wszystkim zadbanie o bezpieczeństwo stanowiska oraz minimalizację ryzyka wystąpienia choroby symulacyjnej.

4. Projekt

Aplikacja stworzona w ramach projektu CiSuUs miała na celu wspomaganie procesu projektowania i przeprowadzania testów psychologicznych. Została ona podzielona na dwie części - edycyjną i wykonawczą. Część edycyjna jest przeznaczona dla psychologów i ma pomagać w ustalaniu planu testu, jego kolejnych etapów i ich kolejności, a także dobieraniu parametrów i ustawień dla poszczególnych etapów. Część wykonawcza jest odpowiedzialna za przeprowadzanie testu według przygotowanego scenariusza.

Test psychologiczny podzielony jest na kilka etapów, które mogą być wykonywane dowolną ilość razy i w dowolnej kolejności. Etap informacyjny pozwala na wyświetlanie tekstu i obrazów w celu dostarczenia instrukcji czy dodatkowych informacji uczestnikowi badania. Etap ankiet pozwala na zebranie danych od uczestnika poprzez formularze. Kolejny etap pozwala na przeprowadzanie prostych testów (takich jak np. SART), których celem jest wywołanie zmęczenia lub jego pomiar. Centralną częścią aplikacji CiSuUs jest etap wirtualnej rzeczywistości, podczas którego uczestnicy mogą poruszczać się po wirtualnym świecie i oglądać go przez Oculusa. Aby pozwolić użytkownikom na zaznajomienie się z tą technologią i sposobem poruszania się po otoczeniu, stworzony został dodatkowy etap stanowiący samouczek.

5. Implementacja

Tworzenie aplikacji w Unity odbywa się z wykorzystaniem Edytora, który pozwala na zarządzanie projektem i wizualizację jego obecnego stanu. Elementy świata

wirtualnego rozmieszczane są na scenach w postaci obiektów, do których dodatkowa funkcjonalność może być dołączana za pomocą komponentów. Potężne możliwości daje skryptowanie, które Unity wspiera w 3 językach (C#, JavaScript i Boo).

W implementacji interfejsu użytkownika przydatne są klasy GUI oraz GUILayout. Pozwalają one na stworzenie rysowanie elementów interfejsu jako overlay we współrzędnych okna. Unity dostarcza znaczną ilość podstawowych kontrolek takich jak przyciski, suwaki czy pola tekstowe. Znacznym ułatwieniem jest opcja automatycznego rozmieszczania elementów, udostępniana przez klasę GUILayout. Natomiast dla większej kontroli nad wyglądem interfejsu można użyć klas GUILayoutUtility i GUIStyle.

SDK Oculus Rift dostarcza pakietu umożliwiającego prostą integrację z silnikiem Unity. W jego skład wchodzą pluginy a także standardowy kontroler, który może być od razu umieszczony w scenie. Wymaganiem dla używania Oculusa jest posiadanie Unity w wersji Pro.

6. Podsumowanie i wnioski

Silnik Unity pozwolił na stworzenie aplikacji wspomagającej badania psychologiczne, spełniającą większość stawianych wymagań. Jego dużym atutem jest łatwa integracja z Oculus Riftem, który otwiera możliwość wykorzystania wirtualnej rzeczywistości. Graficzny edytor oraz liczne narzędzia pozwalają psychologom uczestniczyć w procesie powstawania aplikacji. Do stworzenia pełnego produktu wymagane są jednak umiejętności programowania, jednak łatwość nauki, prężnie działająca społeczność i duże możliwości tego silnika mogą skłonić grupy badawcze do zainteresowania się tym silnikiem.