Norwegian University of
Science and Technology

Development of Gaze Tracking Platform

Author(s)

Bjørn Kaare Aune
Kristoffer Baardseth
Benjamin Gordon Wendling

Bachelor in Game Programming
20 ECTS
Department of Computer Science
Norwegian University of Science and Technology,

16.05.2018

Supervisor Simon McCallum

Sammendrag av Bacheloroppgaven

Tittel:	**Utvikling av gaze tracking plattform**
Dato:	16.05.2018
Deltakere:	Bjørn Kaare Aune Kristoffer Baardseth Benjamin Gordon Wendling
Veiledere:	Simon McCallum
Oppdragsgiver:	Progress Interactive AS
Kontaktperson:	Richard Barlow, richardjbarlow@googlemail.com, +47 46746741
Nøkkelord: Antall sider: Antall vedlegg: Tilgjengelighet:	Norway, Norsk 166 Åpen

Sammendrag:	Denne bacheloren tar for seg vårt arbeid med en interaktiv plattform basert på gaze tracking. Den tar for seg utvikling og diskusjoner rundt bruken av gaze tracking, samt hvordan tilrettelegge for dette i spill og annen software.

Summary of Graduate Project

Title:	**Development of Gaze Tracking Platform**
First Date:	First 16.05.2018
Authors:	Bjørn Kaare Aune Kristoffer Baardseth Benjamin Gordon Wendling
Supervisor:	Simon McCallum
Employer:	Progress Interactive AS
Contact Person:	Richard Barlow, richardjbarlow@googlemail.com, +47 46746741
Keywords:	Thesis, Latex, Template, IMT
Pages:	166
Attachments:	
Availability:	Open

Abstract: This bachelor presents our work with a prototype for a software platform based on gaze tracking input. It will discuss our development, and use of gaze tracking and how to facilitate games and software for gaze tracking input.

Preface

We would like to thank Richard Barlow giving us this project, as well as lending us the necessary equipment.

We would also like to thank Martin Sandberg and Odd-Kjetil Aamodt Dahl for inviting one of us to visit Sunnaas Sykehus, and Sunnaas for graciously welcoming us on short notice. In addition, we would like to thank all those who wanted to test our prototype and provide us with valuable feedback.

We would like to thank Simon McCallum for being our supervisor. He has provided us with aid and guidance throughout the project, as well as providing us with the LaTeXtemplate for this thesis.

Clarifications

The terms "eye tracking" and "gaze tracking" are often used in this thesis. They are used to refer to different parts provided by an eye tracking system. When the term "eye tracking" is used, what is referred is the system or functions that track the movement and position of the eye. The term "gaze tracking" is used to refer to tracking where the user is looking, e.g. at what point of the screen.

The terms GameObject and game object are similar, but GameObject (capital G, O, one word) is used for Unity3D's GameObject class. Game object (two words) is used when referring to objects in a game.

Contents

List of Figures

List of Tables

Listings

1 Introduction

In this chapter, we will cover our initial motivation and plans for the project.

1.1 Project Introduction

The extended use of Gaze Tracking in software is still a very fresh field, becoming more available to the market as the technology advances. While many games are now integrated with gaze tracking, this is still a limited field[2]. The use of gaze tracking as the single input method in a software is most often aimed at those who suffer from extensive paralysis or physical handicaps. The aim of this bachelor is to prototype gaze tracking software for the everyday user.

1.2 Background

During the fall semester of 2017, Richard Barlow visited NTNU in Gjøvik to discuss different Bachelors he wanted to present. One of these were the development of a computer program aimed at quadriplegics. This meant using gaze tracking to control a program, to play games or do everyday tasks. The opportunity to work with gaze tracking hardware and software was one that we considered interesting. The decision to work with these kind of systems was quickly made. The added idea of this as a "software for health"-project helped cement the decision.

Our starting point was to create a software that could be deployed for quadriplegics to use as a way to interact with computers. This was changed during the course of the project, to become more of a project in which the design of UI with gaze and how to plan and execute a project of this kind. The reasons for this will be discussed in further detail later in the Discussion.

1.3 Project Description

The project is aimed at developing a software allowing basic interactions through gaze tracking. The development is done with the game engine Unity3D. The goal is to make software that allows a user to interact with simple games, a web-browser, and chat using their eyes and gaze as the main input.

Our main focus has been on researching and prototyping User Interface(UI) and gaze tracking input functionality for our application. They lay the foundation for the functionality that we may wish to extend upon, such as different games. In addition, they were viewed as problematic by Sunnaas Sykehus (4.1.1).

1.4 Scope

1.4.1 Project scope

The main field of study is the use of eye tracking / gaze tracking to implement the basis for a software platform where gaze tracking can be used as the only input. The

development will be done through a game engine, and and will mainly consist of code in C#. The end product will be available at BitBucket/GitHub as an open source software.

1.4.2 Restrictions

We will not be looking at implementing a gaze tracking system using only the hardware and API as the input. It will be done through a game engine, and will be available as an executable.

The language will be in English, since our employer isn't a native Norwegian speaker. We wish to make it open source, and by having it in English we can reach a broader audience.

The system will be tailored to a specific eye tracking system. Integrating other systems can be considered when these are more readily available to us, but will not be done as part of the thesis.

1.4.3 Goals

Project Goals

The goal of the project is to deliver a prototype software with gaze tracking as the primary input method. This software should contain four specific types of functionality: web browser, a game, online communication, and a writing tool.

Learning Goals

The project contains multiple learning goals:

- Learn to develop a software utilizing gaze tracking.
- Familiarize more with integrating existing software and functionality into our own software.

1.5 Target audience

1.5.1 Software Audience

The intended audience of the software are people interested in using their eyes for computer interaction. A potential focus group is quadriplegics or the physically disabled.

1.5.2 Thesis Audience

The thesis is written for anyone who might be interested in creating software with gaze tracking input. It should help provide some insight into what decisions we made, why we made them, and why they could be important to consider when developing gaze tracking software.

1.6 Development

1.6.1 Team Members

The thesis group consists of three members; Bjørn K. Aune, Kristoffer Baardseth, and Benjamin G. Wendling. All are Game Programming students at NTNU in Gjøvik. We have programming experience with C++, C#, and Java, and multiple game engines, such as Unity3D. All share a common interest in Games for Health, which is one of the reasons we decided to take this project.

1.6.2 Development Plan

Our employer did not give us any specific functionality requirements when starting the project. The core functionality of the software is to be able to interact with a computer through gaze, allowing you to play games, browse the web, and more. This gave us a lot of freedom, and we set a plan for what functions were interesting to implement that allowed user interaction.

The development plan for the project was initially made in the planning phase at the very start. The plan, shown in Appendix A: Plan Template, sets up a best-case scenario for how the components of the software was to be developed. While the Sprint-plan (Plan Template p. 10-12) is not a good representation of the results of the development, it provided us with a backlog for the project. This gave us a good "framework" when developing, since our supervisor had helped us discuss the contents.

1.7 Thesis Structure

This thesis contains ten different chapters, each with a specific focus. Below is a short description of each chapter's contents.

1. Introduction: The background, purpose, scope, and a project overview.
2. Requirements: The functionality and system required for the program, and the platform it will be developed on.
3. Technical Design: The architecture of the software and its components.
4. User Interface Design: The decisions of User Interface design in the software.
5. Development Process: What tools were used to develop the software, and how they were used, as well as the process behind it.
6. Implementation: How the functionality of the software was implemented.
7. Deployment: How to install and use the software.
8. User Testing and Feedback: The results and a discussion of the testing.
9. Discussion: Discussion of results of project as a whole. Potential future work of the project.
10. Conclusion: Evaluation of the project, how the team worked, and what we can take away from the process.

In addition to these chapters, there are six appendices.

(A) Terminology: A list of terminology commonly used in the thesis.
(B) Plan Template: The plan template for the project.
(C) Contract: The contract with our employer.
(D) Tobii SDK License v2: The license for using Tobii's SDK.
(E) Test Questionnaires: The questionnaires used for user testing.
(F) Doxygen Documentation: Generated code documentation.

2 Requirements

This chapter will cover the system requirements, core functionality, and the basis for the support of the chosen hardware.

2.1 Functional Requirements

The main goal of the application is to allow anyone to utilize computer functions, such as communicating online, using a web browser, and play games. This functionality should be available with the use of gaze tracking.

The software itself should be designed in a way that makes it easy enough to use so that a physically limited user needs little or no help from an aide. This means that once the software is setup and running, the user should be able to navigate near all functions with gaze tracking only.

To fit these descriptions, the software needs some basic functionality:

- Functionality
  - Ability to use an integrated web browser with a gaze tracker based pointer.
  - Ability to play a simple game, such as Chess or Match-3.
  - Ability to write and output a text file.
  - Ability to chat with another user online.
- Customization
  - Options to change UI interactions.

This functionality is what we deem essential to have a prototype software.

2.1.1 Use Case Model

The use case model for our software is illustrated in figure 1.

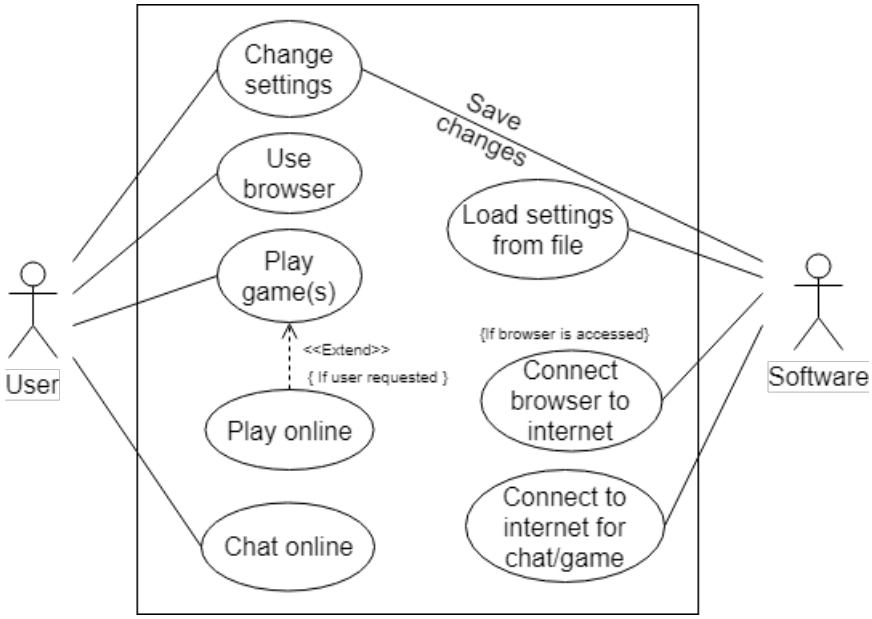


Figure 1: Use case model for the software

2.1.2 High-level Use Cases

User

Table 1: High-level Use Cases: User

Use case	Change options
Actor	User
Goal	User can change settings, such as dwell time
Description	User presses "settings"-button. New window enables user to change UI settings.

Use case	Use browser
Actor	User
Goal	User is moved to window where he can browse websites
Description	User presses "Browser"-button. New window gives user access to web browser and its functionality.

Use case	Play game(s)
Actor	User
Goal	1. User is moved to game overview. 2. User is moved to implemented game
Description	User presses "Games"-button. 1. User is moved to window with list of available games 2. User is moved to window with game to play.

Use case	Play online
Actor	User
Goal	User is moved to online game lobby
Description	User presses "Play Online"-button. User is moved game lobby, where the user can choose to play with others.

Use case	Chat online
Actor	User
Goal	User is moved to online chat lobby
Description	User presses "Chat"-button. User is moved to chat lobby. Here the users can chat with other users

Software

Table 2: High-level Use Cases: Software

Use case	Load settings from file
Actor	Software
Goal	Previous settings is loaded to running program
Description	Software is started. If the settings-file is already existing, load values for settings from this. Else file will be created with standard values.
Use case	Connect browser to internet
Actor	Software
Goal	Connection to internet is established, websites can be loaded.
Description	Software establishes an internet connection. Website data is continually loaded as this is in use.
Use case	Connect to internet for game/chat
Actor	Software
Goal	Connection to server is established, users can chat or play games
Description	Establishes connection to the server. Makes sure connection is running.

The use case diagram in figure 1 and high-level use cases in tables 1 and 2 does not describe the detailed functionality when using browser, game, chat, or settings in detail. This is because the functionality can rapidly change after testing and reviewing, and use cases can quickly become obsolete. For this reason, the use case diagram and descriptions consider the overall structure and functionality of the software.

2.2 Hardware

2.2.1 Eye Tracking system

To be able to properly develop, test, and use the software, hardware that enables gaze tracking is necessary. The system should meet certain conditions, so that the user experience of the software is at an acceptable level. These conditions are:

Development Developers should be able to build upon this system, either by using an Application Programming Interface(API) or a Source Development Kit(SDK) to implement it in the software.

Precision The precision of the system should be good enough so that a user can use it to select functions through buttons, screen areas, and similar functions.

Licensing The license of the hardware and/or software should allow developers to use the Gaze Tracking in public applications, at least with non-commercial software.

There are several options to choose from regarding gaze tracking systems, with Tobii, FOVE, and aGlass being discussed the most. These different systems can provide:

Tobii Tobii offers multiple available systems that we can use, both VR and screen-mounted[3][4]. It offers a SDK (Software Development Kit) for Windows, a plugin for Unity3D, with one for Unreal under development [5].

FOVE FOVE offers a VR headset with a built-in eye tracking system. It offers a SDK for Windows, as well as plugins for Unity3D and Unreal[6].

aGlass aGlass offers a HTC Vive input which allows gaze tracking features to be used with a previously owned headset. They offer a SDK for Windows, as well as plugins for Unity3D and Unreal[7].

The availability of aGlass was limited, and it was hard to get good information and pricing info. There was a risk that the system would be delivered too late, or an error was made when ordering.

FOVE was available, but had the drawback of a high cost. The FOVE costs $599 USD (as of May 2018), before taxes. At approximately 7.500 NOK, it was too expensive for us as students.

This leaves us with Tobiis VR system, which has the same problem. Their VR Development Kit is priced at request, and would likely cost around 10.000 NOK, more than the FOVE.

Our employer was able to lend us a Tobii EyeX he had access to. The EyeX is a screen-mounted "bar" with gaze tracking capability. We made the decision that the project could be done outside of a VR environment, and decided to use the EyeX.

The EyeX was acceptable within all of our requirements. It offered an API and a SDK, and the developers license allowed for the development of interactive software, within some boundaries[8]. The license can be found in appendix D. There were some concerns regarding the accuracy and precision of the system, as can be read in "Toward Everyday Gaze Input: Accuracy and Precision of Eye Tracking and Implications for Design" by Feit, et al.[9]. However, it was precise enough that we decided to use it.

The EyeX is discussed further in Tobii EyeX(p.26).

2.2.2 Target Operating system

Part of our goal is to make the software widely accessible. Our focus is on stationary computers and laptops, where the range of operating systems(OS) is large. Available to us is Windows, OS X, Linux, and Chrome OS. Windows has a clear majority at about 80% of the market[10][11]. In addition to this, the Tobii EyeX and 4C only supports Windows at the moment, with some possible configurations working on Apple's OS X[12]. Because of these factors, Windows was the most natural platform for us to develop for.

2.3 Development Platform

Using Unity3D3D for development was planned early on. The employer had mentioned this as a possible engine when announcing the project, and that Unity3D would be beneficial for future expansion of the software.

In addition, Tobii currently only supports Unity3D, so using another engine would have required us to fix the integration ourselves. We decided that this was not something we wanted to do as a part of this project.

Another option would be to develop the software without Unity3D, using their NuGet package with Visual Studio. On one hand this could give us more control over our application, however using Unity3D3D makes it easier to create and edit User Interfaces. Because testing design options and different functionality was in focus, Unity3D was kept as the development platform.

2.4 Game to Implement

2.4.1 Selection Process

As mentioned in Bejeweled in chapter 3.4 we looked at different games to be implemented for this project. Our initial ideas for games were games such as chess, checkers, card games, patience (single player card game), or Bejeweled. Trying to port other larger games, such as FreeCiv, was also considered. It was decided that integrating something like this would be a stretch goal.

When selecting a game, we looked at multiple factors. These factors were:

Game Familiarity How familiar most users would be with the game, without having to explicitly explain the game to them.

Amount of game objects How to navigate a world with many game objects was something that needed testing for our projects.

Interactions The amount of interactions that is required for the game, and how one would interact with them.

These requirements were weighted differently, based on the goals of the project. When testing gaze tracking, seeing how a user interacted with a larger amount of game objects present on screen was important. This can tell us more about the precision and accuracy of the tracker, and how it impacted the user. Limiting the different types of interaction would mean that the results would not differ from interaction to interaction. While seeing how users reacted to different types of interaction, we were more interested in testing repetition of an interaction.

Familiarity is harder to measure, because people's use of games can vary very much user-to-user. We looked at what we personally thought was popular, and discussed with people from outside our group. A type of game that kept coming up was Match-3 games, such as Tetris, Bejeweled, Candy Crush Saga, and more. Especially Candy Crush Saga has been very popular as a mobile game since it's release, marking the popularity of the genre[13].

In the end the selected game was Bejeweled. This is due to the few game interactions, and the high amount of game objects required close to each other. The genre is also popular on the mobile market, which can indicate that many is somewhat familiar with the rules. Bejeweled was chosen over Candy Crush as the rules are simpler, and the game board's size and layout is static.

3 Technical Design

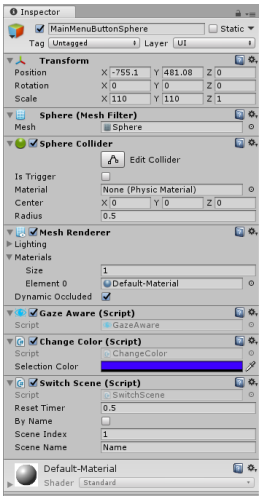
This chapter describes the planned technical design and architecture behind the software. It explains the basis of the engine, implementation, and technical decisions.

3.1 Unity3D

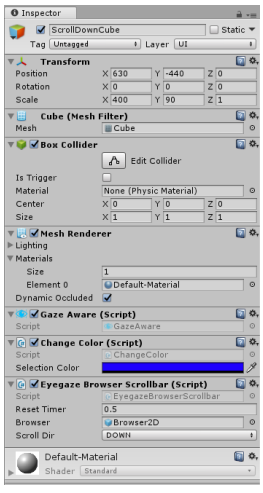
As discussed in Requirements(p.4), the decision was to use Unity3D as our development platform. We've used Unity3D v2017.3.0f3 throughout our project, not updating to more recent versions as Tobii's Unity3D SDK is already somewhat aged. Their last update to the SDK was the 15. May 2017 for Unity3D v5.6, roughly a month before Unity3D v2017.1.0 was released 10. July 2017[14][15]. We had already established that it worked with v2017.3.03f, and did not want to risk the newer version not being compatible. This is discussed in Implementation.

3.1.1 Unity3D Game Objects

The most defining property of Unity3D is it's use of GameObject. In Unity3D, every object in a scene is a GameObject. All GameObjects are then defined by which components they are given[16]. E.g. giving an object the Button-component now gives it the attributes of a button. If we want the button to glow, we can add a light-component to it. Using this system we can create components that can by used by many different GameObjects, even if their intended functionality is different. This can be seen in many of our buttons, as shown in figure 2, where they share many of the same components, but a different script for the action they perform.



(a) Unity3D Editor: Components of a main menu button.



(b) Unity3D Editor: Components of a scroll button.

Figure 2: Editor view of two different buttons. Note that the components are very similar, except for the last scripts, "Scene Loader" and "Eyegaze Browser Scrollbar".

This gave us some flexibility when developing, because we didn't have to tailor the scripts we wrote to a single function. The use of GameObjects and components means that scripts can be generalized, and then setup in the editor. This saved time when coding, and reduced the total amount of scripts required.

Additionally, GameObjects can be given child GameObject. This is very useful for creating groups of objects that are dependent on each other, and is used extensively for implementing UI canvases.

3.2 Singleton Design Patterns

The use of singletons in software, especially Unity3D, is useful when data is used across multiple objects. In Unity3D, a singleton can be loaded and used across Scenes, without being re-initialized. At the same time, a singleton makes sure that there is only one instance of the object being referenced. This makes sure that all values are persistent.

3.2.1 Game Manager

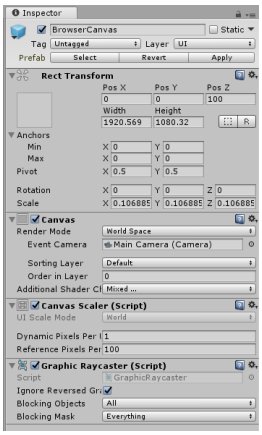
A Game Manager(GM) will be used to store all persistent data (user settings, variables) for the software. It will handle actions that can happen independently of scenes, such as loading other singletons.

3.2.2 Scene Loader

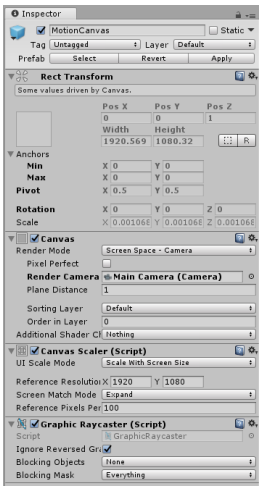
A scene loader will be used to switch scenes in the software. Having the scene loader as a singleton means that the action for switching scenes can be the same for all functions.

3.2.3 Network Manager

Having the network manager as a singleton means that all network actions can be performed over the same class and connection. In turn, all network based activities (except the web browser) can be done through one object.



(a) Unity3D Editor: The canvas for the browser



(b) Unity3D Editor: The canvas for motion, used for zoom.

Figure 3: Editor view of two different canvases. The browser canvas is set to world space and will remain static, while the motion canvas follows the camera, and will scale according to screen size.

3.3 UI Architecture

3.3.1 Unity3D Canvases

Most of the UI will be done with Unity3D canvases. In Unity3D, a canvas is a 2D-layer targeted specifically at making UI, with several editor options to specify size, position compared to screen, and more. An example of the canvas editor can be seen in figure 3.

Due to the fact that the canvas is a Game Object, it is possible to store with all it's child objects. Because of this a canvas can be instantiated in multiple scenes without having to be remade. Using this method, UI can easily be reused and modified in scenes with different functionality.

3.3.2 Clicking with Gaze

There are multiple suggested ways to click when using gaze. Tobii themselves suggest 3 ways of clicking: dwell, blink, and switch[1].

Dwell Using dwell means recording the amount of time a user looks at a specific point or area, and after a given time clicking it.

Blink A click will be sent when the user blinks. It will send the click to the last recorded position.

Switch A switch is essentially a simplified mouse, where the user can press a button to click where they are looking.

The software is intended to be used with gaze tracking exclusively, so the switch will not be discussed further. However, it might be worth noting that the clicker could be the best solution. This is mentioned later in Discussion.

Between dwell and blink, there are multiple issues. For dwell, input and response time can pose a challenge. Users will have to spend a certain amount of time waiting for the software to react to their decisions. This makes the response feel slow, and users can become impatient.

Blink poses the problem of recognizing what is a blink, because for the tracker it would be represented as a "loss of data" (the tracker can no longer find the eyes). It could be imprecise to use blink for clicking as the tracker might momentarily lose the position of the user for other reasons. In addition, a user may blink for longer amounts of time due to dry eyes, or a medical condition.

Comparing the two options, dwell was decided to be the preferred input method. In Implementation, this is referred to as "fixation time".

3.4 Bejeweled

Part of what we wanted to test by implementing bejeweled was the accuracy of the gaze tracker, and the viability of using a gaze tracker to select from objects positioned close to each other in world and screen space. For this purpose, we considered a few different games to implement and test, before deciding on implementing Bejeweled.

3.4.1 Structure

To simplify the logic needed to play Bejeweled, it was designed with a partly monolithic architecture in mind. All functionality required to play the game, bar the rendering of

the game world and the extra UI required, were designed to be included in one class. A Bejeweled controller will handle all functionality for Bejeweled, and the storage of the GameObjects used to represent jewels and the game's border. The same monolithic approach allows for quick transition between the use of gaze tracking and regular mouse and keyboard as input, and makes the game's pause mechanic easy to implement.

While the system is created for gaze tracking, having mouse and keyboard as input methods is still useful. It allows for the testing of the base mechanics without the use of a gaze tracker.

3.4.2 Game Logic

Initialization of the game board instantiates the game objects for the board and the border, and giving the board game jewels their highlight mechanic, and color. It will then make sure that there are no patterns on the game board at the start of the game.

A simple update-loop forms the basic of the game's logic. It runs for as long as the game is running, and does the same checks each time. By having the check for patterns in the game board only run when a jewel is moved, the amount of calculations will be slightly less per frame.

For pattern checking, the algorithm was created with the variety of patterns that exists in bejeweled in mind (see figure 8 in User Interface Design). The check will go through all jewels in the game board, and see if there are any patterns attached to the current jewel. It checks the two jewels that came before it, first from the side, and then above. When it finds a pattern, it returns the position of the pattern, and direction it is going. The patterns are then flagged for removal.

Pattern removal loops though the board, and removes the flagged jewels, and then moves the jewels above it down. After this is done, it will then check for new patterns, until there are no patterns in the game boards, and the update loop starts anew. While removing patterns, the game will allot an amount of points to the player.

Pausing will be handled by a simple Boolean check in the update loop.

Figure 4 shows the working sequence of the update loop for our architecture of Bejeweled.

3.4.3 Gaze Tracking Integration

Interaction with the gaze tracker is to be implemented using the relevant SDKs for the gaze tracker available to the project. The interaction required will be the selection of jewels, which then will run the same logic for movement as when playing with mouse and keyboard. For the selection process, there will be a need to highlight which jewels that is being looked upon. One way to solve this will be to attach a light as a child object to the jewel, and to make it light when the jewel is the focus of the gaze tracker.

3.5 Chat

Part of the project is to give the users a possibility to communicate with gaze tracking. Two distinct models were considered, the Client-Server model, and Peer-to-Peer(P2P). The decision was made to implement Client-Server for several reasons:

- Client-Server makes it easier to write different clients for the same chat system.
- Client-Server allows us to save chat-logs easier.

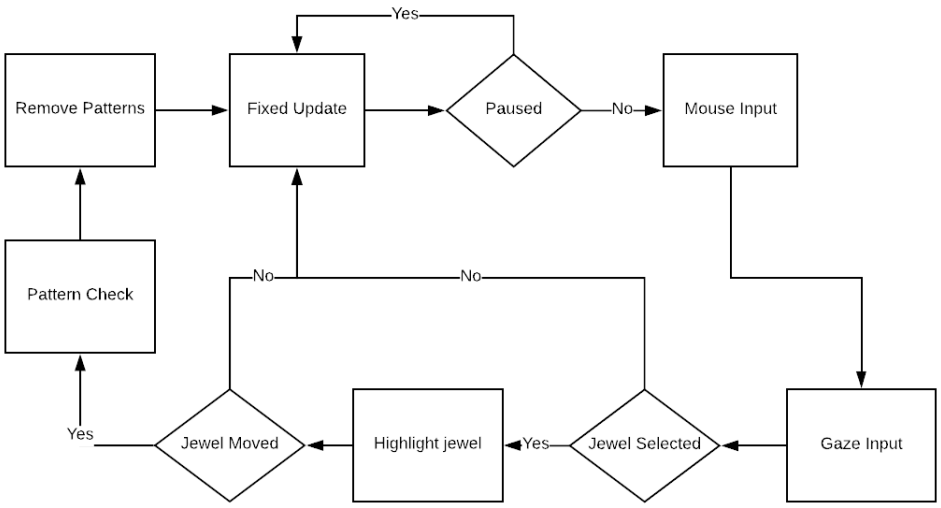


Figure 4: Flowchart of Bejeweled FixedUpdateLoop

- We already have experience with this system.

The possibility to write different chat clients for the same system creates the opportUnity3D to communicate across different software. This functionality can be included if cross-play between the gaze tracking software and regular mouse-keyboard software is wanted.

Saving chat-logs makes it possible for users to read previous messages sent before the entered the chat. It also creates an easier system for moderation, should it be needed.

The chat system is done by setting up a back-end server that handles communication between systems. The system should is scalable, in the event that the amount of users increase beyond current capacity.

3.5.1 Communication

All communication between a client and a server is handled by sending JSON(JavaScript Object Notation) objects between the connected clients and the server. The reason for choosing JSON as the message structure is that they are easy to manipulate using different libraries for it. For full details on requests and responses for the chat system, see Implementation.

3.5.2 Client

The basic functions of the chat client is to be able to send and receive messages between a user and the server. When a message is received it is saved and displayed in the chat window. A user is also able to write messages with a selected recipient in the Chat Window. These messages are then converted to JSON and sent to the server.

3.5.3 Server

The back-end for the chat server is to be implemented in such as a way that it can scale for more users, and to make it easy to interact with. For these purposes a RESTful web

service was chosen as the basis for our server. This was to increase our knowledge on how to write one, and to save time by re-factoring old code. The basis for the server was to be able to communicate with one, and to get a response.

In addition to this, by setting it up as a RESTful Web Service, it will be easy to mock communication with the server by using programs such as Postman. Postman can be used to test the communication server, and is discussed more in Communication testing.

Database

A database is needed when handling communication by Client-Server. Several factors must be accounted for when choosing a database type. We considered MongoDB, CouchDB, and MySQL[17][18][19].

MongoDB NoSQL database that uses JSON-like documents.

CouchDB NoSQL database that handles JSON natively.

MySQL SQL database that can handle JSON.

MySQL requires some workaround when using JSON, and because of this, we decided not to use it. Between CouchDB and MongoDB, both supported the functionality needed. However, CouchDB uses URL-calls to communicate with the server, while MongoDB uses an API. Adding this to the fact that we had previous experience with MongoDB and the GO language, we decided to use it for the database.

For testing purposes, the database is stored locally on the client running the server. The database will also be storing things in plain text for the time being, although steps to secure it will be added in case of further development. For further details on deployment see Deployment(p.46), and for further details concerning further development, see Further Development(p.60).

The database contains two collections The first collection stores information about the users using the database. The other stores all messages sent using this system.

Since all messages are sent and received as JSON objects, they can easily be stored directly by using MongoDB. The same is true for the information about the users, which is also stored by creating JSON objects with he relevant information.

Both message and user information will require the setup of the database API to handle the respective JSON objects. This will also create the searching tools that is required to allow for the service to work.

Main Handling

Upon reception of a update request from a client, the sever will parse the update request for info about the user, amount of messages the user has on it's client, and whereas the user is logging in or out. Originally, it was intended for the client to send two different types of requests. One for updating the information about the user on the server side, which also told the client how many users were online, and how many messages the user had received. The other for receiving a specified message. This was simplified to the server sending all new messages together with the update response.

The server should also know which users are online at any given time. It will do this by checking how long it has been since the last update request received from a specific

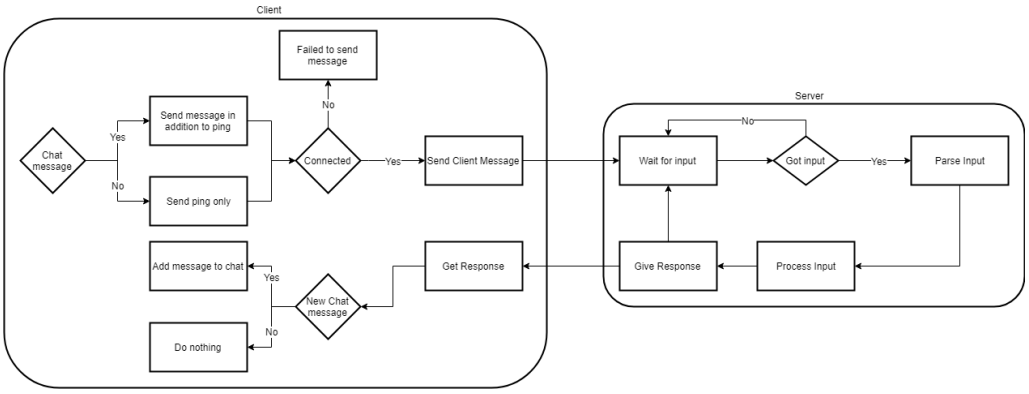


Figure 5: Flowchart of Server and Chat communication

user. When the time since last request passer over a certain threshold, the user will be set as offline. It is also possible to send a set as offline with the update request, upon termination of the session from the client side.

A flowchart of this process can be seen in figure 5.

3.5.4 Security

With all communication tools security becomes an issue. The primary security concerns for this chat system would be privacy. This would be achieved by encrypting all messages between the client and the server, having user log in to the system before using it, encrypting and salting the password of the users, encrypting the database, among others. Because this project focuses on the functionality aspect of developing a gaze tracking system, the chat server has been designed with testing of functionality in mind. Therefor, most of these security features has not been implemented, to allow for quicker testing.

For further discussions on security and implementation, see Further Development(p.60).

4 User Interface Design

This chapter will present the discussions and decisions surrounding design of user interface(UI).

4.1 General Designs

When gaze is the only controller for a software, UI is an important factor to consider. Gaze tracking, even with modern technology, is still imprecise. The Tobii EyeX could be an example of such software, because it sits at the base of your monitor, and the focus seems to be tracking the area the user is looking at, rather than the precise spot[9]. This gives us a frame of limitations that we had to work with when designing the UI.

4.1.1 Visit to Sunnaas Sykehus

During the project one of our members visited Sunnaas Sykehus, one of Norway's largest centers for physical rehabilitation. The goal was to talk to someone working with eye tracking systems. Many of those who suffer from paralysis or physical disabilities have used eye tracking systems, so the professionals at Sunnaas have a lot of experience with these systems. In addition to confirming some of our assumptions, Sunnaas gave us a lot more insight into problems these users face.

Our biggest worry was the chance that using only the eyes to control a computer would be straining. Sunnaas could partly confirm our hypothesis, saying that some patients could feel tired or "worn out" after using their eye tracking systems. However, it is important to note that this could also mean that they had been straining their necks or backs as well, so it shouldn't be traced directly to Computer Vision Syndrome (discussed further in Computer Vision Syndrome on p.59).

The staff at Sunnaas presented some topics they felt were beneficial in a gaze tracking system, especially if quadriplegics is the focus group of said system. Here are the topics that were presented as the most important ones:

- Simple and straight-forward design: Users would find having too many functions on screen, or unclear functions frustrating. When feedback takes extra time, it is important that users can make decisions fast, and that they get what is expected.
- Utilize small amount of screen-space: avoid having to use corners, and areas furthest away from the center. This is a way to reduce eye strain by limiting how much a user needs to "stretch" their eyes to look at corners, etc. Especially important for using large screens.
- Easy to adjust to individual users. Having some control over how you interact with the system, can allow you to tailor it to yourself. In turn, it can become more intuitive to use, and the slow response time can be alleviated.

The feedback provided more insight into which areas should be considered more extensively.

4.1.2 Layout

When using gaze tracking for input, the UI arrangement is a defining part of the software and how a user interacts with it. As part of the early design process, circular layouts were explored. The goal of these layout designs were to alleviate the need for a large screen space, so a user didn't have to turn their neck or look towards the edge of their vision.

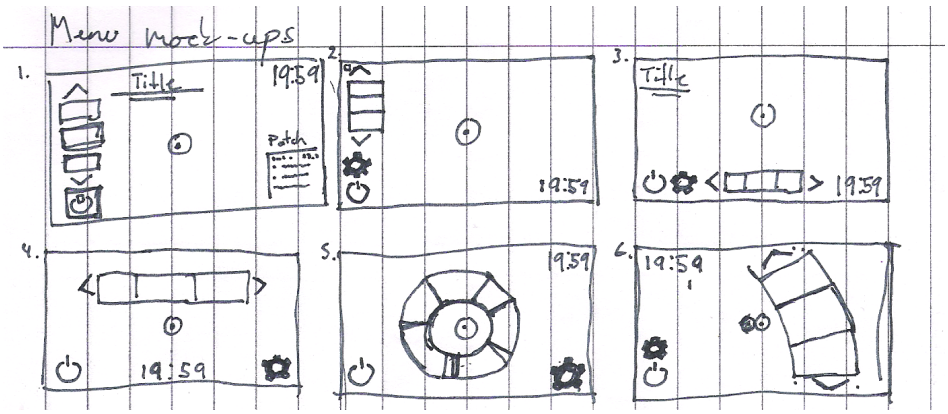


Figure 6: Some of the first UI sketches. While #1 were the first to be implemented, the circular design of #5 became the most central later on.

The different designs displayed in figure 6 represents some of the options that have been explored.

Design #1 through #3 uses the edges to contain all functionality, clearing up space in the center of the screen. Their most apparent drawback is that the user needs to utilize the edges and corners more, which can tire their eyes.

On the other hand, #4 through #6 has a design more geared towards utilizing the center of the screen. The user doesn't have to use the edges that much, but it is harder for a user to rest their eyes without looking at UI elements.

Considering the feedback from Sunnaas, a decision was made to use circular UI design in order to improve the user experience.

4.1.3 Buttons

When performing interactions with gaze, buttons have to be adapted to the imprecision of gaze tracking. Buttons with good naming and icon sets are immediately more intuitive for the user, because they effectively inform a user of their function. This kind of design can be found in many mobile applications, and is something that can be taken into consideration.

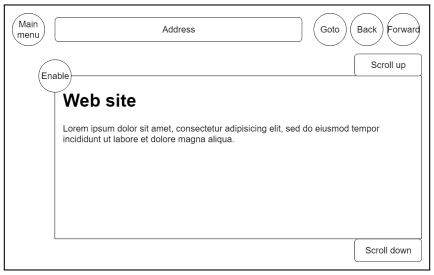
Buttons in mobile applications often have to account for being rendered on small screens, and being used by relatively imprecise input - touch. It can be hard for users to press at the exact right location when using their fingers, which can be countered by larger buttons and intuitive placement. This can likely be transferred to the use of gaze tracking.

When designing buttons for gaze tracking, it is important to keep the limitations of the hardware in mind. Options are limited when the hardware is unable to track the user. However, imprecision in the tracking can be alleviated by larger buttons and better

placement, in addition to self-labeled buttons. Examples of this deisgn can be seen in figure 7.

(a) One of the initial designs for an integrated browser. Note that it does not contain zoom-functionality.

(b) Later design for the browser. Zoom is still not considered, but it now has functionality for writing.



(c) The layout used for testing, containing the basic functionality. A menu for zooming is available through the "Enable" button.

Figure 7: Browser design sketch ups

4.2 Web Browser

For a web browser to work with gaze tracking, most functions have to be adapted to alleviate the lack of precision. For this implementation, this means that buttons needs to be enlarged, and an alternative pointer needs to be used. We graded the necessity of these features for gaze tracking like so:

1. Website navigation - the user is able to navigate the website. We defined the basic functions of this as scrolling, and using a mouse to click links, as well as back/forward.
2. Writing - the user should be able input the address they wish to access, or text on a website. This means accessing a keyboard, writing, and going to specified address.
3. Zoom - the user is able to zoom in/out. When zoomed, the used is able to move in directions up, down, left, and right.

The grading is based on the level of autonomous use of the eyes. #1 allows a user to navigate starting from a website, and navigating back and fort as with a mouse. #2 allows a user to navigate to specific websites, and using text areas such as search engines. This is fairly close to full access of a website. #3 is mostly for convenience, giving the user the possibility to zoom in on text, pictures, and more. We created some simple sketches, shown in 7 to visualize how we might design this UI.

4.3 Bejeweled

It was important to keep the gameplay similar to the original Bejeweled. This will ensure that the gameplay and game rules are familiar to a wide audience.

4.3.1 Game Design

The game design of the implemented Bejeweled is a simplified version of the original. The game board consists of eight by eight pieces, that can be moved within those limits. There is no time limit, and the game ends once there are no more possible moves on the board.

When moving pieces, the only legal moves will be ones that creates one or more patterns in the game. The patterns that are recognized are three - or more - in a row, and creating L-shapes or T-shapes, as shown in figure 8. Upon detection of patterns, the patterns will be removed from the game, with it cascading until there are no more patterns.

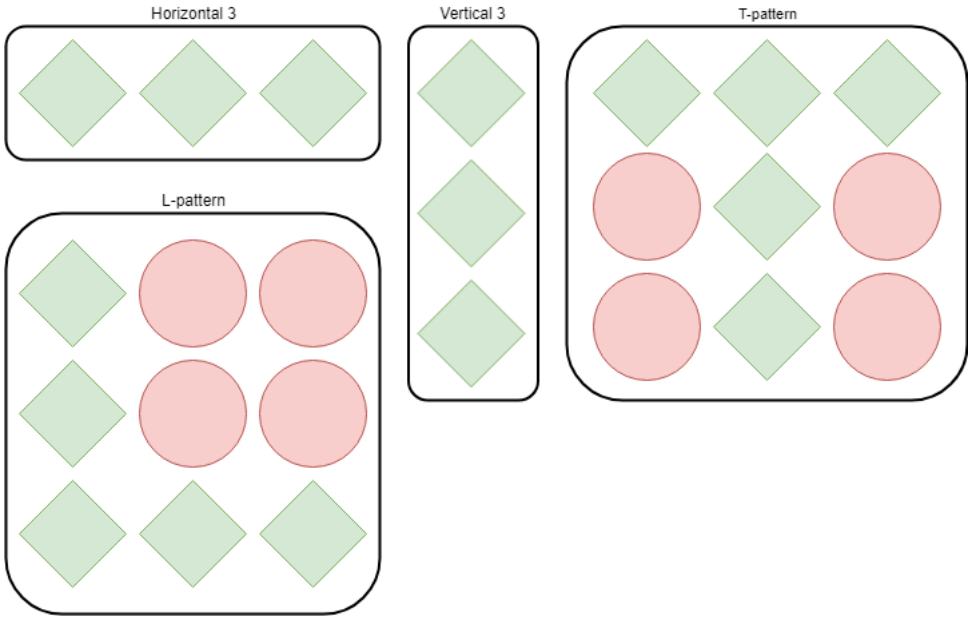


Figure 8: Common patterns in Bejeweled. The green diamonds represent legal patterns.

Another way to give feedback to the user is the use of animations. For Bejeweled, these animations would be when jewels move down on the game board. This feedback would be a great addition to the game, as it makes it easier to identify when jewels are being removed. In addition, the satisfaction that can occur from watching jewels cascading down into new patterns makes people continue playing. Since the project is

to implement a prototype, the need for animations has been put on the backlog. They would be a nice addition to the game, although not necessary for the game to function.

4.3.2 Color Choice

The colors chosen for the jewels were intended to be the same colors of the original game. Because this is a prototype, there was no creation of extra assets for the jewels, and the standard Unity Sphere game object were used. This was to save time on assets, and it would only change the visuals, not the gameplay. Figure 9 shows the original size and color of the jewels.

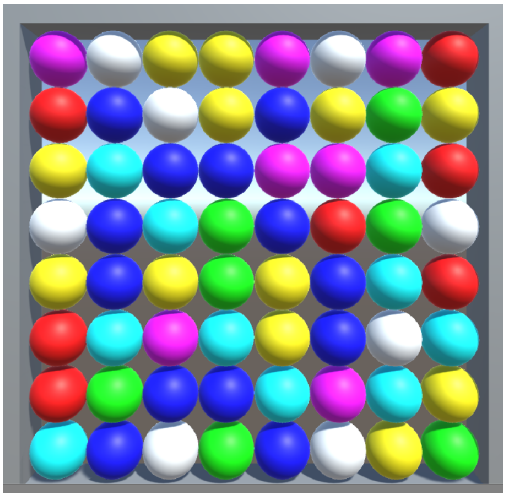


Figure 9: Original Bejeweled Colors

The spheres were given a grey border, to define the space of the game to the user. Gray was chosen as a neutral color so it does not interfere with the more colorful spheres visually.

Considerations

Colors can be perceived differently from person to person as some people are born with color vision deficiencies. This can make the distinction between colors difficult. It is something that should be considered when creating UI, to make sure that all users can distinguish between the different elements. Since the game is about matching pieces, having them be distinguishable is important.

For this prototype, the only difference between game objects in Bejeweled is color and shape. The border is grey, and the jewels have a multitude of strong colors. Because of time, the decision was made to not create different models for the different jewels. While this could make it more difficult to play for some users, it was not deemed important enough for the primary testing.

This problem could be solved by creating different models for the different colors of jewels. In the original Bejeweled by PopCap Games, the jewels have different shapes (see figure 10)[20]. This is something that could be added during further development.

Figure 10: Original Bejeweled, showing colors and shapes for jewels

4.3.3 Extra gaze feedback

Having feedback of where the user looks is important. This will help the user when selecting jewels, and helps tracking their current looked at jewel. To give this feedback, a white dot appears on the jewel that the user is looking at. This will then move when the user looks at a different jewel. This is intended to display at the jewel looked at, although not the precise position, as this is not necessary information when playing bejeweled.

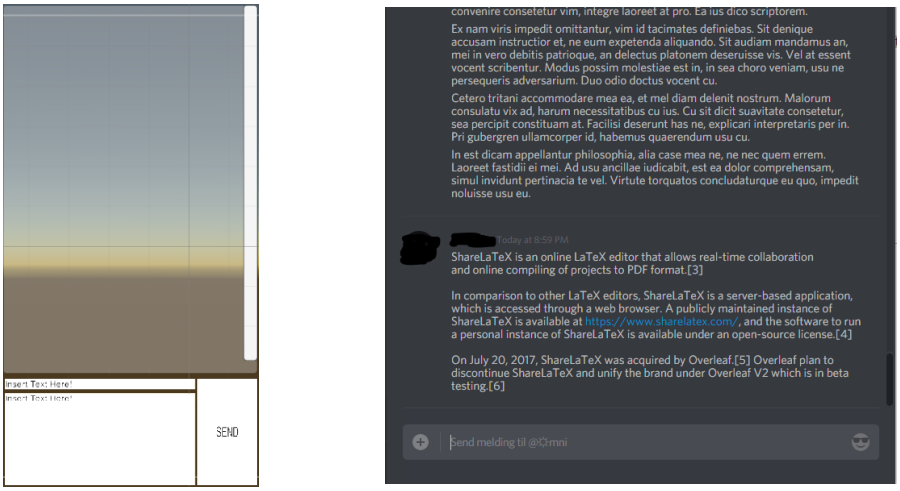
4.4 Chat Client

The layout of the chat client is based on the layout of existing chat clients, such as Facebook, Discord, Slack, and more. This is to create a familiarity for the user when using the client. A comparison of our prototype and Discord can be seen in figure 11.

The general layout can be split into two parts, one part for viewing messages, and another part for writing messages. The chat viewing area contains all messages a user has received or sent. These messages are displayed with a user name, message, and a time stamp. These messages are continually displayed as the chat refreshes.

The part for sending messages contains two input fields. One input field is for the recipient (who the user wants to chat with), and the other input field is for the actual message the user wants to send. Additionally there is a "Send message" button, which sends the contents of the input fields to the server.

Many user interfaces, such as Discord and Slack, does not feature a dedicated "Send" button. Instead they utilize the keyboard "Enter" key to send messages (and often keys shift+enter for newline). Due to the fact that the program is made with gaze tracking as the only input in mind, a "Send" button is very helpful. It doesn't occupy the "Enter" key (in the software, a button), and pressing shift+enter on the gaze tracking keyboard would for many be a tedious process (especially if a message contains multiple newlines). This difference can be seen in the comparison in figure 11.



(a) Image of prototype chat layout in software.

(b) Example of Discord's chat layout.

Figure 11: Layout of our prototype, next to Discord's layout. Note that the scroll bar and text field are roughly equal, but the prototype contains a "Send"-button where Discord has a selection menu for emotes.

5 Development Process

This chapter describes the tools used when developing, and the development model.

5.1 Environment

5.1.1 Unity

Unity3D was decided upon as the engine for our program, and gives two options in choice of scripting language: C# or UnityScript, a "dialect" of JavaScript. In addition to the team having already used C# with Unity before, UnityScript is discontinued from Unity 2017.2[21]. As a result of this, it is natural to use C# for development, so the project more easily can be continued in the future.

5.1.2 Visual Studio

Most of the component scripting were done using Visual Studio, Microsoft's own platform for development of programs using C#. Visual Studio Community is provided free of charge from Microsoft, and Visual Studio's IntelliSense can be used with Unity's script classes, which makes it easier to develop and write code.

5.1.3 Sublime

All development for the chat server was done on a machine running Linux. All code was written using the Sublime text editor. The choice of Sublime as the editor is a preference choice, as it without any packages for GO installed offers nothing that makes it different from Nano or other basic text editors. The code written using sublime was then compiled using command line calls to the GO packages for Linux.

5.2 Tools

5.2.1 Version control

We have used Atlassian BitBucket for version control of the project, with Git as the interface. GitHub was considered at the beginning of the project, because it is open source. However, should we wish to or need to make the repository private due to licensing concerns, our existing BitBucket license would allow this.

5.2.2 Documentation

We have used Doxygen to create code documentation for the project. Doxygen allows to easily create documentation for all of the code. This can be seen in appendix F.

Google Docs have been used extensively to create notation and documentation for the project, to aid in the development of the code and thesis.

Draw.io and Lucidchart have been used to create diagrams for this thesis.

5.2.3 Report writing

To write the final report, the use of LaTeXas a document writing tool was chosen. This was recommended to us, and it allows for the disregard of layout when writing. The writing environment chosen was ShareLatex, as it allowed for simultaneous writing similar to Google Docs. For the LaTeXtemplate, the template supplied by McCallum was selected as the basis for this thesis[22].

The report is written in LaTeX.

5.2.4 Communication testing

When developing the chat server, the API testing tool Postman was used to mock requests to the service[23]. This allowed for the development of the server independently of the client. With Postman being a useful tool for API testing, it sped up the server development process.

5.3 Hardware

5.3.1 Tobii EyeX

The Tobii EyeX is an eye tracker that can be used to track where the user is looking on a screen. Figure 12 shows how IR and cameras are used to calculate where a user is looking.

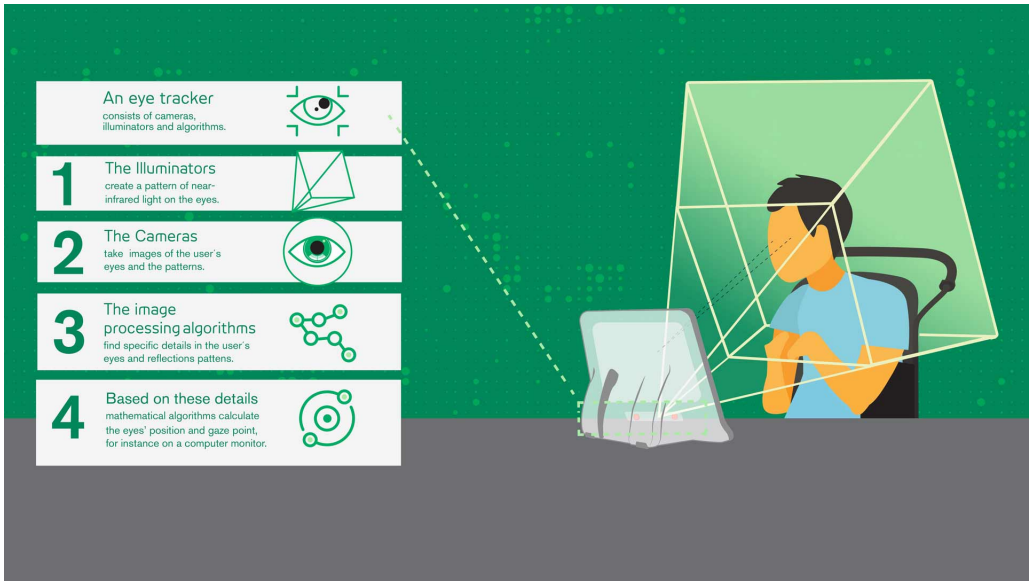


Figure 12: Tobii's diagram on how an eye tracker works[1].

The EyeX is screen-mounted, so it can be used at most computers with a bottom bezel, or something else to affix it to. In figure 13 the EyeX can be seen mounted on the computer we used for testing.

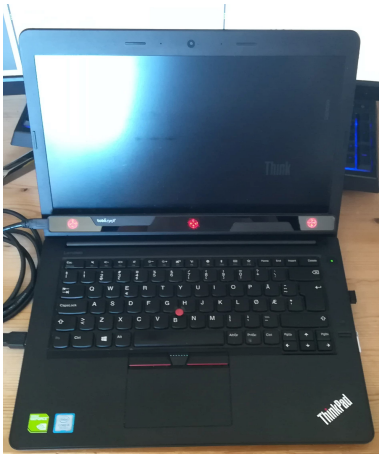


Figure 13: Image of the testing computer with the EyeX eye tracker mounted on the bottom bezel.

5.4 Testing

5.4.1 Unit Testing

Unit test are not something that have been used for the most of this project. This is due to the way the internal implementation process has worked. However, there were unit test for the implementation of the chat database. This was to make sure the database worked properly when implementing it.

For the unit testing, the GOLANG test package was used[24]. The reason for this is that the test package for go is the official test package for go, and it has automated test features built in.

5.4.2 User Testing

When testing the software and the usability of it, user tests with external testers were held. These test were to give feedback on how the software was to use, and to give us extra feedback on the features wanted for such a system. For full information about the user tests, see chapter 8 User Testing and Feedback(p.47).

5.5 Software Development Model

Due to the open ended nature of the project, the use of a rigid model was considered to be inefficient. This is because of the strict structure of the rigid models, and that it would clash with our project's goals. Having every step of development planned beforehand would become a hindrance, as the project would most likely evolve over time.

The use of a agile model would provide more flexibility when developing. This is beneficial as it would be easy to add components to the software. It would also make it easier to adapt to any changes done during the project, and to adapt to the varying development times.

Scrum and Kanban were considered as our development model. As both of these are considered good options for agile development, however there were reasons for us not to chose either method. Where Scrum is concerned, it is a great model for small teams, working with smaller increments during sprints[25]. It has mechanisms for quick

changes to the project plan, and has planned meetings as part of the model. Kanban uses a board to track all features that are to be, are in, or has been developed[26]. This model also allows for things to be added to the product during development. However, Kanban has restrictions on how many items are to be implemented at a given time, and has fewer options for dropping features during development.

The final decision was to use a development model similar to Scrum, but without using all the Scrum artifacts. This is to allow for incremental design and development, without having to include the project owner. Another reason was that the need for daily Scrum meeting was not considered necessary for the team. The best way to describe the end model is that it is an iterative and incremental model, following the basis for the incremental development model.

5.6 Work process

By making the system modular, the workload could be split between the group. This allowed the different group members to work on their separate parts of the implementation, without interfering with each other. Upon completion of modules, they were merged together to the central repository.

When using git, we had separate branches for each module, being merged into the production branch when done. This was to keep the main branch with a running build at all times. When the modules in production was done, they were pushed onto the main branch.

6 Implementation

This section will present how we implemented the software, and showcase some of the code most central for the software.

6.1 Tobii Integration

Integrating the Tobii SDK into Unity3D was done through a Unity3D package. This installs much of the necessary components into Unity3D, as well as some scripts from Tobii themselves. Since the latest SDK was made for Unity3D v5.6, some minor adjustments to the package has been made in order for it to work with Unity3D v2017.1.

6.2 Generic UI

User-Interface and experience has been a key focus throughout development. The program is intended to be comfortable to use, even over longer periods of time while being fully controllable with gaze tracking.

6.2.1 Fixation Time

As mentioned in Technical Design, we decided to use fixation time to handle a user clicking. Throughout the code, this has been done by setting a variable float to a given value. When a user then looks at a button, the method gazeAware.hasGazeFocus() returns true, and a countdown starts. When this reaches zero, a function is entered and an action is performed.

6.2.2 Buttons

Buttons are a critical component in the program. Implementing buttons that can be interacted with using only Gaze Tracking proved to be an interesting task. A click timer was the solution for this problem, if a user looks at a button for a given time (specified in settings file) a button is clicked.

Listing 6.1: Button Selection

```
1       //If the object is being looked at
2       if (mGazeAware.HasGazeFocus)
3       {
4         resetTimer = GM.instance.settings.clickResetTimer;
5         clickTimer -= Time.deltaTime;
6         if (clickTimer <= 0)
7         {
8           OnClickEnter();
9           clickTimer = GM.instance.settings.clickTimer;
10        }
11      }
12      else
13      {
14        //ensures this is only entered once.
```

```
15          if (resetTimer > 0)
16          {
17            //counts down reset timer if object is not being
                looked at
18            resetTimer -= Time.deltaTime;
19            if (resetTimer <= 0)
20            {
21              //resets the clicktimer for a given button
22              clickTimer = GM.instance.settings.clickTimer;
23            }
24          }
25        }
```

The code featured in listing 6.1 shows how a button registers a click. This function is found in an abstract class that all buttons inherit from. That way a unique button only needs to implement its own method of the "OnClickEnter()" function. Having to rewrite the code snippet above in every single button is also circumvented by having the UI_Clickable as an abstract class.

The reset timer is used to prevent the click timer from resetting in case the tracker records the gaze at the wrong position. This gives the tracker some time to correct before the click timer is reset. The reset timer is set to it's default value (listing 6.1, line 4) every frame. While setting the value every frame is an extra operation, the performance impact is negligible. This is because the action only happens in the object that is currently gazed upon.

6.3 Web browser

Unity does not provide an in-game web browser. The options were to either implement one, or integrate an existing solution into our software.

Implementing a browser meant building a wrapper for an existing engine, such as Awesomium or Chromium[27][28]. Building a wrapper could potentially be a very time consuming task, but also be tailored more towards the use of gaze tracking.

Optionally, we could find an existing solution that can be adapted and used with gaze tracking. This means to find an open source solution or buy one that can be used in an open source project. This would most likely be less time consuming, but needs to be adapted for gaze tracking. It would likely not allow us to tailor it like a wrapper could.

Since the goal is to create a prototype that can test and showcase the use of gaze tracking in different settings, integrating with an existing solution was chosen. This would likely be faster to implement, and allow us to adapt and test it faster.

6.3.1 Implementation

The software uses a web-browser integrated into Unity3D. The code and assets for the browser itself is gotten from Vitaly Chasin's Simple Unity Browser[29]. This is a pre-built browser for Unity3D, and was implemented directly into our project. It is an open source solution, so we are able to use it freely and expand upon it.

The browser is intended for regular keyboard and mouse input, and needed some extra functions to work properly with the limited input of a gaze tracker. To allow for basic usability, the gaze tracker should be able to cover the following functions:

1. Mouse click
2. Scroll up/down
3. Write
4. Forward/Back

These functions should allow a user to perform most of the basic tasks done through a browser. Clicking and scrolling allows a user to navigate web pages, and writing allows them to use the address-bar and in-browser writing functions. Forward/Back functions makes it easier for the users to navigate between visited pages, speeding up the process should they wish to visit a page from earlier, or go back to where they were.

The Simple Unity Browser uses a C# handler to pass commands to a script that loads a website. By altering the C# code, and adapting our own Gaze Tracker Pointer, we are able to pass mouse functions to this script based on the Gaze Tracker Pointers position, and "mimicking" mouse actions. An example of this can be seen in listing 6.2.

Listing 6.2: Browser2D.cs: The function for passing mouse actions to the browser

```
1         //Edited by Kristoffer Baardseth to public function
2         public void SendMouseButtonEvent(int x, int y,
              MouseButton btn, MouseEventType type)
3         {
4             MouseMessage msg = new MouseMessage
5             {
6                 Type = type,
7                 X = x,
8                 Y = y,
9                 GenericType = MessageLibrary.
                      BrowserEventType.Mouse,
10                // Delta = e.Delta,
11                Button = btn
12            };
13
14            _mainEngine.SendMouseEvent(msg);
15        }
```

Mouse

To be able to navigate and trace their gaze, we have implemented a pointer to represent the mouse pointer for the user. The pointer, which is a small red dot, is the basis for mouse actions within the browser. This is mainly the clicking-action needed to navigate by clicking on links, images, etc. The idea of using a small red dot is to use a clear color that is easily visible, but small enough not to obstruct the users view.

To "press the left mouse button", or click, the program uses what is called point fixation time. Point fixation time is defined as the time a user keeps their gaze at the same area, with little to no deviation, that constitutes selection.

Listing 6.3 is the code used to translate the current position of the gaze, to one that can be used inside the browser window. This is done by translating the position of the gaze on the screen, to one that is inside the rectangle the browser is displayed on.

Once the position is calculated, two methods are called in the browser, telling it the position of the mouse, and that the left mouse button has been "clicked" once. This will

simulate a mouse press.

Listing 6.3: GazeBrowserPointer.cs: Getting the mouse position and sending it to the browser,doing both LeftMouseDown and LeftMouseUp.

```
1         var raycaster = mBrowserCanvas.GetComponent<
              GraphicRaycaster>();
2         Vector2 localPos;
3         bool downClickSent;
4
5         Vector3 pointerInBrowser = ProjectToPlaneInWorld(
              gazePoint);
6         pointerInBrowser = Smoothify(pointerInBrowser);
7
8         Vector3 pointerInBrowserSpace = mCamera.
              WorldToScreenPoint(pointerInBrowser);
9
10
11        //Translate from screen space to local space in
              browser(Transform)
12        RectTransformUtility.
              ScreenPointToLocalPointInRectangle(mBrowser.
              transform as RectTransform, pointerInBrowserSpace
              , raycaster.eventCamera, out localPos);
13
14        RectTransform trns = mBrowser.transform as
              RectTransform;
15        localPos.y = trns.rect.height - localPos.y;
16
17        //Send mouse event to browser
18        //Sends full press (Click+release), or else it will
              lock
19        mBrowser.GetComponent<SimpleWebBrowser.WebBrowser2D
              >().SendMouseButtonEvent((int)localPos.x, (int)
              localPos.y, MessageLibrary.MouseButton.Left,
              MessageLibrary.MouseEventType.ButtonDown);
20        mBrowser.GetComponent<SimpleWebBrowser.WebBrowser2D
              >().SendMouseButtonEvent((int)localPos.x, (int)
              localPos.y, MessageLibrary.MouseButton.Left,
              MessageLibrary.MouseEventType.ButtonUp);
21    }
```

Scrolling

Implementing scrolling with the browser is done in the same vein as the mouse clicking. Data representing a mouse wheel scrolling is sent to the browser, which then performs the action and updates the browser image. Since most of the user interaction was already done through buttons, this action is done by buttons above and below the browser.

Each button signifies the direction it scrolls by location relative to the browser (over is up, below is down). This is to create some visual relation to the placement of the scroll bar in common browsers such as Chrome, Edge, or Firefox.

Back and Forward

The browser allows user to navigate between previous websites by using a back or a forward button. The functionality of these buttons were already included in the code and assets of the browser. To integrate this feature with gaze tracking, the necessary button-components were applied.

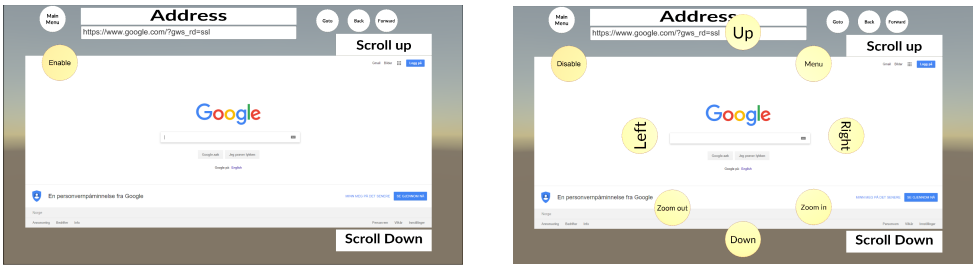
Zooming

To implement a zoom function, two options were considered. The browser has values for size of content within the displayed window. Adjusting these would create an effect similar to zoom. The browser would need to load when the values changed, and this could potentially cause a drop in performance.

Another alternative was to move the position of the users camera closer or further away from the browser, but this meant that another set of buttons for zooming in and out, as well as moving the camera in various directions. These buttons would add to the "clutter" on the screen, and the user could lose view of the buttons for the browser interaction, such as scroll.

The solution was to create an extra UI overlay that followed the camera. This overlay contained buttons for zooming, as well as moving the camera by changing the position values with the script CircularMotionDirection.cs. The script moves the camera based on an enumerator and a switch. This allows the user to zoom in and out, and pan the camera to view different areas of the browser.

The layout of the browser implementation can be seen in figure 14a. The browser is centered, with the secondary functions located more towards the edges of the screen. In figure 14b the zoom canvas has been enabled, showing the options for the user.



(a) The UI of the implemented browser.

(b) The Zoom UI of the browser enabled.

Figure 14: The UI of the browser, with zoom disabled, then enabled.

6.4 Keyboard

The keyboard is a central aspect of our software, as a lot of the planned functionality rely upon a functional keyboard. As mentioned in Visit to Sunnaas Sykehus avoiding the extremities of the screen can help reduce eye-strain. Utilizing a circular keyboard allows for having a short travel distance to every key, as opposed to having a standard layout on the keyboard.

Listing 6.4: Keyboard button placement

```
1       //instantiates keyboard buttons in a circle around the
            centre of the screen
```

```
2       for (int j = 0; j < keyboardStrings[i].Length; j++)
3       {
4         angle = j * Mathf.PI * 2 / keyboardStrings[i].Length
              ;
5         position.x = Mathf.Cos(angle) * distanceFromCenter;
6         position.y = Mathf.Sin(angle) * distanceFromCenter;
7         position.z = 0;
8
9         temp = Instantiate(keyboardButton, keyboards[i].
              transform, false).GetComponent<KeyboardButton>();
10        position += gameObject.transform.position;
11        temp.transform.position = position;
12        temp.Initialize(keyboardStrings[i][j]);
13      }
```

The listing 6.4 shows how the different keyboard buttons are placed within the keyboard canvas. This method uses basic trigonometry to instantiate the keyboard buttons in a circular form. Easily customizable keyboard size is a very positive attribute gained from using an algorithm such as the one above, as it allows for changing the size of the keyboard only by editing the radius. This way a user can enjoy more customization options pertaining to their own layout.

6.5 Bejeweled

This section will look at parts of the implementation for Bejeweled, highlighting important parts of the implementation.

Th choice to implement the Bejeweled as one controller class was done to make the logic simpler. The most prominent example of this is how pattern checks and gaze tracking checks has been done, which is discussed later in this section.

6.5.1 Initialization

At start up the Bejeweled game will initialize all variables, and prepare the game board for playing.

The Initialization of the game board is handled by the MapCreate() function in listing 6.5. This will create the GameObjects for the game board, including the games border and the jewels. At line 16 in listing 6.5 the GazeAware-component which allows for gaze tracking is added.

To include the Light object that is used for showing where the user is looking during run time, we run a secondary function that creates the required Light-object. This is then attached to the main Jewel object, with a local transform moving them slightly towards the camera, to let them shine on the jewels they are attached to.

When creating the jewels, two main ways of instantiating the objects could have been used. The first one is the dynamic attachment of components to the game object. The other is to have the game object blueprint stored as a Unity prefab, and instantiate copies of that object. For this project, the first option was chosen. This is to easily change the jewels during development, and made it quick to test new changes.

By having the jewels stored in one 2D array, they could be iterated over, which is something that is done for both pattern checking and gaze interaction checking. This

made the code for these checks easier to implement. While iterating over the same items multiple times takes up time on the CPU, the program is so small that this has next to effect on performance

Listing 6.5: MapCreate function

```
1     void MapCreate()
2     {
3         for (int i = 0; i < BoardSize; i++)
4         {
5             for (int j = 0; j < BoardSize; j++)
6             {
7                 if (i < EdgeSize || j < EdgeSize || j >
                      BoardSize - EdgeSize - 1 || i > BoardSize
                       - EdgeSize - 1)
8                 {
9                     jewels[i, j] = GameObject.
                          CreatePrimitive(PrimitiveType.Cube);
10                    jewels[i, j].GetComponent<Renderer>().
                          material.color = Color.grey;
11                }
12                else
13                {
14                    jewels[i, j] = GameObject.
                          CreatePrimitive(PrimitiveType.Sphere)
                          ;
15                    jewels[i, j].GetComponent<Renderer>().
                          material.color = ColorSelector();
16                    jewels[i, j].AddComponent<GazeAware>();
17                    jewels[i, j].GetComponent<Transform>().
                          localScale = new Vector3(0.8f, 0.8f,
                          0.8f);
18                    GameObject light = LightInit();
19                    light.transform.parent = jewels[i, j].
                          transform;
20                    light.transform.localPosition = new
                          Vector3(0, 0, -1f);
21                }
22
23                float xPos = -5.5f + i * 1;
24                float yPos = 5.5f - j * 1;
25                jewels[i, j].transform.position = new
                      Vector3(xPos, yPos, 0);
26
27
28            }
29        }
30        PatternCheck();
31        mapCreated = true;
32    }
```

Both checks for gaze and mouse input selection is run in the update loop (listing . This allows for both kinds of input at the same time, and also allows for the system to function with both mouse and gaze tracker, independently of each other. When the game

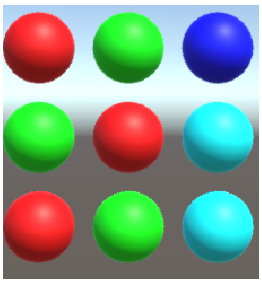
is paused, it will simply skip the entire update loop, and wait for the game to become unpaused again. Each iteration will reset the point multiplier to 1, run the selection checks, and then, if needed, move the selected jewel.

Listing 6.6: Bejeweled Update Loop

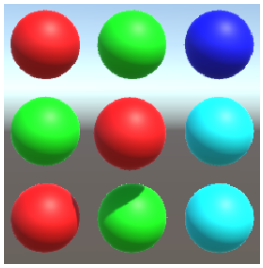
```
1     private void FixedUpdate()
2     {
3         if (playing)
4         {
5             pointMultiplier = 1;
6
7             SelectJewel();
8             SelectJewelGaze();
9
10            MoveJewel();
11            oldSelectedI = selectedI;
12            oldSelectedJ = selectedJ;
13        }
14    }
```

To ensure users knows which jewel they have selected, they need to be highlighted in some form. There are many different ways of doing this, one could make the jewel glow, give it a outline, change the color, to name a few. Our chosen method is to move the jewel in the game's world space. This is to give the small and important feedback of which jewel is currently selected.

When selecting jewels, they move forward in world space to signal that they are the one currently selected. There is no animation added, so they will simply pop forward, and when moved, pop to their new position. Since movement is restricted to moves that the game considers legal(see figure 8, jewels will only move when paired in such a way that at least one pattern in created. Figure 15 shows the way the center jewel is highlighted. While this isn't that apparent in the figures, when shown in the its full size, it is more noticeable.



(a) No jewel is selected



(b) The center jewel is selected

Figure 15: Jewel highlight mechanic

Upon interaction with the gaze tracker, the jewels will get a white dot on them to signal that this is what the user is looking at. This dot is made by shining a spotlight directly at the jewels. This is done in the SelectJewelGaze() function. See listing 6.7. This will loop through all jewels, and shine a light on the one being looked at, and turning off the light on all others. It will also select and highlight a jewel if it is looked at

long enough.

There were two main options presented when implementing gaze tracking. The first was to use a pointer similar to the one used for the main menu and the browser. Attaching a gaze aware component to the jewels was the other one. Both options would make the gaze interaction possible. The latter option was chosen as the preferred method. This decision was made in order to reduce the amount of objects on screen. Additionally, it was the quickest way to implement it.

Selection of jewels through gaze tracking happens in the same way as buttons. When the game object has gaze focus, it will increase a timer towards the click threshold(listing 6.7 line 15). For the jewels that does not have gaze focus, it will turn off the gaze feedback light, and reset the timer. While the reset for each jewel is resetting the jewels more than necessary, it also ensures that variables have the right value

Listing 6.7: Bejeweled Gaze Interaction

```
1   void SelectJewelGaze()
2   {
3       for(int i = EdgeSize; i < BoardSize - EdgeSize; i++)
4       {
5           for (int j = EdgeSize; j < BoardSize - EdgeSize;
                j++)
6
7             if(jewels[i, j].GetComponent<GazeAware>().
                  HasGazeFocus)
8             {
9                   jewels[i, j].GetComponentInChildren<
                        Light>().intensity = 10;
10                  if (LastLookedAt.x == i && LastLookedAt.
                        y == j)
11                  {
12
13                      lookTimer += Time.deltaTime;
14                      Debug.Log(LastLookedAt);
15                      if(lookTimer >= clickTimeThreshold)
16                      {
17                          selectedI = i;
18                          selectedJ = j;
19
20                          HighLightJewel(selectedI,
                                selectedJ);
21                          DeHighLightJewel();
22                      }
23                  }
24                  else
25                  {
26
27                      lookTimer = 0;
28                      LastLookedAt.x = i;
29                      LastLookedAt.y = j;
30                  }
31            }
32            else
33            {
```

```
34                      jewels[i, j].GetComponentInChildren<
                            Light>().intensity = 0;
35                  }
36
37              }
38          }
```

6.5.2 Pattern checks

The jewels are implemented so that they are instantiated during run time as GameObjects, having the necessary. Using the Unity3D Cube and Unity3D Spheres to differentiate between the border jewels and the game board jewels. Then, to enhance this difference further, the Color value of the GameObject's Renderer's Material (see listing 6.8 for variable structure) component is changed to a different color. For the border, it is grey, for the rest, it uses the original colors from Bejeweled, except white, which is changed to black. More info on color choice can be found in 4.3.2 Color Choice, and the change to black in 9.3 Bejeweled.

Listing 6.8: Structure of GameObject color change

```
1                       jewels[i, j].GetComponent<Renderer>().
                            material.color = ColorSelector();
```

6.5.3 Possible race condition

Since both the gaze tracking and the mouse input is checked during each iteration of the loop, there is a chance that these inputs might be interfering with each other. As shown in code listing 6.6, the mouse selection will always run before the gaze selection. If there by any chance were to be registered input from both the mouse and the gaze tracker that results in the selection of a jewel, the gaze input will take precedence, as it is handled later in the update loop.

As a result of the sequence of the logic, the gaze input takes priority. In the edge case that a user gives both gaze selection input and mouse selection input at the same time, the mouse selection input will be ignored. If these inputs are different, it will result in the drop of one of the input methods. This could leave users frustrated with the game. However, this happening is deemed highly unlikely, because of the way gaze input is handled, and the rate at which the game updates.

6.6 Chat

The implementation of the chat client is done in two parts, the server side and the client side. The server side was implemented using the language GO, whilst the client side is written in C#.

6.6.1 Client

The chat client handles translating JSON objects to and from structs. The JSON objects the chat client generates are passed onto the Network Manager, in order to to Post a request to the web server. The network manager then passes this data onto the web server, and awaits an answer, which is then sent back to the chat client. Depending on the type of response the client receives, it will update the UI accordingly.

Listing 6.9: Post to Server

```
1   /// <summary>
2   /// Return page information with json request. Takes in
        URL to send the request to, and returns the result as a
         page.
3   /// </summary>
4   /// <param name="postURL">The URL to Poll.</param>
5   /// <param name="jsonString">Information to send. (use
        JsonUtility.ToJson(string)) to generate this easily</
        param>
6   /// <returns>Page information from request (UnityEngine.
        WWW)</returns>
7   public WWW Post(string postURL, string jsonString)
8   {
9     WWW www;
10    var formData = System.Text.Encoding.UTF8.GetBytes(
          jsonString);
11    www = new WWW(postURL, formData);
12    StartCoroutine(WaitForRequest(www));
13    return www;
14  }
15
16  /// <summary>
17  /// Waits for the www download to complete, and prints
        debug information
18  /// </summary>
19  /// <param name="data">the UnityEngine.WWW to be
        downloaded</param>
20  /// <returns>Enumerator for the coroutine</returns>
21  IEnumerator WaitForRequest(WWW data)
22  {
23    yield return data; // Wait until the download is done
24  }
```

The code featured in 6.9 features the process of which the Network manager interacts with the web server. The first function takes in the URL or IP to post to, as well as the JSON string to send. It stores the data from the web request in a UnityEngine.WWW variable which can hold the data received from a web request. Once a request is made the second function is started as a co-routine which awaits a response from the server. If a response is not received from the server, it will still complete, however the WWW object will only contain the error. When a request is received it will allow the first function to return the WWW object. On a successful request the text attribute of the WWW object contains the JSON string with the data the server has returned.

6.6.2 Server

Using GOLANG to implement the RESTful API on the server side allowed for refactoring of code from previous projects. This allowed for faster internal deployment of the chat server, which in turn allowed for more time spent on implementing other parts of the project. In addition, the creation of RESTful APIs is something that we have previous experience in doing with GOLANG.

Main

Handling inputs to the server is done through the http.HandleFunc functions. These sets up the ways that the server is to handle the different requests that can be sent to it. While the server is running (listing 6.10) , it is running the http.ListenAndServe function, allowing it to wait for input, before choosing the right handler to run. The first argument in http.ListenAndServe gives the server the specified port to listen to when running. In the handler functions, the first argument is which trailing slash is to be used for the API, while the second argument is which handler function to run when invoked.

Listing 6.10: Server Main

```
1 func main() {
2   http.HandleFunc("/send/", HandlerSendMessage)
3   http.HandleFunc("/get/", HandlerGetMessage)
4   http.HandleFunc("/update/", HandlerUpdate)
5   http.ListenAndServe(":5050", nil)
6 }
```

The first handler, HandlerGetMessage, retrieves single messages from the server. It was implemented to test web communication, and follows the same design principle as all other handlers when checking the input it receives. The handler, shown in listing 6.11, uses the function PareMessageRequestInput from listing 6.12, to parse out information from the request JSON that is sent together with the POST http Method used to invoke the server. The last part of the input checking is to check if the fields in the incoming JSON are what they are supposed to be (listing 6.13).

Listing 6.11: HandlerGetMessage

```
1  //HandlerGetMessage handles all requests for specific
       messages
2  func HandlerGetMessage(w http.ResponseWriter, r *http.
       Request) {
3    fmt.Println("/get/ invoked")
4    http.Header.Add(w.Header(), "content-type", "application/
       json")
5    if r.Method != http.MethodPost {
6      http.Error(w, http.StatusText(http.StatusBadRequest),
         http.StatusBadRequest)
7    } else {
8      messageRequest, ok := ParseMessageRequestInput(r)
9      if ok {
10       message, ok := MessageGet(messageRequest)
11       if ok {
12         json.NewEncoder(w).Encode(message)
13         fmt.Println("Message sent to user")
14       } else {
15         fmt.Println("Message not found")
16         http.Error(w, http.StatusText(http.
             StatusInternalServerError), http.
             StatusInternalServerError)
17       }
18     } else {
19       http.Error(w, http.StatusText(http.
```

```
            StatusInternalServerError), http.
            StatusInternalServerError)
20      }
21    }
22  }
```

Listing 6.12: ParseMessageRequestInput

```
1  //ParseMessageRequestInput takes param r(http.Request
       pointer), returns MessageRequest from JSON struct in r
2  func ParseMessageRequestInput(r *http.Request) (
       messageRequest MessageRequest, allWentWell bool) {
3    var tempMessageRequest map[string]interface{}
4    err := json.NewDecoder(r.Body).Decode(&tempMessageRequest)
5    if err != nil {
6      allWentWell = false
7      fmt.Println("MessageRequest␣not␣decoded")
8    } else {
9      allWentWell = CheckMessageRequestStruct(
           tempMessageRequest)
10     if allWentWell {
11       messageRequest.ToUser, _ = tempMessageRequest["toUser"
             ].(string)
12       messageRequest.MessageID = int(tempMessageRequest["
             messageId"].(float64))
13     }
14   }
15   return
16 }
```

Listing 6.13: CheckMessageRequestInput

```
1  //CheckMessageRequestStruct takes param tempMessageRequest(
       map[string]interface{}), returns true if contents match
       specifications
2  func CheckMessageRequestStruct(tempMessageRequest map[string
       ]interface{}) (isMatch bool) {
3    if len(tempMessageRequest) == 2 {
4      _, to := tempMessageRequest["toUser"]
5      _, messageID := tempMessageRequest["messageId"]
6      if to == messageID && to == true {
7        isMatch = true
8      } else {
9        isMatch = false
10       fmt.Println("MessageRequest␣not␣matching")
11     }
12   } else {
13     isMatch = false
14     fmt.Println("MessageRequest␣not␣correct␣lenght")
15   }
16   return
17 }
```

For sending messages from one user to another, listing 6.14 HandlerSendMessage will

be invoked. This will receive input, parse it in the same vein as HandlerGetMessage, and then store it the right database.

Listing 6.14: HandlerSendMessage

```
1  //HandlerSendMessage handles all messages sent to server
2  func HandlerSendMessage(w http.ResponseWriter, r *http.
       Request) {
3    http.Header.Add(w.Header(), "content-type", "application/
       json")
4    fmt.Println("/sent/ invoked")
5    if r.Method != http.MethodPost {
6      http.Error(w, "400", 400)
7    } else {
8      message, ok := ParseMessageInput(r)
9      fmt.Println(message)
10     if ok {
11       ok = MessageRecieved(message)
12       fmt.Println("message sent")
13       if !ok {
14         http.Error(w, http.StatusText(http.
             StatusInternalServerError), http.
             StatusInternalServerError)
15       }
16     } else {
17       http.Error(w, "400", 400)
18     }
19   }
20 }
```

The primary function that the server has is the storing and re-sending messages that are sent by it's users. As the server is implemented as a RESTful API, it will only send updates back to the client whenever the client requests them. This is done by invoking HandlerUpdate(listing 6.15). This will then parse the request, and return a response containing all online users and all new messages the requesting users has required.

Implementing the server as a RESTful API was done a a easy way to create persistent chat logs. It also reduces the amount of work the server does, as it only executes code when it receives a request. However, this can result in a slower response rate for the chat clients, as they only receives messages when they ask for updates.

Listing 6.15: HandlerUpdate

```
1  func HandlerUpdate(w http.ResponseWriter, r *http.Request) {
2    fmt.Println("/update/ invoked")
3    http.Header.Add(w.Header(), "content-type", "application/
       json")
4    if r.Method != http.MethodPost {
5      http.Error(w, http.StatusText(http.StatusBadRequest),
         http.StatusBadRequest)
6    } else {
7      updateRequest, ok := ParseUpdateInput(r)
8      //fmt.Println(updateRequest)
9      if !ok {
```

```
10        http.Error(w, http.StatusText(http.StatusBadRequest),
              http.StatusBadRequest)
11      } else {
12        db := SetUpDB()
13        inDB := VerifyUser(updateRequest.UserName, db)
14        if !inDB {
15          if updateRequest.StatusOnline {
16            createErr := CreateNewUser(updateRequest)
17            if createErr != nil {
18              http.Error(w, http.StatusText(http.
                    StatusInternalServerError), http.
                    StatusInternalServerError)
19            }
20          } else {
21            http.Error(w, http.StatusText(http.
                  StatusBadRequest), http.StatusBadRequest)
22          }
23        } else {
24          user, _ := db.GetUser(updateRequest.UserName)
25          UpdateOnline(user, updateRequest.StatusOnline, db)
26          if updateRequest.StatusOnline {
27            updateResponse := UpdateResponse{}
28            updateResponse.UserName = updateRequest.UserName
29            updateResponse.AmountOfMessages = db.
                  CountMessageInUser(updateRequest.UserName)
30            updateResponse.OnlineUsers, _ = GetOnlineUsers(db)
31            updateResponse.Messages = GetMessages(
                  updateRequest, db)
32            fmt.Println(updateResponse)
33            json.NewEncoder(w).Encode(updateResponse)
34          }
35        }
36      }
37    }
38  }
```

One thing worth noting is that GOLANG allows for multiple return statements from functions, and therefor most of the function similar to the one in listing 6.13 would be redundant. This was originally done to make the logic more readable, however, the argument can be made that it only increases clutter. Changing this to only checking values once is discussed further in Further Development(p.60).

Database

When implementing the server specific database API, it was implemented using test driven development. The basic tests were written to make sure the MongoDB API worked the correct way. The tests made sure that the basic functionality worked for both data structures required. While the test were useful during implementation, they will not be included here.

The database stores two different JSON objects. The first of the two stores info on the username, id, status, and last request for an update. The struct is shown in full in listing 6.16. The second contains the information needed to send a message from one user to

another(see listing 6.17).

Listing 6.16: User

```
//User struct
type User struct {
  UserName   string `json:"userName"`
  Online     bool   `json:"online"`
  LastOnline int64  `json:"lastOnline"`
  UserID     int    `json:"userId"`
}
```

Listing 6.17: Message

```
//Message struct
type Message struct {
  FromUser  string `json:"fromUser"`
  ToUser    string `json:"toUser"`
  TimeStamp int64  `json:"timeStamp"`
  Message   string `json:"message"`
  MessageID int    `json:"messageID"`
}
```

All functionality for the database was written using the mgo.v2 GOLANG package. Subsequently, all the database functions are attached to a database struct, as GOLANG doesn't use classes, to store the extra info required for the database to run. The main struct for the APIMongoDB contains information on which database to call, and how information is stored in it. Then the following code example for retrieving online users (listing 6.19) shows how the function is attached to the struct, and how it calls the database.

Listing 6.18: APIMongoDB

```
import (
  "fmt"
  "gopkg.in/mgo.v2"
  "gopkg.in/mgo.v2/bson"
)

//APIMongoDB stores the details of the DB connection.
type APIMongoDB struct {
  Host                  string
  DatabaseName          string
  MessageCollectionName string //_messages
  UserCollectionName    string
}
```

Listing 6.19: GetOnlineUsers

```
//GetOnlineUsers returns User slice containing all users
    where onlie equals true
func (db *APIMongoDB) GetOnlineUsers() ([]User, bool) {
  session, err := mgo.Dial(db.Host)
  if err != nil {
    panic(err)
```

```
6     }
7     online := true
8     defer session.Close()
9     user := []User{}
10    allIsWell := true
11    errFind := session.DB(db.DatabaseName).C(db.
          UserCollectionName).Find(bson.M{"online": online}).All
          (&user)
12    if errFind != nil {
13      allIsWell = false
14    }
15    return user, allIsWell
16  }
```

Communication

As discussed in 3.5.1, all communication between the client and the server use JSON objects to store the information sent. There are three primary JSON objects used for communication, one for update request, one for update responses, and one for message sending. The latter one being a modified version of the Message struct, containing all the info, except the time stamp and message id.

Listing 6.20: UpdateRequest

```
1 //UpdateRequest struct
2 type UpdateRequest struct {
3   UserName          string  `json:"userName"`
4   StatusOnline      bool    `json:"statusOnline"`
5   AmountOfMessages  int     `json:"amountOfMessages"`
6 }
```

Listing 6.21: UpdateResponse

```
1 //UpdateResponse struct
2 type UpdateResponse struct {
3   UserName          string     `json:"userName"`
4   AmountOfMessages  int        `json:"amountOfMessages"`
5   OnlineUsers       []string   `json:"onlineUsers"`
6   Messages          []Message  `json:"message"`
7 }
```

7 Deployment

7.1 Software Installation

A zip-folder with the software is available at https://bitbucket.org/bgwendling/bachelor_project_eyevr/downloads/.

This folder includes an executable that is used to run the software. To do this, the zip-folder must be unpacked into a new folder. The executable should then be able to run.

7.2 Server setup

The necessary files for the server can be found at https://bitbucket.org/bgwendling/bachelor_project_eyevr/downloads/. To run the server the files main.go and database.go are required. Then, make sure that an instance of MongoDB is running either locally or remotely. If this is run remotely, change the values of the variables in SetUpDB() in main.go to the new location. Then execute main.go together with database.go.

8 User Testing and Feedback

In this chapter we discuss the results from the testing of the software.

8.1 Summary

For the first round of testing, we held a preliminary test round, which was used as a basis for the first full test round. The preliminary test was held to identify and remove bugs that would affect the testing, and also to identify any obvious problems that might occur at a later point in time. Then, as the results from the preliminary testing was addressed, the first full round started. During this test round the focus was on how the users interacted with our software, and their feedback on the software.

From this round of testing, we gathered data on how users used the program, what their thoughts were on the UI layout for our web browser, and how they felt interacting with it, and with our implementations of bejeweled. The data was gathered by using questionnaires, and through discussions with the testers during their testing. Furthermore, all data was collected without any personal data, to prevent any issues regarding regulations on storage of personal data. Instead, we have asked our testers to tell us their age and gender, on a volunteer basis, to be able to build a demographic for our testers. This information was recorded separately from our questionnaires and notes, as to not make us able to connect the specific user to testing.

The initial data we wanted to gather was on how our software was to use, and how the interactions with the computer was compared to using regular mouse and keyboard input. We also wanted to get feedback on the interfaces that we had created for our web browser, and for our implementation of Bejeweled. Most of this data would be gathered through the interviews done with our test subjects whilst testing, with our questionnaires to be used as supplementary data to further back up their answers.

8.1.1 Extra Precautions

By using interviews with our test subjects whilst they are testing, we could run the risk of encountering the Hawthorne effect[30]. This could potentially lead us getting results, that are not a reaction to the thing that our testers are testing, but a reaction to the fact that they are being observed whilst testing.

There is also the possibility that we encounter results that stem from our users suddenly being aware of how they use a screen, and how they look at browser. This is something that we found examples of, and it will be discussed in further detail in the test results discussion.

8.1.2 Data Gathering

The first round of questions established data on the test demographic. This included data on testers use of screens, if they have any experience with software development, or if thought there was anything that would hinder the testers ability to use our gaze tracking system. At the same time, we asked our testers to leave us their age and gender, so that

we could have a slight demographic overview.

Since a description of what people thought about the system being tested was required for the test, a Likert scale was used in the second questionnaire. The scale used was a 7 item scale, to best convey the thoughts of our users.

There are ethical and legal limitations when testing systems. Especially pertaining people within "vulnerable groups" [31]. As a result of this, all test subject have been fellow students. This does imply that not all of our test data will be relevant for the core user group that the project focuses on, however, the tests will still be able to give us a better understanding of the challenges that such a system might face.

We have also consulted with Sunnaas Sykehus on their experiences on using systems that utilizes gaze tracking technology, and have taken their feedback into consideration when looking at our test results.

8.2 Results

8.2.1 User base

There were eleven testers in the first round of testing. Among these, most spend a lot of time using digital media and digital devices in their everyday life. All testers spend more than 4 hours a day using a digital device, with almost half of them spending more than 10 hours using devices with screens a day. Most of the testers are male, and the age ranges from 21 to 25. Figure 16 and table 3 show the age and gender distribution of the testers. Tables 4, 5, 6, 7, and 8 show result from the first questionnaire. Table 9 show the results from the second questionnaire. For the test questions, see appendix E.

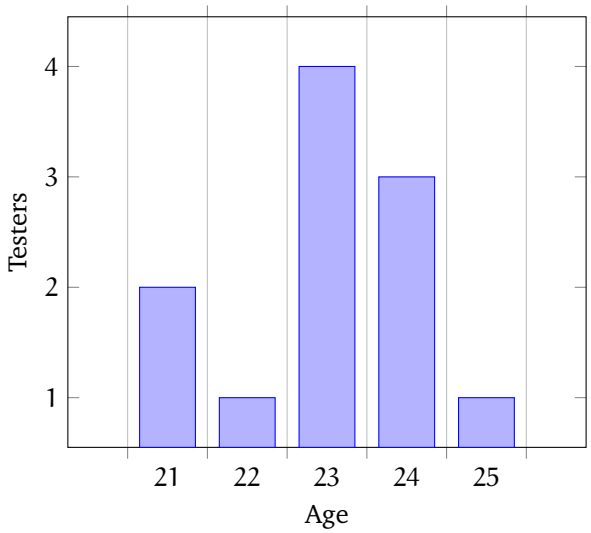


Figure 16: Age distribution of testers

Table 3: Gender distribution

Gender	Amount
Men	8
Women	3

Table 4: Earlier use of gaze tracking

Options	Amount
Yes	3
No	8

Table 5: Hours spent with a screen per day

Hours	Amount
10+	5
7-9	4
4-6	2

Table 6: Most used digital system

System	Amount
Mobile/Cellphone	5
Laptop	4
Desktop	2

Table 7: Might have trouble using gaze tracking

System	Amount
Yes	0
Maybe	1
No	10

Table 8: Experience with software development

System	Amount
Yes	7
No	4

Table 9: After test questionnaire

Rank	**Question 1**	**Question 2**	**Question 3**	**Question 4**
1	N/A	N/A	1	3
2	3	5	1	1
3	N/A	1	3	1
4	1	N/A	1	2
5	3	1	3	2
6	3	2	1	1
7	N/A	N/A	N/A	N/A

8.2.2 Interview results

This section will cover the results gathered from the interviews that were conducted.

Bejeweled

Because the testing was divided into two parts, one on testing Bejeweled, and one on using a web-browser, the interviews conducted reflect that. They were conducted in such a way that most test subjects gave feedback on first one aspect, and then the other, with a few wanting to swap between the two to give more feedback, retest something, or testing things twice to give better feedback.

For Bejeweled, we've gathered that most users found the gaze tracker too inaccurate to be used successfully. This is further enhanced by the fact that the jewels were too close to each other to accurately pinpoint which one was looked at. We also encountered problems with the edges of the game board, where the inaccuracy was considerably worse.

The two types of feedback that the system gave to the player got mostly positive results. Most testers found that having the selected jewel move towards them in game space helped them track what they had selected. The Light game object that was attached to each jewel, lighting up when looked upon, also got positive feedback, as it allowed players to know which one they were looking at. Although the feedback types from the system got positive results form our testers, most also wanted the highlight of the selected jewel to be less than what it was in the first iteration, as the new position of the jewel could interfere with the selection process. In addition, the use of a white light to show the current looked at jewel was not clear against the white jewels on the board.

There was also feedback on that the system wasn't responsive enough. Having the gaze tracker as the input for both position and selection on the screen, it became apparent that the selection took too long to what most users was comfortable with. When then asked the question on if they would prefer to have a separate clicker device, most users again answered that this would be a preferred option, with using regular mouse and keyboard as the most preferred one.

Web browser

For the web browser, we wanted to test how our navigation system and general use of the browser worked. This had users interacting with a web page, for most tests, this was `https://www.nrk.no/`. For non Norwegian testers, `https://www.bbc.co.uk/`. The purpose of using a news site as a page was to see how it was to navigate and interact with, using only gaze tracking as input. It would also serve as a test of how navigating an environment filled with interact-able links.

Most testers reported that the use of the gaze tracker to look around, and move a cursor felt comfortable, and that it was easy too look at items in the web page. However, when it came to selecting and interacting with links, the dwell time became too long.

The UI that was built around the browser, to allow for zooming and extra navigation within the browser got feedback that the UI elements were too close together, and that they sometimes overlapped each other. There were also negativity regarding that the UI surrounding the browser worked without the cursor that the browser used for gaze tracking input.

Since scrolling was implemented as jumps in the page, rather than as a continuous scrolling, there were some users who reported that it wasn't intuitive. The preferred way to use gaze input scrolling would be to have it scroll slowly upwards or downwards when

looking at the respective edge of the browser window.

Regarding the cursor that follows the users gaze, there were only one user having any negative feedback. This feedback was not on how the cursor behaved, but rather the colour of the cursor.

With the browser, as with Bejeweled, the use of the gaze tracker both as the input for where to look and to give selection input was not responsive enough.

Main menu

There was not much feedback regarding the main menu for the system, although what was gathered was an expansion upon the results from our tests of the web browser. The feedback here was primarily concerning the speed at which clicks were registered. In addition, some users wanted more feedback when closing the software. The primary concerns for our testers was that the way that they were given feedback from the system was not enough for them to use it properly.

Other findings

Due to the way we conducted the interviews, being short questions about what our users thought about the system, and also encouraging them to think loudly, we got some feedback that we consider to be on the side of our intended test. This has mostly been on how people themselves use browsers, and where they look at the screen, but it has also included some testers thoughts on the uses of gaze tracking technology.

One user reported that having a dot follow along the text that the tester was reading, it became easier for the tester too read. The same tester also thought that this could, based on the testers own challenges and experiences, that the feedback form the gaze tracking could help people with dyslexia read.

There was also one tester that reported that the gaze tracker didn't work properly, as the tester has trouble with one eye suddenly moving. This was something that the tester didn't think about in regards to question 4 in the pre-test questionnaire. As a response to the tester realizing this during the software testing, the tester asked to change the answer given to the tester's original question 4 answer.

8.2.3 Post Test Questionnaire

Likert scale questions

After the interviews and testing, the testers were given a questionnaire with questions on the use of the system. The answers given to these questions will give an indication on the possibility of having users for the software that has been developed as part of this project.

The test results shows a distribution of opinions among the testers. Because of this, there are no clear conclusions that can be made. However, on question 2(It was hard to use the software), the majority of testers have reported that they somewhat disagree or disagree with the statement.

From the result of this, there are a few conclusions that can be made, although they are more on the possibility with the technology, than the way the software worked. For further discussions regarding the test results, see 9 and 9.7

Detailed answers

Accompanying the second questionnaire was a set of extra questions where the testers could fill in their own answers and assessment of the system. These answers were varied, and differing from each other, giving many inputs on how to develop the software further.

There were three extra questions, these were

- Is there any activities that you think might work better with gaze tracking?
- Why would you / would you not use the software on a daily basis?
- Other comments

Multiple testers answered that the use of gaze tracking could be useful when browsing the net. Others wrote that for browsing through graphical file systems, it could be useful as an interaction tool. Another answer was that it could be helpful when aiming in first person shooter games.

The answers to the second question varied a bit more than the first one. For example, there are answers that state that this system in particular is not well enough developed. Others reported that the precision was off, and that they couldn't control the cursor well enough.

While most answers for the second question were on why the users would not use the system. There were some who would want to use this system for when their fingers were dirty.

There was one who wrote that the tester did not want use the system at all. The reason for this was that the tester in question had the full use of their hands.

There was not many people answering the third question. Less than half the testers responded. No two answers were the same. One thought that this system could be fun for the kids. One user would prefer a headrest when using this system, and two users reported that they felt tired in their eyes after testing.

9 Discussion

In this chapter, we will discuss the results of the project. Additionally we will discuss and evaluate the development process, the testing process, and the use of an eye tracker.

9.1 Results

What initially started as project to develop a software platform for quadriplegics, ended up as a deep dive into developing gaze tracking software. While much of the software requirements and goals are the same, it definitively changed our focus during the development. The most significant change to our development, was to go from creating a platform for people with severe physical limitations, to focusing on researching and prototyping gaze tracking software.

Changing from a specific target to a more general approach was probably better for the overall goal of the project. When we learned more about the already existing solutions for the physically disabled, we realized that we did not have the time or skill to create something that could compete. We decided to work towards a more generalized solution, and create a prototype gaze tracking software. This allowed us to develop for a broader audience, and while our requirements became less defined, we could still use our initial plan while developing.

The final product is not as large as we planned it to be at the start of the project. We had a bigger scope for our project plan than we were able to finish, however this was accounted for. Our project was planned to be incremental, and as a result having a big scope is beneficial. While our prototype was not satisfactory when compared to our original plan, we are confident that the current software can be useful for further development.

Though the prototype was not satisfactory, the process behind making it was very educational. We have had the opportunity to delve into the development of gaze tracking software, integrating new and foreign software into our own development. In addition, the use of Tobii EyeX allowed us to experiment with UI, layouts, and universal design.

Working as a team has given us the insight into how we prefer to work as individuals, and how this transfers into a group. Through our supervisor we have gotten experience on how to mediate and handle problems within our group. This is something that we hope to bring with us into future work.

9.2 Software Implementation

9.2.1 Unity3D

The decision to use Unity3D as a game engine for development is something that proved very beneficial for us as a group. While it posed some challenges when combined with Git, it provided a good platform for developing modular software. The use of game objects and the component-system meant that functionality could be translated and adapted for different functionality. This meant that components we created could be adapted to new

uses, without having a significant impact on performance.

9.2.2 Gaze Tracking Implementation

Working with the Tobii SDK proved to be easier than expected, but how we approached it as a group could have been done better. We made the error of not gathering to learn the possible functionality and limitations of the Tobii Unity SDK. Because of this, we had occasionally made errors when developing that other members of the group knew the solution to. We later gathered and went over our own knowledge of the SDK, but having done it earlier could have saved us some time.

Having only one Tobii EyeX available was also a limiting factor in the very beginning of the development. This meant that we sometimes were not available to test the functionality we were implementing at that very moment. We do not believe that this really slowed down the process, but it has caused some errors that we later had to fix, because we had used incorrect numbers or method-calls.

9.2.3 Browser

Using an existing browser implementation still proved to be a challenge. It took some time to build the correct functions for gaze tracking interaction. Most functionality could be implemented with the use of a gaze tracker, and while writing text into the browser was not implemented we know that there are possible ways to do it.

As the tests showed, there are still some problems with the tracking of the gaze pointer within the browser. It seems to appear that the pointer either does not always represent the correct position, or that the fixation time is not set correctly. It is imaginable that this error would not have been made if we had built a browser for Unity ourselves, and tailored it more for a gaze tracker. Still, this would probably take more time, and the error can likely be fixed with some adjustment of the existing code.

Zoom

The decision to implement zoom by panning and moving the camera worked surprisingly well, but the UI should be done in a less intrusive way. While the option was there to hide it, making it smaller and putting it towards the edges of the screen will likely help. This would also make it more natural for users when reading. As they reach the edge of the screen, it will pan in the direction they are looking.

Having two separate UI canvases for zooming and interacting with the browser was not a good solution for handling the UI. In addition to functions going out of view when zooming, it creates a discrepancy in the position of functions as user pan the camera. Integrating the UI canvas of the browser buttons with the UI canvas of the zoom-functionality would make it more consistent.

Mouse Pointer

Using a small red dot as the pointer proved to be very beneficial, according to the testing. Compared to a previous iteration, which presented a dot within a circle, it was less obstructing while still being very visible to the user.

The pointer should give the user more feedback when they are trying to click. This would provide users a better experience when using the browser, and is something we should have considered more from the beginning. However, seeing how users reacted to

the lack of feedback when using it also gave some insight into the fact that feedback is expected, and likely important for many of the interactive functions.

9.2.4 Bejeweled

The greatest challenge while implementing Bejeweled proved to be concurrency issues surrounding checks for patterns and legal moves. This issue arose as a result of the sequence that the code was executed in. Another issue was how to implement gaze tracing. While the gaze pointer used in the main menu could have been a good way to show the user where they were looking, the selected method has less clutter.

9.2.5 Chat

The client and a server for chat was implemented at a late stage in development. This led to them being ready after the build that was used for testing was created. Therefore, these have not been tested with external testers.

For the server side, implementing the server as a RESTful API was probably not the best solution. While it was easy to set up, and quick to develop, it should most likely be scrapped in favor for a more robust and better system. In addition, the current setup only send messages once invoked, which makes receiving chat messages slow.

9.2.6 Keyboard

The implementation of the keyboard is good in multiple ways. Placement of the keyboard buttons is the best feature of this implementation specifically. This method of placing the keyboard button ensures that the buttons are equal distance apart from each other, and placed in a circle regardless of the amount of keyboard buttons a given keyboard requires. As a result implementing different keyboards with varying amount of keys is an effortless process.

In relation to the keyboard a significant amount of time was dedicated to making a custom input handler. However, whilst developing the input handler, there was a realization that a canvas has the ability to hold 3D objects, if set to exist in world space. Tobii natively support Gaze tracking of 3D objects, and as a result the input manager was scrapped in favour of Tobii's "GazeAware" component.

9.3 User Tests

The testing gave insight into what worked, and what did not work for the prototype. Changes were made to the software based on feedback from the testing. In addition, as the testing started late into the project, there were changes for the software that were proposed, but not implemented. It was also possible to use the feedback as a basis for our discussion of which types of games and software could be easy to integrate with gaze tracking as the primary input.

Bejeweled

As stated in 8.2.2, there were some conclusive results from testing Bejeweled. Primarily, users thought that it would be possible to play games close to Bejeweled using only gaze as interaction. This provides an indication of how a developer should design games with gaze tracking as the primary or solitary input method.

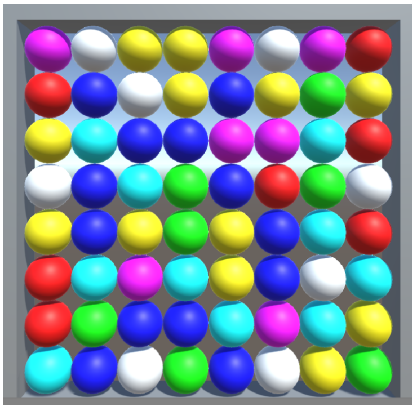
Given that most testers thought that the feedback from the game was enough to play

it, there were only minor adjustments that needed to be made. These adjustments apply to quality of life, and only changes the way the game looks.

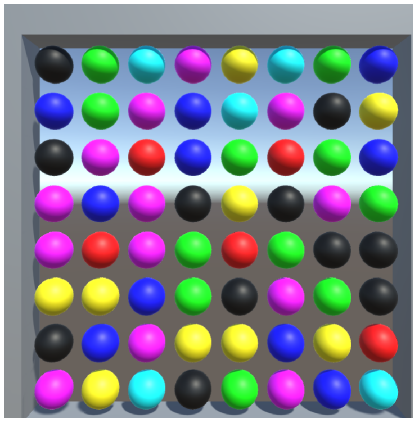
The first change was changing the white jewels to a different color. This is because the testers stated that the white light did not stand out against the white jewels. Due to the importance of feedback given to users, the color was changed to make the white light stand out more against them. The new color selected was black, as this would easily contrast against the white light. Figure 17 (p. 56) shows the changes between the old and new Bejeweled colors.

The second change was to decrease the size of the jewels in the game board. This change was added as a way to make it easier to distinguish between the jewels in the game. By changing the scale of the jewels to make them slightly smaller, more space between the jewels was created. An additional effect of this was that the jewels were easier to select when looked at, as they were further apart.

The third change to Bejeweled was to decrease the distance the highlighted jewels moved. Cutting the distance in half reduced the amount of visual overlap that the jewels had. Due to the nature of how they are selected, the decreased overlap made it easier to select individual jewels.



(a) Original Bejeweled colors.



(b) Changed Bejeweled colors.

Figure 17: Changes in Bejeweled.

All changes to Bejeweled were smaller changes achievable through changing variables in the script. This made them quick to implement. As a result of this, they were given a quick user test, to see if they had had any effect. From this test, we gathered that the new changes had their intended effect.

Browser

The changes to the browser, as a result of the feedback given from the testers were mostly changes to the UI layout. Due to time restrictions on the project, these changes were not possible to implement or test, and will therefore only be discussed as the changes intended to be done.

The first change will be how to scroll the web page. Having buttons on the top and bottom of the page that scrolled in jumps were not liked by most testers. To make this interaction better, the proposed change is to create a transparent UI element over the top

and bottom of the browser window. This will slowly scroll the page up or down, when the user looks at the respective position on the page.

The second change will be to change the position of the UI layout slightly. This is to prevent the UI from overlapping and blocking the browser view. Most users were okay with using the far edges of the 14" screen for button placement. It should be considered that this feedback could be different had we used a larger screen.

Using a dot on the screen as the cursor got positive feedback. Since it received mostly positive feedback, there were no need to change it. The only feedback it got that did differ was one user complaining about the color. Because of this, a option to change the color has been considered.

Other findings

In the process of gathering feedback from testers additional ideas, for gaze tracking, surfaced. The most common suggestion is one that would drastically change the experience of using the program, namely a clicker. Additionally the testers gave their ideas on how to use gaze tracking for everyday purposes.

Some of the suggestions are akin to the ways we have looked at using a gaze tracker. One suggestion is the navigation of a graphical file browser, another is the use of a system that allows for writing notes during lectures. The final interesting one is one who wants a system that allows the tester to interact with a computer while the testers hands were dirty, which they apparently were a lot.

One proposed way to increase the responsiveness was to include a separate way to send a "click" input to the software. The use of a separate click device, by having a separate button to click. If we were to add one for quadriplegics, a sensor to either bite or suck on could be used to send the input.

9.4 Group Dynamic

Another important part of working on this project was working together as a team. There has been good and bad sides to the way this has functioned.

9.4.1 Good

Splitting up the project into modules gave each member more independence when working. Most components could be done separately, with a few exceptions, so there was no need to wait for someone else to finish their components. This meant that when a member was done with a component, they could start working on another one. As a result less time was spent being unproductive.

Not having a very rigid list of requirements proved both positive and negative. It could be easy to lose focus because of the lack of feature specification. However, it allowed the group to implement and test features more flexibly. This gave us the opportunity to explore gaze tracking widely. Creating our own set of requirements proved valuable as a system development experience, where we had to find and discuss features ourselves.

9.4.2 Bad

Communication was a recurring issue for the group throughout the project. During the initial planning, the group met frequently to discuss, revise, and plan the devel-

opment. However, in the development period communication was often fragmented. It was planned to use Discord and Facebook frequently as communication tools, and while this was used, it was not used as intended.

Project status and current progress of each member was not expressed clearly enough, and tools like an issue tracker or a more defined sprint board should have been used more efficiently. Using tools to communicate our current goals and progress better could have helped the group be more efficient in the development period.

Another part of the planned way to work was by having planned meeting times. This was to create a set time for work, with the intention that when the time was over, we could go home and have the rest of the day off. Due to the flexible nature of development, the meeting times changed, and not all days required all members present. When the meeting times were considered important, we were not the best to adhere to them. The only meeting that was never missed by some of the members was the weekly scheduled meeting with our supervisor.

During planning and development of the project there were multiple tools the group decided upon using. However a portion of said tools were either abandoned after a while, or were never used by the group as a whole. In addition the use of git has not been used to the same extent by all members of the group. If the tools had been properly utilized by the group as a whole it is safe to assume that the project would have ended up in a different state compared to where it is today.

Trello is a great example of such a tool. Trello is a tool used for issue tracking, which we intended to utilize. In the end it was neglected. Had we used it effectively, we would have had a better total overview of the development progress. In turn, this could have led to better group cooperation, and a better end product.

The group had recurring communication issues during development. When discussing the results of the project, we concluded that lack of good communication is the key reason for the project not proceeding as planned.

In the more development-heavy periods of the project, each group member was usually confined to their own working station. The plan was then to continue communicating infrequently when needed, and to gather at the start and end of the work week. This meant that each member was available, but could focus on the separate components. However, part of the group sometimes failed to meet these expectations, and in turn, this lead to disagreement.

9.4.3 Solving Teamwork Issues

When these problems first started, we tried to solve them internally in the group. We did this by trying to open a dialogue, explaining to each other what we thought about the progress and problems so far. This often led to the group cooperating better for the coming week, but was not an effective long-term tool. In the end, we brought in our supervisor as a mediator, to openly discuss each members opinion on the group dynamic. This gave all members the opportunity to discuss in an arena that was considered neutral, and it became easier to find the underlying issues of the group, and start working on them.

9.5 Computer Interaction for Quadriplegics

As part of the project we looked into existing systems that were created with the intention to aid people with severe physical limitations. These are some of our findings:

9.5.1 Systems on the Market

Other computer interaction systems for quadriplegics include

Glassouse Head mounted input device, using bites as click input

Tobii Dynavox Augmentative and alternative communication. Made to be make communication easier.

Both these systems provide a way to interact with computers for people with reduced mobility. Only the Tobii systems use gaze tracking as input [32]. Glassouse uses head movements to control the cursor on a computer screen [33].

9.5.2 Possible Health benefits

By creating ways to communicate and interact with the world, we could potentially make everyday life better for people with severe physical disabilities. This could remedy the need for entertainment and social interactions.

In addition, there would be no way for other users to know which users are disabled. This could allow for users to interact with each other, without knowing each others physical limitations. In turn, this could allow for social interactions on a more equal footing.

9.6 Computer Vision Syndrome

9.6.1 Description

Computer Vision Syndrome, CVS for short, is a collection of symptoms related to the prolonged use of digital displays. This collection of symptoms include eyestrain, headaches and dry eyes, to name a few[34]. Over the recent years CVS has become more common, as the use of screens has become more widespread.

9.6.2 Project importance

Since the software uses gaze tracking as primary input, the eyes are already under strain. CVS can be contracted from normal use of digital displays. This becomes an extra concern when gaze tracking systems is an additional factor. As a result precautions must be made in order to reduce the amount of strain.

9.6.3 Test Results

When doing users test of the system, there were two users who reported eyestrain after use. Whether the reason for this is the use of a screen, or the use of gaze tracking as input, is not known. It could indicate that the use of gaze tracking increases the risk of contracting CVS symptoms. Given more time, this could be a full field of research itself.

9.6.4 Prevention

There are some ways to prevent the contraction of CVS. The American Optometric Association recommend following the "20-20-20" rule to help ease the symptoms of eyestrain

[34]. The 20-20-20 rule states that once every 20 minutes, look at something 20 feet away, for 20 seconds [35]. They also recommend positioning the screen so it is slightly below the viewer.

9.7 Further Development

For a project such as this there will be features that are left unimplemented, this project is no exception. There are a few things that have been left undone, some things that were originally planned to be implemented, and others ideas which came to fruition throughout the development process.

9.7.1 Bejeweled

For Bejeweled, the main change that should be implemented is the way jewels are created. Currently, they are dynamically assigned components during run time. One way to make this cleaner is to instantiate the jewels from a prefab containing all the components needed for the game to work. This would also decrease the lines of code in the Bejeweled initialization.

Chat Server

In the chat server there are uses of GOLANG's ability to get multiple values from functions. One of these is to get values from key value pairs. Both the parsing and checking functions for input uses this feature. However, this could be shortened to just one check, and then use the same results from the checking function to determine the return value. This would essentially eliminate unnecessary parts of the code.

9.7.2 Useful Features

Here are some of the features that would be a major improvement to the software.

9.7.3 Online Environment

A shortcoming of the program is the lack of a lobby system. Especially when considering that typing is done using gaze tracking only, having to type out usernames might be a strenuous process. To help with this, an online lobby system that can show who is online, as well as potentially a friend list would be useful. Both of these additions could then be tied in with the chat client so that selecting a recipient for a chat message would be as easy as just selecting someone from the online lobby.

Predictive Keyboard

A predictive keyboard would be an immense improvement to the quality of life for users using the software. Due to gaze tracking being the only form of input for the program, typing is something that takes a lot of time, and a lot of looking around (which can cause CVS). Having a predictive keyboard could reduce the time it takes to type, and how much a user has to use their eyes.

Chat server security

Currently, the chat server has little in terms of security. There is no unique user identification, and any user could use the same username. To improve security, the addition of a password protected username, and a log in sequence at the start of the program is beneficial.

In addition to the ability to log in, privacy is a key concern when creating chat services. To ensure this in future development, the password needs to be hashed, and salted, before being stored in the database. Another way to increase privacy would be encrypt users messages, so that they are only readable for the sender and receiver.

9.7.4 Gaze Tracking Controlled Games

We also looked into which game genres would be most suitable for gaze tracking as primary or only input based on what we have learned.

Visual Novel

The simple control scheme of visual novels makes for an easy and intuitive playing experience, even if gaze tracking is the only input. Visual novels are games that generally features very little gameplay, and is a lot more focused on story. A typical visual novel will have a user go through a story, and read dialogue. When a chance occurs for the player to make a choice, they are generally given a few different options represented with buttons. Button selection is something gaze tracking have proved to work well with, and as such there is reason to believe that visual novels would go hand in hand with gaze tracking quite seamlessly.

Point & Click

Point & Click games are games based around finding objects in an environment and clicking them in order to trigger scripted events. Much like visual novels the focus is on the story, and the only needed input is the ability to select something.

Turn Based Games

Turn based games allows one player to take their turn at a time. As such a user playing with gaze tracking would not be limited in their ability to play the game competitively. While using gaze tracking to play these games require more time, the fact that the gameplay is based on turns rather than speed makes it viable.

10 Conclusion

When the project started, we considered the use of gaze tracking as something that was mostly practical for quadriplegics and the physically disabled. With this project, we have seen that gaze tracking is something that most certainly can be used by the everyday user. Designing, implementing, and prototyping a software utilizing an eye tracker has shown us that it can be a useful tool in many different applications, such as reading, browsing, and playing games.

While we are not entirely satisfied with the state of the prototype, it has proved to be a great learning experience for us as programmers. In addition to this, the code written and the assets we have made and used is something we can bring with us when we further develop software for an eye tracker.

Creating a prototype as a team has proved to be a very challenging task for us. All the while we have learned to integrate gaze tracking, we have had to work and develop ourselves as team. Better understanding our own habits and patterns will help us meet future challenges with a better understanding of how we can handle them.

In the end, what we can take away from this project is that it has given us a chance to develop as members of a team, and programmers. It has been a fun experience to work with an input type we had no prior knowledge about, and to learn it from the very beginning.

Bibliography

[1] 2018. How eye tracking works. https://www.tobiidynavox.com/about/about-us/how-eye-tracking-works/. (Last visited May 2018).

[2] 2017. Games with tobii integration. https://tobiigaming.com/games/. (Visited May 2018).

[3] 2018. Tobii eye tracking: the next natural step in vr. https://www.tobii.com/tech/products/vr/. (Last visited May 2018).

[4] 2018. Tobii eye tracker 4c. https://tobiigaming.com/eye-tracker-4c/. (Visited May 2018).

[5] 2017. How to develop with eyex on ue4. https://developer.tobii.com/community/forums/topic/how-to-develop-with-eyex-on-ue4/. (Visited April 2018).

[6] 2018. Home - fove eye tracking virtual reality headset. https://www.getfove.com/. (Last visited May 2018).

[7] 2018. aglass. http://www.aglass.com/. (Last visited May 2018).

[8] 2018. License agreement for tobii core sdk and tobii gaming sdk, version 1.2. https://developer.tobii.com/license-agreement/#.

[9] Feit, A. M., Williams, S., Toledo, A., Paradiso, A., Kulkarni, H., Kane, S., & Morris, M. R. 2017. Toward everyday gaze input: Accuracy and precision of eye tracking and implications for design. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, 1118–1130, New York, NY, USA. ACM. URL: http://doi.acm.org/10.1145/3025453.3025599, doi:10.1145/3025453.3025599.

[10] 2018. Operating system market share. https://netmarketshare.com/operating-system-market-share.aspx. (Visited May 2018).

[11] 2018. Statcounter: Desktop operating system market share worldwide. http://gs.statcounter.com/os-market-share/desktop/worldwide. (Visited May 2018).

[12] 2016. Mac os support. https://developer.tobii.com/community/forums/topic/mac-os-support/. (Visited May 2018).

[13] Takahashi, D. 2017. Candy crush saga: 2.73 billion downloads in five years and still counting. (Last visited May 2018). URL: https://venturebeat.com/2018/05/09/nvidia-geforce-gtx-gpus-are-in-stock-at-their-original-price/.

[14] 2018. Tobii unitysdk releases. https://github.com/Tobii/UnitySDK/releases. (Visited May 2018).

[15] 2018. Unity download archive. https://unity3d.com/get-unity/download/archive. (Visited May 2018).

[16] 2017. Unity: Game objects. https://docs.unity3d.com/Manual/GameObjects.html. (Visited May 2018).

[17] 2018. Mongodb. https://www.mongodb.com/what-is-mongodb. (Last visited 2018).

[18] 2018. Couchdb. http://couchdb.apache.org/. (Last visited May 2018).

[19] 2018. Mysql. https://www.mysql.com/. (Last visited May 2018).

[20] Wikipedia contributors. 2018. Bejeweled — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Bejeweled&oldid=837849536. [Online; accessed 15-May-2018].

[21] Fine, R. 2017. Unityscript's long ride off into the sunset. https://blogs.unity3d.com/2017/08/11/unityscripts-long-ride-off-into-the-sunset/. (Visited May 2018).

[22] McCallum, S. 2018. Bachelor thesis template. https://github.com/COPCSE-NTNU/bachelor-thesis-NTNU. (Visited May 2018).

[23] 2018. Postman homepage. https://www.getpostman.com. (Visited May 2018).

[24] 2018. Go package testing. https://golang.org/pkg/testing/. (Visited april 2018).

[25] 2018. What is scrum? https://www.scrum.org/resources/what-is-scrum. (Last visited May 2018).

[26] Radigan, D. 2018. Kanban - a brief introduction. https://www.atlassian.com/agile/kanban. (Last visited May 2018).

[27] 2014. The awesomium wiki. http://wiki.awesomium.com/. (Last visited May 2018).

[28] 2018. The chromium projects. https://www.chromium.org/. (Last visited May 2018).

[29] Chasin, V. 2018. Simple unity browser. https://bitbucket.org/vitaly_chashin/simpleunitybrowser. (Downloaded Feb. 2018).

[30] Wikipedia contributors. 2018. Hawthorne effect — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Hawthorne_effect&oldid=840391987. [Online; accessed 13-May-2018].

[31] nasjonale forskningsetiske komiteene, D. 2009. Retningslinjer for inklusjon av voksne personer med manglende eller redusert samtykkekompetanse i helsefaglig forskning. https://www.etikkom.no/forskningsetiske-retningslinjer/Medisin-og-helse/Redusert-samtykkekompetanse/. (Last visited January 2018).

[32] 2018. Tobii dynavox. https://www.tobiidynavox.com/. (Last visited May 2018).

[33] 2018. Glassouse. http://glassouse.com/#home. (Last visited May 2018).

[34] 2018. Computer vision syndrome. https://www.aoa.org/patients-and-public/caring-for-your-vision/protecting-your-vision/computer-vision-syndrome. (Last visited May 2018).

[35] 2018. How does the 20-20-20 rule prevent eye strain? https://www.healthline.com/health/eye-health/20-20-20-rule#research. (Last visited May 2018).

A Terminology

AAA game A game published by a major publisher, often involving higher levels of standard, but also higher risks.

ALS Amyotrophic lateral sclerosis, also known as Lou Gehrig's disease. Motor neuron disease which gradually degrades affected persons ability to control their muscles.

C++ "Cplusplus". Programming language developed by Bjarne Stroustrup.

C# "C sharp". Programming language developed by Microsoft, based on the C language. Used by Unity for scripting.

CVS Computer Vision Syndrome. An umbrella term for conditions that can be caused by using a digital video screen. Includes dry eyes, eye muscle strain, red eye, headaches, blurred vision, as well as strained neck muscles or shoulder muscles.

Eye tracking To track a users eye, and their positioning through a multitude of methods. Often used interchangeably with gaze tracking.

Gaze tracking To track a users eyes, and determine what/where they are looking. Commonly used when referring to what a user is focusing on a screen. Often used interchangeably with eye tracking.

GOLANG GO. Programming language developed by Google to be open source and easy to use.

JS JavaScript. Scripting language commonly used for web applications.

JSON JavaScript Object Notation. Commonly used for saving data in text-based structures.

Plugin Package that install new functions on existing software.

SDK Software Development Kit. Tools, such as framework or API, for developing a specific software.

Sunnaas Sykehus One of Norway's largest hospitals specializing in rehabilitation.

Tobii A company that specializes in making hardware for the use of eye tracking/gaze tracking. Products to note for the project is EyeX and 4C.

Trello Trello is a web application for handling issue tracking, and development progress.

UI User Interface. The design of the interface the user interacts with.

UX User Experience. What kind of experience the user gets from the User Interface.

B Plan Template

Plan Template

Bachelor in gameprogramming: VR Eye-tracking

Table of contents

Goals and framework

Background

Our project has been given to us by our contractor Richard Barlow, from Progress (Company name). The main goal of the project is the provide a software based entirely on eye-tracking, especially fitting individuals with severe tetraplegia. These individuals often have very limited functionality, some only limited to movement of the eyes. A parent goal would be to increase the quality of life (henceforth referenced to as “QoL”) of individuals with severe tetraplegia or other limiting functions by providing a software platform allowing a higher level of interaction.

Project goals

Main goal:

Develop software whose interaction is only based on eye-tracking through a VR headset (based on available hardware).

Subgoals:

- Allow tetraplegics et al. to participate more as a normal citizen online.
- Give tetraplegics et al. a platform which allows them to interact more easily.
- Provide tetraplegics et al. with interactive entertainment, such as, but not limited to video games.

Project development goals:

- Effectively utilize relevant individuals as test subjects for functionality testing.
- Through collaboration provide the base for further software development, [a base] providing a future developer easily readable code, and a code standard which provides such a developer with a good starting platform.

Scope

Field of study

Interactive software development and game development.

Delimitation

- Virtual reality-based Eye-tracking
- Only UI interaction with eye(s).
- Developed in existing game engine (Unity3D)
- Hardware exclusivity(?)

Description

Develop a software platform where tetraplegics et al. can interact with said software using only their eye(s). A software platform where said interaction can further their feeling of coping skills, and give [tetraplegics et al.] a feeling of a wider interaction and better QoL than their current life situation gives them.

Project organization

Members

Students:

- Aune, Bjørn Kaare : Document organizer/manager
- Baardseth, Kristoffer : Project manager
- Wendling, Benjamin Gordon : Lead programmer

Supervisor:

- Mccallum, Dr. Simon (NTNU i Gjøvik)

Contractor/company contact:

- Barlow, Richard (Progress Interactive AS)

Project routines and rules

On-call work hours is 09.00 - 17.00, all weekdays (monday - friday)
All student members are expected to attend all meetings possible.
Internal project communication is expected to go through Discord.
All student members are expected to work a total of 30 hours every week.

- When the amount of hours are too low, a student is expected to increase workload the following week(s).
- When a student is planning absence, they are expected to build a “buffer” of hours.
- All work should be logged in hours (through toggl team/project), as well as in progress.

No group members are allowed to use the software to develop nuclear weaponry.
All group members should use software decided by the group as a whole.
All group members are expected to properly and with a certain standard document their work:

- All git commit-messages should be descriptive of the changes made, and contain trello project number where applicable.
- All references should be done with Harvard-style reference system.
- All dates should follow ISO8601 (eg.: 2018-01-15, or YYYY-MM-DD)

No scrum-meeting should last longer than 15 minutes. It is project manager's responsibility to track time and effectiveness of meetings.
All members should be free to criticize other members work, as long as it is constructive and helpful to the group as a whole.

In the event of a student group member breaks any of these rules, he is expected to provide other member(s) with 1 unit of 0.5L beer.

Student Member: Aune, Bjørn Kaare

Student Member: Baardseth, Kristoffer

Student Member: Wendling, Benjamin Gordon

Planning, follow-through, and report

Main project division

Choice of development model

Choice: Incremental development style towards project milestones, subdivided into scrum-style sprints.

Pros:	Cons:
Easier to split project into milestones Easy to change milestones if necessary Geared toward version releases (helpful towards contractor) Geared towards a smaller group Easier to gather group towards a single goal Risk identification can be done between increments/sprints, flexible Encourages communication through daily and weekly meetings, low time-cost	Plan might be “too flexible”, too much room for comfort No defined “End-of-project” Time spent on meetings Some project overhead during development, requires effective use of tools

We have decided to use a lean development model seeing as our project is very open-ended, with the task geared towards a general outline of what we should work towards. This gives us more leeway to change our plan as we progress between sprints and increments, and does not restrict our planning on a set goal, but allows us to decide between milestones and goals.

We have decided to use what we’ll call “Incremental Scrum”, splitting our project into incremental *Milestones,* points at which our project reaches a stage that adds a large function to the software as a whole. Our currently defined milestones are:

Milestone 1: Basic user interaction functionality

This milestone defines the point at which basic interaction is possible with the software. These “basic” interactions, we have decided to be writing/clicking, web-browser usage, and simple entertainment, namely Sudoku and/or Bejeweled.

Milestone 2: Social user interaction functionality

This milestone defines the point at which online social interaction is possible with the software. These interactions should include a chat/chatroom, a game (Draughts(Br.Eng) / Checkers (Am.Eng)), and saving user data along with friend-lists and chat-rooms.

Milestone 3:

This milestone defines the point at which we manage to fully integrate or fit our UI to use a program like FreeCiv or FlightGear. This is considered to be a stretch goal that we should work towards, to test the integration of eye-tracking only with different software.

Scrum

Incremental Scrum is using elements from both Incremental Development and the Scrum Development Model. Therefore we want to clarify what we want to use from the scrum model.

Roles:

We have decided not to follow Scrum's framework for roles. Compared to scrum, our own roles would match it best like the following:

Product owner:
Our group does not contain a product owner in the sense that scrum has one. Our closest product owner is supervisor S. McCallum, who will aid our "direction" while we are developing the software, and help us communicate with contractor R. Barlow.

Development team:
Our development team consists of all three student members, who will work on the entirety of the project. This is our most "true-to-name" role.

Scrum master:
The project manager will, in essence, function as a scrum master. While Scrum emphasizes on its Scrum Master to coach and guide the team throughout the project, our project manager's responsibility is to keep the development team "on track", and make sure that the project is progressing towards the same end goal. The project leader will be a decision maker in the event that the project is stagnating.

Meetings and Sprints

Our main takeaway from Scrum is the use of sprints and a product backlog, as well as sprint meetings and daily scrum meetings. Since we have a relatively open-ended project, in terms of implementation goals and end product, the agility of scrum sprints is more appropriate than more rigid development models, because we might need to change our definition of what the end product should be at some occasions.

Plan for status meetings and decision points

We have planned weekly meetings with our guidance counselor, every monday at 13.30. We expect these meetings to give us aid in deciding our most important tasks for the coming week(s). We also hope that they can provide us some insight into what direction our project is currently taking, and help us foresee, understand, and prepare for future problems and workloads.

We hope to have frequent meetings with our contractor, bi-weekly or more, to relay our progress, and get a clear idea of what our next current goal is or should be. These meetings should also be used to determine how we should approach the project in terms of milestones.

The group will hold set meetings at the end of every sprint and incremental period, along with scrum-meetings during the development-phase of the project. Sprint meetings should be used to decide the end result of the sprint, and what the following sprint should contain. Increment meetings should happen at a point in time when the group reaches a milestone in development, such as finishing a prototype, a major function, etc. These meetings will be used to decide what our next milestone goal should be, and what tasks we should incorporate in the coming sprint(s) to reach that milestone goal.

Decision points for the project will be at the end of every incremental period (when a milestone is reached) and at the meetings with the contractor. When an increment is finished, we need to decide where we want to take our project next, based on time remaining and progress made. Meetings with the contractor should aid us in this progress, but if the contractors opinion is that the progress and project is not going in the right direction, we will have to change our approach, and potentially our next milestone goal.

Testing

An important part of our finished project is making our software intuitive and comfortable to use. To do this, properly testing the software is essential. Because we would like to test with the appropriate user-group as much as possible, contacting and meeting individuals and groups fitting our user-group should be a priority. There are certain things we need to ensure to do this:

- Our software needs an “out”-warning system. A tetraplegic might not be able to communicate without using their eyes, and with a VR-headset strapped to their head we or caretakers will not be able to see their eyes. Our software needs a function that let outsiders know that the individual using it wants to stop.
- We need to acquire the right papers to ask our testers for confirmation that they want to test our software. It is important to make sure that no testers that cannot communicate verbally or written tests the software without consenting to it.
- Having a writing tool in-software is needed to get good and descriptive feedback from testers who cannot otherwise communicate verbally or written. This function needs to be finished before any testing begins

Analysis

A sub-goal of our testing should be to analyze testing data input. This is mostly saving eye-tracking and input, to analyze the how a tester is moving his/her eyes while using the software.

Organization of quality assurance

Documentation

All code should be sufficiently documented before it is merged into our master/release branch. This entails that the code should not need more documentation after it is merged into master, but improving upon it is always encouraged.

Only finished code should be merged into master branch, so that master branch does not contain any functions, scripts, object, or other that is still a work in progress. All members are expected to merge only when they feel that their code is sufficiently tested and functional at an acceptable level.

All members are allowed and encouraged to read and learn each other's code and progress, to either encourage or critique it. This should be documented by all members for future references, so that we may continually improve upon the code and its documentation.

Risk analysis

Overall risk-analysis will be done at the beginning of the project, deciding on risks of the project as a whole. This does not entail risks in terms of coding certain functions or implementing certain functionalities, but rather the risks that can affect our overall progress of the project.
Some of these risks are:

- Lack of hardware technology

Without the right hardware, it will be hard to test our product based on its final form and shape. In our case, an eye-tracking system is needed to properly test all functions in terms of usability, comfort, and all basic functionality based on eye-tracking input. In such a case, our option is to build our project on the research we have made regarding eye-tracking systems, but test it using regular input, such as a keyboard and/or mouse. While this allows us to develop the core interactive functionality of our project, it does not allow us to implement eye-tracking integration, or test our interactive functionality using eye-tracking only. This could lead the end product to be lacking in terms of ease of use, comfort, and integration with eye-tracking solutions.

Notes: After the first meeting with Richard Barlow (Contract) on 2018-01-19, we were given a Tobii EyeX eye-tracking bar that can be mounted on a screen. We must consider that we will have to use this for our entire Bachelor Project. In contrast to VR, this will not be mounted on a person's head.

- Hardware lacks proper functionality

If the hardware we are using is found to be “lacking” considering our current goals and plan of implementation, we may have to change it drastically. This can cause our UI to be less comfortable and intuitive, and might give us problems tracking the eyes with the precision wanted.

- No user group testers

It is essential that we can test our software with our intended user group (tetraplegics), to suit it more towards their needs and interest. The testing is needed to further customize the software, especially the UI and UI interaction, to suit the core user group as much as possible. While we can test this ourselves, or use other non-handicapped individuals to test the software, having tetraplegics test it can give us some indication of what can be improved to suit their needs more, if so needed.

- Tobii rejects use of our software

The Tobii licensing for their software is mostly limited to interactive software. This should not pose a problem for our development, but we should ask for permission to use their SDK to save eye-tracking data for test-purposes. If they do not agree to this, we may have problems testing.

Plan for follow-through

18 ½ weeks total for sprints

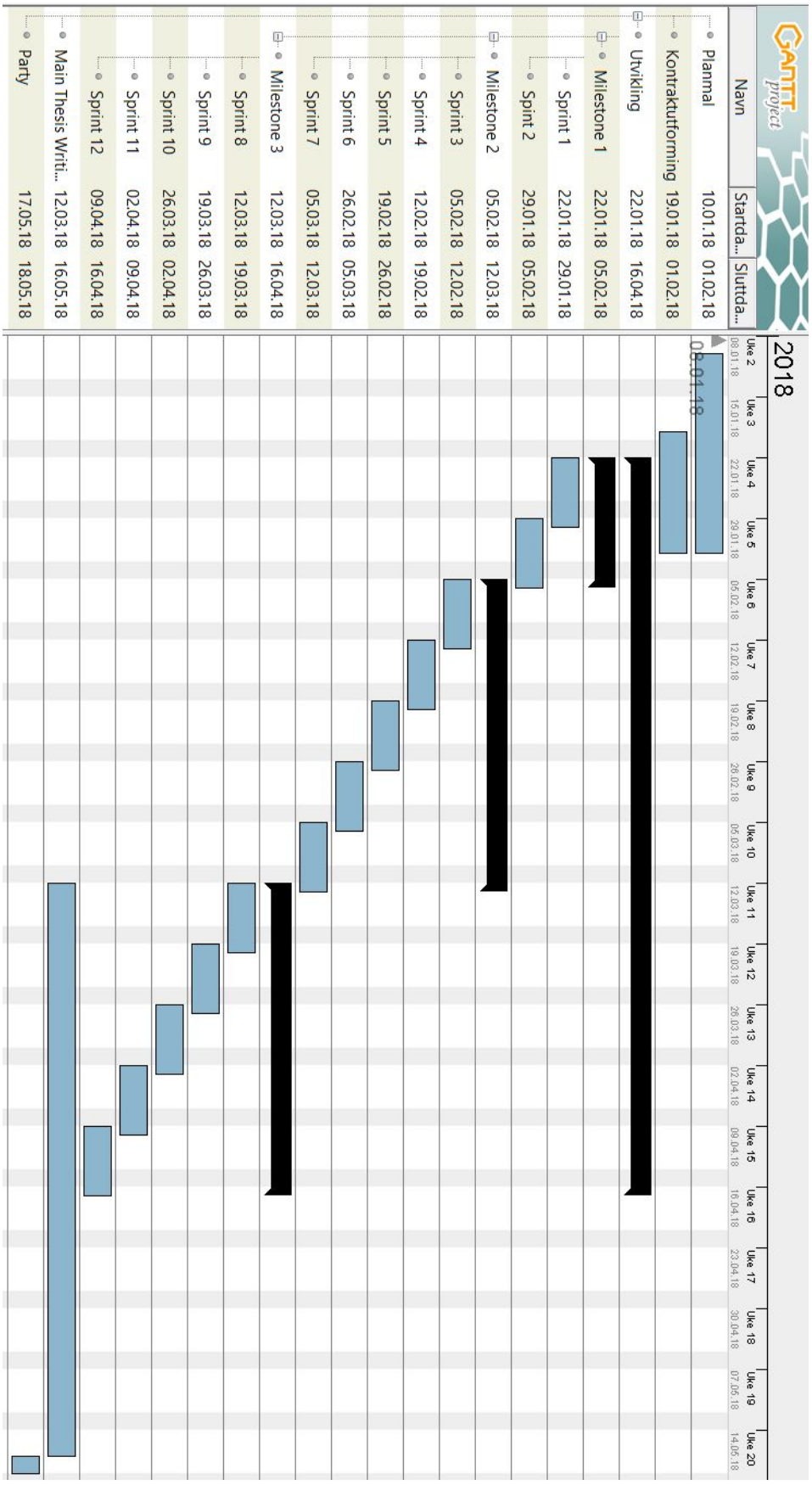
Sprint chart

Sprint number	Date of sprint	Sprint workload	Notes
Planning	2018.01-02.08-01	Start of BsC.project, intro to bachelor. Made contact with Contractor, guidance counselor. Finish the plan. Do project-repo set-up. Sign contract	A lot of time will be spent planning and fixing contacts, contracts, documents, software etc. Licensing should be taken care of. Research and sources are important.
Sprint 1	2018.01.22-28	Deliver plan/contract. Start thesis doc. Development: Main Menu On-screen keyboard Unity->Eye-tracker SDK integration.	Set up the thesis with ShareLaTeX. Start dev on the basic input functions of the software. Set-up hardware-to-unity if applicable.
Sprint 2	2018.01-02.29-04	Development: Reading and writing .txt-files Integrated web-viewer. Offline-game (Sudoku/Bejeweled) End of sprint puts us at Milestone 1.	Focus will be on integrating web-viewer with good UI and interaction. Text files would be good for users to write. Can be used for testing.
Sprint 3	2018.02.05-11	Documentation: Results from Milestone 1 Development: Online lobby functionality	Milestone 1 should be documented in good detail. Finesse is not yet needed. Focus should now be on implementing multiplayer/social interaction.

Sprint 4	2018.02.12-18	Development: Online chat w/ rooms	These phases should also include a large amount of testing
Sprint 5	2018.02.19-25	Development: Online gameplay (Checkers?)	
Sprint 6	2018.02-03.26-04	Development: Polishing of online features, rooms	
Sprint 7	2018.03.05-11	Documentation: Milestone 2 result Development: Integration with existing game, FreeCiv	
Sprint 8 - 12	2018.03.12 To 2018.04.16	Development: Further on FreeCiv, FlightGear, Polishing Documentation: More on Thesis	Crossphase of developing interface and polishing, while writing thesis
Main Thesis Writing	2018.04.16 To 2018.05.16	Documentation: Finish and delivery of thesis.	Writing of thesis, finishing documentation, cleanup.

Gantt-diagram

The Gantt-diagram is made to suit the sprint overview, and shows a timetable for all phases, until delivery.



C Contract

NTNU
Norwegian University of
Science and Technology

PROJECT AGREEMENT

between NTNU Faculty of Information Technology and Electrical Engineering (IE) at Gjøvik (education institution), and

Progress Interactive AS

(employer), and

Kristoffer Baardseth

Björn Koare Aune

Benjamin Gordon Wendling (student(s))

The agreement specifies obligations of the contracting parties concerning the completion of the project and the rights to use the results that the project produces:

1. The student(s) shall complete the project in the period from 01.02.2018 to 16.05.2018.

   The students shall in this period follow a set schedule where NTNU gives academic supervision. The employer contributes with project assistance as agreed upon at set times. The employer puts knowledge and materials at disposal necessary to complete the project. It is assumed that given problems in the project are adapted to a suitable level for the students' academic knowledge. It is the employer's duty to evaluate the project for free on enquiry from NTNU.

2. The costs of completion of the project are covered as follows:
   - The right of ownership to potential prototypes falls to those who have paid the components and materials and so on used to make the prototype. If it is necessary with larger or specific investments to complete the project, it has to be made an own agreement between parties about potential cost allocation and right of ownership.

3. NTNU is no guarantor that what employer has ordered works after intentions, nor that the project will be completed. The project must be considered as an exam related assignment that will be evaluated by lecturer/supervisor and examiner. Nevertheless it is an obligation

for the performer of the project to complete it according to specifications, function level and times as agreed upon.

4. The total assignment with drawings, models and apparatus as well as program listing, source codes and so on included as a part of or as an appendix to the assignment, is handed over as a copy to NTNU who free of charge can use it in lessons and in research purpose. The assignment or appendix cannot be used by NTNU for other purposes, and will not be handed over to an outsider without an agreement with the rest of the parties in this agreement. This applies as well to companies where employees at NTNU and/or students have interests.

   Assignments with grade C or better are registered and placed at the university's library. An electronic project assignment without attachments will be placed on the library part of NTNU's website. This depends on that the students sign a separate agreement where they give the library rights to make their main project available both on print and on Internet (ck. The Copyright Act). Employer and supervisor accept this kind of disclosure when they sign this project agreement, and they must possibly give a written message to students and dean if they during the project period change view on this kind of disclosure.

5. The assignment's specifications and results can be used by the employer's own work. If the student(s) in its assignment or while working with it, makes a patentable invention, relations between employer and student(s) applies as described in Act respecting the right to employees' inventions of 17th of April 1970, §§ 4-10.

6. Beyond the publishing mentioned in item 4, the student(s) have no right to publish his/hers/theirs assignment, fully or partly or as a part of another work, without consensus from the employer. Equivalent consent must be made between student(s) and lecturer/supervisor regarding the material placed at disposal by the lecturer/supervisor.

7. The students shall hand in the assignment with attachments electronic (PDF) in NTNU's digital exam system. In addition the students shall hand in a copy to the employer.

8. This agreement is drawn up with one copy to each party. On behalf of NTNU it is the head of the Department/Group that approves the agreement.

9. In each case it is possible to enter separate agreement between employer, student(s) and NTNU who closer regulate conditions regarding issues such as ownership, further use, confidentiality, cost coverage, and economic utilization of the results.

   If employer and student(s) wish an additional or new agreement, this will occur without NTNU as a partner.

10. When NTNU also act as employer, NTNU accede to the agreement both as education institution and as employer.

11. Possible disagreements concerning understanding of this agreement are solved by negotiations between the parties. If consensus is not achieved, the parties agree that the disagreement is solved by arbitration, according to provision in Civil Procedure Act of 13th of August 1915, no 6, chapter 32.

12. Participants by project implementation:

NTNUs supervisor (name): Simon McCallum

Employers contact person (name): Richard Barlow

Student(s) (signature): _____ date 20.02.2018

_____ date 20.02.2018

Benjamin G. Wendling date 20.02.2018

_____ date _____

Employer (signature): _____ date 19.02.18

The Project Agreement is to be handed in in digital version in Blackboard. Digital approval by head of the Department/Group.

If a paper version of the Agreement is needed, is must be handed in at the Department in additional.

Head of Department/Group (signature): _____ date _____

D Tobii SDK License v2

LICENSE AGREEMENT FOR TOBII CORE SDK AND TOBII GAMING SDK

Document version 1.2

Please note! The Tobii Core SDK and the Tobii Gaming SDK are software development kits intended for use in interactive and gaming applications only.

If you want to develop or distribute software for so-called "Analytical Use", where eye tracking data is stored or transferred to another device with the purpose to analyze, record, visualize or interpret behavior or attention, you must instead use the Tobii Pro SDK. Applications developed using the Tobii Pro SDK are compatible with Tobii Pro eye tracker hardware, as well as with consumer devices with Tobii eye tracking in combination with a specific license key.

More information about Tobii's different SDKs are available at developer.tobii.com/tobii-sdk-guide/*.*

We also offer other licensing options to selected partners - please contact SDKlicensing@tobii.com if you want to discuss your situation.

PREAMBLE

This *License Agreement* (the "**Agreement**") forms a legally binding contract between **Tobii AB (publ)** (reg. No. 556613-9654), with registered office at Karlsrovägen 2D, SE-182 53, Danderyd, Sweden ("**Tobii**"), and the licensee ("**Licensee**"). The Licensee is entered as the Licensee by completing the *Licensee Information Box* (the "**Infobox**") when downloading and installing the *Tobii Core SDK or Tobii Gaming SDK* (the "**SDK**"), or by utilizing, accessing or distributing the Software Components in any other manner.

An individual entering as a Licensee on behalf of a legal person (e.g. his or her employer) confirms the authority to bind such legal person in accordance with the terms and conditions of this Agreement. An individual that does not have the necessary authority, may neither accept the terms and conditions below, nor use the SDK, on behalf of the legal person.

BY DOWNLOADING, INSTALLING, USING, ACCESSING OR DISTRIBUTING THE SDK OR THE SOFTWARE COMPONENTS, LICENSEE (i) CONFIRMS THAT LICENSEE HAS READ AND UNDERSTOOD THE TERMS AND CONDITIONS BELOW; AND (ii) AGREES TO BE BOUND BY THIS AGREEMENT.

BY DOWNLOADING, INSTALLING, USING, ACCESSING OR DISTRIBUTING THE SDK OR THE SOFTWARE COMPONENTS, LICENSEE FURTHER AGREES THAT (i) THIS AGREEMENT IS THE COMPLETE AND EXCLUSIVE STATEMENT OF THE RIGHTS AND LIABILITY BETWEEN LICENSEE AND TOBII IN RELATION TO LICENSEE'S USE OF THE SDK AND DISTRIBUTION OF THE SOFTWARE COMPONENTS (AS DEFINED HEREIN); AND (ii) THIS AGREEMENT SUPERSEDES ALL PRIOR COMMUNICATIONS (BOTH ORAL AND WRITTEN), STATEMENTS IN MARKETING, ADVERTISING, OR ANY OTHER COMMUNICATION BETWEEN LICENSEE AND TOBII CONCERNING THE SDK (INCLUDING THE SOFTWARE COMPONENTS).

TERMS AND CONDITIONS

1. Background.

1.1 Tobii is a supplier of hardware and software solutions for eye tracking.

1.2 Licensee is a software vendor or individual that develops, markets and licenses software solutions or intends to do the same.

1.3 The SDK is a "toolbox" (including *inter alia* various building blocks) for developing applications to use Tobii Data as a user input for interactive experiences in games or other software (**Gaze Interaction Use**).

1.4 **This Agreement is intended to provide Licensee with limited rights to develop, use and, if applicable, distribute applications for Gaze Interaction Use in games and other software, that process data from Tobii Hardware and Software only.** A separate commercial license with Tobii is required if Licensee desires the right to use the SDK or Software Components for any other purpose.

2. Definitions.

In addition to the capitalized words defined in the *Preamble* and

Section 1, the following defined terms are used in this Agreement:

2.1 **"Application"** means any software application developed by Licensee (i) using the SDK; and/or (ii) including, utilizing or accessing the Software Components.

2.2 **"Effective Date"** means the date when the Licensee completes the Infobox (as defined in the Preamble) and downloads the SDK, or otherwise starts to use the SDK or Software Components.

2.3 **"End User"** means any person or entity that sub-licenses Software Components through Licensee (or Licensee's appointed resellers or distributors) as part of an Application.

2.4 **"Gaze Interaction Use"** means to use Tobii Data as a user input for interactive experiences in games or other software.

2.5 **"Software Components"** means all object code files (for example dynamic- link library files, commonly referred to as DLL-files or .SO files) of the SDK (and any Updates, modifications and/or patches or hot fixes thereto that Tobii may make generally available from time to time) that are intended to be reused in an Application. This includes, but is not limited to, the Tobii Stream Engine and the Tobii Interaction Engine.

2.6 **"Tobii Hardware and Software"** means eye tracking hardware and software components designed or provided by Tobii. These components may be provided directly by Tobii or incorporated within a third party product.

2.7 **"Tobii Data"** means data about eye movements, attention or behavior, or any other data generated by or derived from the Software Components.

2.8 **"Tobii Materials"** means the SDK (including the Software Components) and any ideas, know-how, programs, processes, designs, inventions, works and other information, which may be developed or created by Tobii.

2.9 **"Trademarks"** means the registered or unregistered trademarks and service marks related to the SDK or the Software Components that Tobii may adopt from time to time.

2.10 **"Updates"** means (to the extent that such items are not accompanied by a separate license agreement or terms of use) any subsequent releases, software updates, add-on components, stencils, templates, shapes, web services and/or supplements of the Software Components intended to replace or enhance a prior release of the Software Components.

3. **Grant of License.**

3.1 **Grant of License.** Subject to the limitations specified in this Agreement and during the term of this Agreement, Tobii grants Licensee a limited, worldwide, royalty-free, non-assignable, revocable and non-exclusive license to use the SDK (including the Software Components and use of example code) solely to develop, use and distribute Applications that process data from Tobii Hardware and Software for the purpose of Gaze Interaction Use.

3.2 **Limitations**. Licensee may not (i) copy (except for backup purposes), modify, adapt, decompile, reverse engineer, disassemble, or create derivative works of the Software Components; or (ii) use the SDK to build software that may be used as an SDK providing API's that use functionality from the Software Components.

3.3 **Grant of sub-licensing rights**. During the term of this Agreement, Tobii designates Licensee as non-exclusive sub-licensor for the Software Components. The right to sublicense (directly or indirectly through appointed resellers or distributors) the Software Components is non-transferable and applies solely to the sub-licensing of the Software Components in machine-readable object code version to End Users licensing the Application.

3.4 **Updates**. The terms of this Agreement will apply to any Updates, modifications and/or patches or hot fixes that Tobii makes available to Licensee. Licensee agrees that Updates may require Licensee to change or update the Applications. Updates may also affect Licensee's ability to use, access or interact with the SDK.

3.5 **Anonymous data logging**. The Software Components may log information about the use of the Software Components on Tobii servers. Such information is limited to the usage of the Software Components, and does not contain any actual gaze data information or other personal identifiable data.

3.6 **Not for Analytical Use**. Unless Licensee enters into a separate agreement with Tobii, the SDK may not be used to develop and distribute software that (a) store Tobii Data; or (b) transfer Tobii Data to another computing device or network; in both cases where the intent is to use or make it possible to use Tobii Data to analyze, record, visualize or interpret behavior or attention ("**Analytical Use**"). To clarify; storing, using or transferring Tobii Data for the sole purpose of implementing software that uses Tobii Data for Gaze Interaction Use does not constitute Analytical Use.

3.7 **No High Risk Use or Medical Classified Use.** The SDK (including the Software Components) is not fault tolerant, and is accordingly not designed or intended for use in any software or situation where failure or fault of any kind could lead to death or serious bodily injury of any person, or to severe physical, environmental or property damage (**"High Risk Use"**). The SDK (including the Software Components) may only be used to develop Applications that do not involve High Risk Use. High Risk Use includes, for example, aircraft navigation, military and industrial use, control of nuclear, chemical facilities and of other modes of human mass transportation, as well as medical, surgical, or other use intended to support or sustain life. Furthermore, but nonetheless, the SDK (including the Software Components) is not certified for medical classified environments (**"Medical Classified Use"**) and Tobii prohibits any such use of the SDK or the Software Components, unless a special license agreement is entered into for this purpose. Tobii disclaims liability for all such use.

4. **Support for the Software Components.**

4.1 **No support of the Software Components.** Tobii is not obliged to support Licensee regarding the use of the SDK (including the Software Components) unless a separate support agreement between the parties has been entered into.

4.2 **No support of the Application.** Tobii shall not be responsible for the support of the Application or any other application that uses the Software Components to access data, content or resources.

5. **Use of Trademarks.** Licensee shall not remove or alter any Trademark, copyright, patent or other proprietary notices contained in the SDK (including the Software Components).

6. **Ownership.** Except for the licenses and rights explicitly granted

herein to Licensee, Tobii retains all right, title and interest in and to the Tobii Materials, and all patents, copyrights, Trademarks, trade names, trade secrets and other proprietary rights in or related to the Tobii Materials, whether or not specifically recognized or perfected under the laws of the country in which the Tobii Materials are located. Nothing contained in this Agreement shall be construed to transfer any rights in or to the Tobii Materials or Tobii's patents other than as explicitly set forth in this Agreement.

7. **Changes.**

7.1 Tobii reserve the right to change in its sole discretion this Agreement or the SDK at any time.

7.2 Tobii may require that Licensee either accepts and agrees to new or revised terms of this Agreement, or, if Licensee does not agree to the new or revised terms, ceases or terminates the use of the SDK. Licensee's continued use of the SDK after changes to this Agreement take effect will constitute Licensee's acceptances of the changed terms. If Licensee does not agree to a change, Licensee must stop using the SDK and terminate this Agreement. For the avoidance of doubt, changed terms do not take retroactive effect with respect to any Application developed before the change or any Application first distributed before the change.

8. **Indemnification.**

8.1 **No Tobii indemnification.** SINCE TOBII GRANTS LICENSEE THE RIGHT TO USE THE SDK AND TO SUB-LICENSE THE SOFTWARE COMPONENTS FOR FREE, TOBII MAKES NO REPRESENTATION OR WARRANTY ON NON-INFRINGEMENT AND TOBII WILL NOT DEFEND AND HOLD LICENSEE, LICENSEE'S AFFILIATES AND THEIR RESPECTIVE OFFICERS, DIRECTORS, EMPLOYEES AND AGENTS, HARMLESS FROM ANY CLAIM FROM A THIRD PARTY THAT THE SDK OR THE SOFTWARE COMPONENTS INFRINGE ANY PATENT, TRADE SECRET OR COPYRIGHT.

8.2 **Licensee's indemnification.** Licensee shall defend and hold Tobii and its officers, directors, employees, subsidiaries and agents harmless from (i) any claim by a third party that an Application infringes any patent, trade secret or copyright of any third party; provided that, Licensee shall not have any obligation to indemnify Tobii if such claim relates only to the Software Components as provided by Tobii; and (ii) any claim, allegation, liability or loss suffered by Tobii arising from Licensee's breach of any provision in this Agreement, provided that: (a) Licensee is promptly notified in writing of the claim; (b) Licensee has sole control in the defense of any claim and any settlement negotiations attendant thereto; and (c) Tobii provides Licensee, at Licensee's expense, all reasonable assistance, information and cooperation to defend or settle the claim. Licensee shall not enter into any settlement of any claim covered by the above indemnification without the prior approval of Tobii, which approval will not be unreasonably withheld. Tobii shall have the right to retain separate counsel and participate in the defense of the action or claim at its own expense.

9. **Term and Termination.**

9.1 **Term.** This Agreement shall become effective on the Effective Date and shall continue until terminated.

9.2 **Termination by Licensee**. Licensee may terminate this Agreement at any time by (i) uninstalling and destroying all copies of the SDK that are in the possession, custody or control of Licensee and its organization; and (ii) providing Tobii written notice thereof.

9.3 **Termination by Tobii.** Tobii may terminate this Agreement for any reason upon six (6) months written notice. Tobii may also terminate this Agreement immediately if Licensee breaches this Agreement and has not cured such breach within thirty (30) days from Tobii's notice to Licensee of the nature of the breach.

9.4 **Survival of obligations.** The following obligations will survive termination of the Agreement for any reason: (i) all obligations relating to protection of proprietary rights; and (ii) all obligations regarding audits; and (iii) all provisions regarding the limitations of warranty, remedy and liability.

9.5 **Effects of termination.** Upon termination of this Agreement for any reason, all rights and licenses granted hereunder shall terminate and revert to Tobii. Any termination of this Agreement except for termination due to Licensee's breach of contract will not affect Licensee's right, subject to Licensee´s continued compliance with Licensee's obligations under this Agreement, to continue to distribute versions of the Applications created and first distributed before termination, and will not affect the right of the End Users to continue using such versions of the Application, both of which rights will survive termination.

10. **Reputation, Goodwill and Compliance.** Licensee shall not knowingly make false or misleading representations with regard to the Software Components or Tobii. Licensee further agrees to conduct business in a professional manner and act in good faith with respect to the Software Components and the good reputation of Tobii. Licensee represents and warrants that it (i) will conduct its performance under this Agreement at all times in keeping with professional standards of ethics and integrity; and (ii) is familiar with applicable laws concerning bribery, corruption and prohibited business practices, and will at all times perform in accordance with the requirements of such laws.

11. **Disclaimer of Warranties**

11.1 **"As is".** Since Tobii grants Licensee the right to use the SDK and the Software Components for free, Licensee's use of the SDK and the Software Components and the sub-licensing of Software Components is at Licensee's sole risk. The SDK and the Software Components are provided "as is" and "as available" without warranty of any kind from Tobii.

11.2 **Complete Disclaimer.** EXCEPT AS SPECIFICALLY PROVIDED HEREIN TOBII MAKES NO WARRANTY, EITHER EXPRESS OR IMPLIED, RELATING TO THE SDK OR THE SOFTWARE COMPONENTS, AND TOBII FURTHER EXPRESSLY DISCLAIMS, TO THE EXTENT PERMITTED BY APPLICABLE LAW, ALL WARRANTIES AND CONDITIONS OF ANY KIND RELATED TO THE SDK OR THE SOFTWARE COMPONENTS, WHETHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO ANY IMPLIED WARRANTIES AND CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT.

12. Limitation of Liability. IN NO EVENT SHALL TOBII BE LIABLE TO LICENSEE, LICENSEE'S AFFILIATES OR ANY END USER UNDER ANY THEORY OF LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES, INCLUDING ANY LOSS OF DATA, THAT MAY BE INCURRED BY LICENSEE, LICENSEE'S AFFILIATES OR ANY END USER, DUE TO THE USE OF THE SDK OR ONE

OR SEVERAL OF THE SOFTWARE COMPONENTS, WHETHER OR NOT TOBII HAD BEEN ADVISED OF OR SHOULD HAVE BEEN AWARE OF THE POSSIBILITY OF ANY SUCH LOSSES ARISING.

13. General.

13.1 **Waiver; Severability.** Except as may be affirmed in writing by the parties, no failure or delay by either party in exercising any right, power or privilege hereunder will operate as a waiver or preclude further exercise thereof. If any provision of this Agreement shall be held by a court of competent jurisdiction to be contrary to law, that provision will be enforced to the maximum extent possible, and the remaining provisions of this Agreement will remain in full force and effect.

13.2 **Entire agreement.** This Agreement sets forth the entire understanding between the parties and supersedes any prior communication or agreement between the parties regarding the right to use the SDK or sub-licensing of the Software Components.

13.3 **Governing Law.** This Agreement shall be construed and enforced in accordance with the laws of Sweden, without giving effect to its conflict of law provisions.

13.4 **Dispute Resolution**. Any dispute, controversy or claim arising out of or in connection with this Agreement or the breach, termination or invalidity thereof, shall be finally settled by arbitration administered at the Arbitration Institute of the Stockholm Chamber of Commerce (the "**Institute**"). The place of arbitration shall be Stockholm and the arbitration shall be conducted in English language. The Rules of the Institute shall apply, and the Institute shall decide whether the tribunal shall be composed of one or three arbitrators. At the option of either party, and if the amount in dispute does not exceed EUR 500,000 the Institute's Rules for Expedited Arbitrations shall apply. The amount in dispute includes the claimant's claims in the Request for Arbitration and any counterclaims in the respondent's reply to the Request for Arbitration.

* * *

E Test Questionnaires

The questionnaires are split in two parts: one taken before the test was conducted, and one taken after the test was conducted.

Questionnaire: Eye-tracking software

Questions to be answered before testing the eye-tracking

Subject:

Date:

1. **Have you ever used eye-tracking hardware and/or software?**
   Mark only one oval.
   - ◯ Yes
   - ◯ No

2. **How often do you use digital displays (Smartphone, TV, Computer screen, etc.) on a daily basis?**
   Mark only one oval.
   - ◯ 10 hours or more
   - ◯ 7 - 9 hours
   - ◯ 4-6 hours
   - ◯ 1-3 hours
   - ◯ Less than 1 hour

3. **Which platform do you most commonly use?**
   Mark only one oval.
   - ◯ Mobile/Cellphone
   - ◯ Tablet/Pad
   - ◯ Laptop
   - ◯ Stationary
   - ◯ Smartwatch
   - ◯ Other, please specify:

4. **Do you think the eye-tracker would have any reason not to work for you specifically?**
   Mark only one oval.
   - ◯ Yes
   - ◯ No
   - ◯ Maybe

5. **Optional: if yes or maybe on the previous question, why?**

6. **Do you have any experience designing or implementing software?**
   Mark only one oval.

   - ◯ Yes
   - ◯ No

Questionnaire: Eye-tracking software

Questios to be answered after testing the eye-tracking software.

Subject:

Date:

On the following questions, select the option you agree with the most.

1. **I felt physically comfortable using the software**
   Mark only one oval.

	1	2	3	4	5	6	7	
Disagree strongly	◯	◯	◯	◯	◯	◯	◯	Agree strongly

2. **It was hard to use the software**
   Mark only one oval.

	1	2	3	4	5	6	7	
Disagree strongly	◯	◯	◯	◯	◯	◯	◯	Agree strongly

3. **The software felt precise and accurate**
   Mark only one oval.

	1	2	3	4	5	6	7	
Disagree strongly	◯	◯	◯	◯	◯	◯	◯	Agree strongly

4. **I could use this software on a daily basis**
   Mark only one oval.

	1	2	3	4	5	6	7	
Disagree strongly	◯	◯	◯	◯	◯	◯	◯	Agree strongly

5. **Is there any activities you think might work better with eye-tracking?**

6. **Why would you / would you not use the software on a daily basis?**

7. **Other:**

F Doxygen Documentation

Bachelor Project Doxygen Documentation

Generated by Doxygen 1.8.13

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

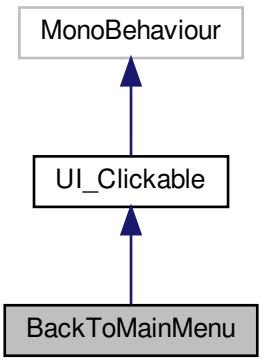
Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

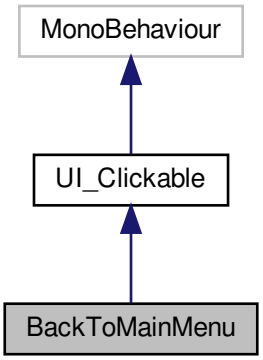
Class Documentation

3.1 BackToMainMenu Class Reference

Inheritance diagram for BackToMainMenu:



Collaboration diagram for BackToMainMenu:



Public Member Functions

- override void OnClickEnter ()

  When clickable UI element is gazed upon enter this function.

3.1.1 Member Function Documentation

3.1.1.1 OnClickEnter()

```
override void BackToMainMenu.OnClickEnter ( )  [virtual]
```

When clickable UI element is gazed upon enter this function.

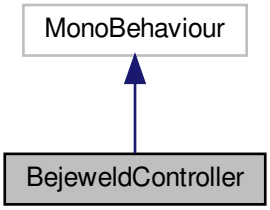
Implements UI_Clickable.

The documentation for this class was generated from the following file:

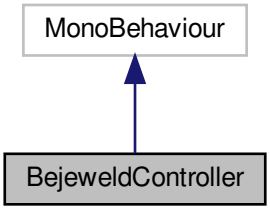
- BackToMainMenu.cs

3.2 BejeweldController Class Reference

Inheritance diagram for BejeweldController:



Collaboration diagram for BejeweldController:



Public Member Functions

- void **ChangePlaying** ()
- string **GetPointsAsString** ()

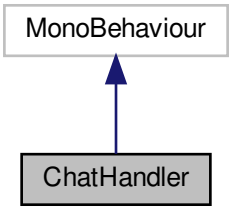
The documentation for this class was generated from the following file:

- BejeweldController.cs

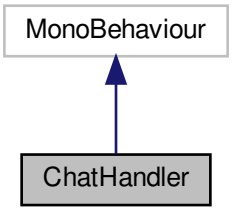
3.3 ChatHandler Class Reference

Class that controls chat window, and communicates with server through Network manager singleton

Inheritance diagram for ChatHandler:



Collaboration diagram for ChatHandler:



Public Member Functions

- void SendMessageToServer (string messageText, string recipient)
  Sends a chat message to the server.

3.3.1 Detailed Description

Class that controls chat window, and communicates with server through Network manager singleton

3.3.2 Member Function Documentation

3.3.2.1 SendMessageToServer()

```
void ChatHandler.SendMessageToServer (
            string messageText,
            string recipient )
```

Sends a chat message to the server.

Parameters

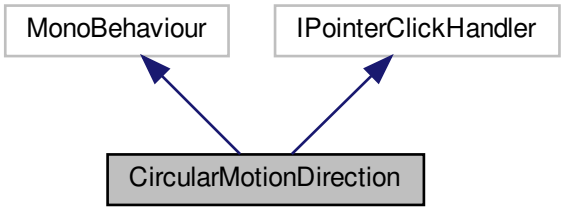
messageText	Contents of chat message
recipient	The target user to receive chat message

The documentation for this class was generated from the following file:

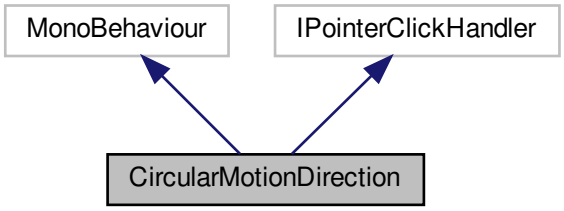
- ChatHandler.cs

3.4 CircularMotionDirection Class Reference

Inheritance diagram for CircularMotionDirection:



Collaboration diagram for CircularMotionDirection:



Public Types

- enum **MoveDir** {
  UP, **DOWN**, **LEFT**, **RIGHT**,
  ZOOM_IN, **ZOOM_OUT**, **RESET_CAMERA** }

Public Member Functions

- void **OnPointerClick** (PointerEventData eventData)
- void Clicked ()

  Depending on set enum, moves the camera around inside the view of the chosen usage canvas. Camera can move to edges of set usage canvas, and zoom as close as possible in z-axis position. Movement-size is defined by CircularMotionSettings.

Public Attributes

- GazeAware **gaze**

3.4.1 Member Function Documentation

3.4.1.1 Clicked()

```
void CircularMotionDirection.Clicked ( )
```

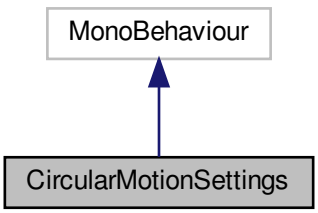
Depending on set enum, moves the camera around inside the view of the chosen usage canvas. Camera can move to edges of set usage canvas, and zoom as close as possible in z-axis position. Movement-size is defined by CircularMotionSettings.

The documentation for this class was generated from the following file:

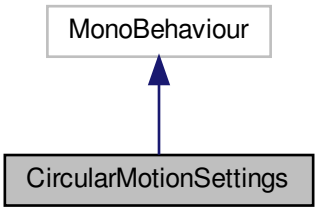
- CircularMotionDirection.cs

3.5 CircularMotionSettings Class Reference

Inheritance diagram for CircularMotionSettings:



Collaboration diagram for CircularMotionSettings:



Public Attributes

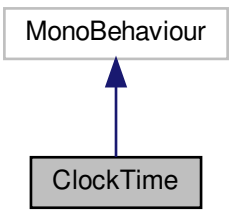
- Camera **mainCamera**
- Canvas **motionCircleCanvas**
- Canvas **useAreaCanvas**
- float **cameraMoveSpeed** = 0.5f
- Vector3 **stdCameraPos** = new Vector3(0, 0, 0)

The documentation for this class was generated from the following file:

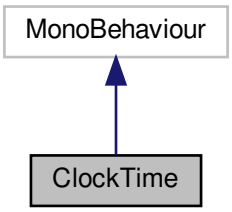
- CircularMotionSettings.cs

3.6 ClockTime Class Reference

Inheritance diagram for ClockTime:



Collaboration diagram for ClockTime:

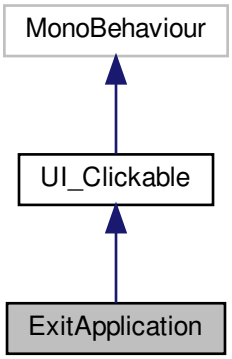


The documentation for this class was generated from the following file:

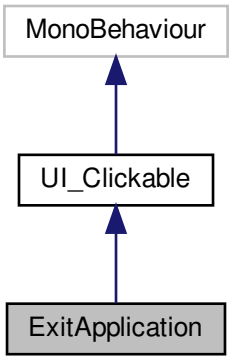
- ClockTime.cs

3.7 ExitApplication Class Reference

Inheritance diagram for ExitApplication:



Collaboration diagram for ExitApplication:



Public Member Functions

- override void OnClickEnter ()

  When clickable UI element is gazed upon enter this function.

3.7.1 Member Function Documentation

3.7.1.1 OnClickEnter()

```
override void ExitApplication.OnClickEnter ( )  [virtual]
```

When clickable UI element is gazed upon enter this function.

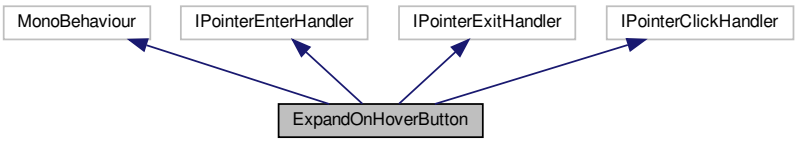
Implements UI_Clickable.

The documentation for this class was generated from the following file:

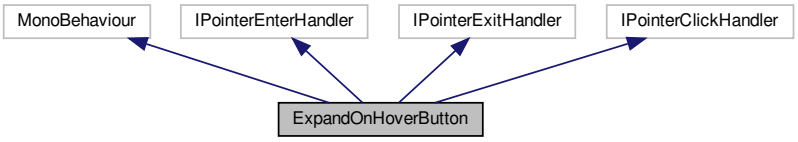
- ExitApplication.cs

3.8 ExpandOnHoverButton Class Reference

Inheritance diagram for ExpandOnHoverButton:



Collaboration diagram for ExpandOnHoverButton:



Public Member Functions

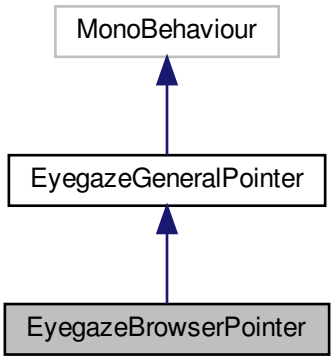
- void **OnPointerEnter** (PointerEventData data)
- void **OnPointerExit** (PointerEventData data)
- void **OnPointerClick** (PointerEventData data)

The documentation for this class was generated from the following file:

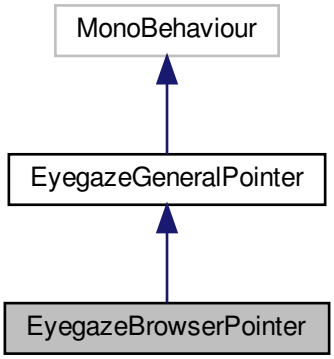
- ExpandOnHoverButton.cs

3.9 EyegazeBrowserPointer Class Reference

Inheritance diagram for EyegazeBrowserPointer:



Collaboration diagram for EyegazeBrowserPointer:



Public Member Functions

- override void UpdateGazePointerPosition (GazePoint gazePoint)

  Updates the pointer position in the world space, projects from screen plane to plane in world Smoothes pointer position on movement (Remove jagged movement)
- override void SendClickMessage ()

  Uses to world space pointer to send a mouse click to the browser. Translates positions from world space to browser transform. Sends a click+release event as mouse to browser

Public Attributes

- GameObject **mBrowser**
- GameObject **mBrowserCanvas**

Additional Inherited Members

3.9.1 Member Function Documentation

3.9.1.1 SendClickMessage()

```
override void EyegazeBrowserPointer.SendClickMessage ( )  [virtual]
```

Uses to world space pointer to send a mouse click to the browser. Translates positions from world space to browser transform. Sends a click+release event as mouse to browser

Reimplemented from EyegazeGeneralPointer.

3.9.1.2 UpdateGazePointerPosition()

```
override void EyegazeBrowserPointer.UpdateGazePointerPosition (
            GazePoint gazePoint )  [virtual]
```

Updates the pointer position in the world space, projects from screen plane to plane in world Smoothes pointer position on movement (Remove jagged movement)

Parameters

gazePoint	

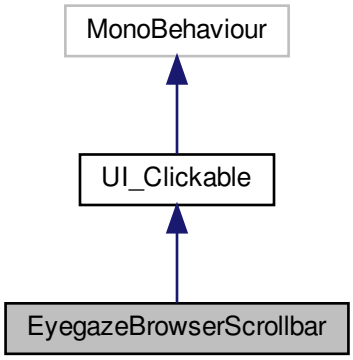
Reimplemented from EyegazeGeneralPointer.

The documentation for this class was generated from the following file:

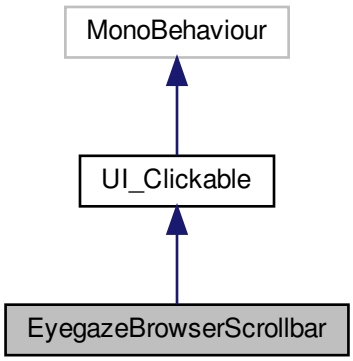
- EyegazeBrowserPointer.cs

3.10 EyegazeBrowserScrollbar Class Reference

Inheritance diagram for EyegazeBrowserScrollbar:



Collaboration diagram for EyegazeBrowserScrollbar:



Public Types

- enum **ScrollDir** { **UP** = 1, **DOWN** = -1 }

Public Member Functions

- override void OnClickEnter ()

  When clickable UI element is gazed upon enter this function.

Public Attributes

- GameObject **browser**

3.10.1 Member Function Documentation

3.10.1.1 OnClickEnter()

```
override void EyegazeBrowserScrollbar.OnClickEnter ( )  [virtual]
```

When clickable UI element is gazed upon enter this function.

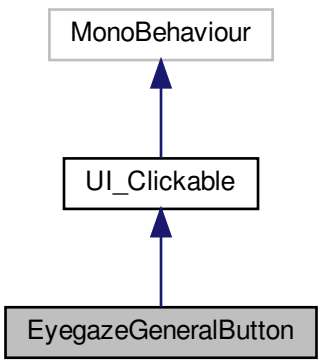
Implements UI_Clickable.

The documentation for this class was generated from the following file:

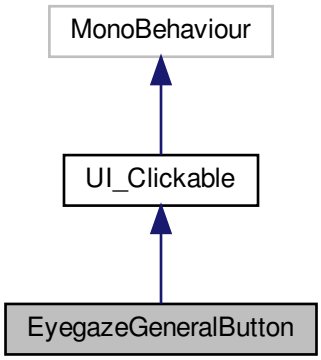
- EyegazeBrowserScrollbar.cs

3.11 EyegazeGeneralButton Class Reference

Inheritance diagram for EyegazeGeneralButton:



Collaboration diagram for EyegazeGeneralButton:



Public Member Functions

- override void OnClickEnter ()

  When clickable UI element is gazed upon enter this function.

Public Attributes

- Button **btn**

3.11.1 Member Function Documentation

3.11.1.1 OnClickEnter()

```
override void EyegazeGeneralButton.OnClickEnter ( )  [virtual]
```

When clickable UI element is gazed upon enter this function.

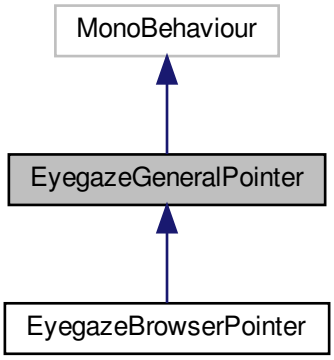
Implements UI_Clickable.

The documentation for this class was generated from the following file:

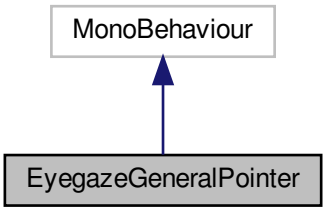
- EyegazeGeneralButton.cs

3.12 EyegazeGeneralPointer Class Reference

Inheritance diagram for EyegazeGeneralPointer:



Collaboration diagram for EyegazeGeneralPointer:



Public Member Functions

- void InitializeGazePointer ()

  Initializes the pointer
- virtual void UpdateGazePointerPosition (GazePoint gazePoint)

  Updates the pointer position in the world space, projects from screen plane to plane in world Smoothes pointer position on movement (Remove jagged movement)
- Vector3 ProjectToPlaneInWorld (GazePoint gazePoint)

  Uses camera and Gaze input to determine gaze location in world space
- Vector3 Smoothify (Vector3 point)

  Smoothes the movement of the pointer. Uses the previous position of the pointer with a smoothing factor to determine how to move it according to gaze location.
- void CheckClicker (Vector3 newPos, Vector3 oldPos, GazePoint gazePoint)

Uses a timer to determine a "Left mouse"-action. Uses a user-set timer to determine action by checking movement compared to old position. Then transforms gaze-position to Browser-view position, send mouse action at said position.

- virtual void **SendClickMessage** ()
- void **SetPointerAlpha** ()
- void **IncreasePointerAlpha** ()

Public Attributes

- float **visualizationDistance** = 10f
- Vector3 **pointerScale** = new Vector3 (1, 1, 1)
- float **filterSmoothingFactor** = 0.99f
- float **timeToClick** = 1.0f
- float **clickFeedbackTime** = 0.5f
- bool **clickActive** = false
- GameObject **mPointer**
- Camera **mCamera**
- GazePoint **gazePoint**

Protected Member Functions

- void **Update** ()

Protected Attributes

- bool **clickSent** = false

3.12.1 Member Function Documentation

3.12.1.1 CheckClicker()

```
void EyegazeGeneralPointer.CheckClicker (
            Vector3 newPos,
            Vector3 oldPos,
            GazePoint gazePoint )
```

Uses a timer to determine a "Left mouse"-action. Uses a user-set timer to determine action by checking movement compared to old position. Then transforms gaze-position to Browser-view position, send mouse action at said position.

Parameters

newPos	
oldPos	
gazePoint	

3.12.1.2 InitializeGazePointer()

```
void EyegazeGeneralPointer.InitializeGazePointer ( )
```

Initializes the pointer

3.12.1.3 ProjectToPlaneInWorld()

```
Vector3 EyegazeGeneralPointer.ProjectToPlaneInWorld (
            GazePoint gazePoint )
```

Uses camera and Gaze input to determine gaze location in world space

Parameters

gazePoint	

Returns

The gaze lovation in world space

3.12.1.4 Smoothify()

```
Vector3 EyegazeGeneralPointer.Smoothify (
            Vector3 point )
```

Smoothes the movement of the pointer. Uses the previous position of the pointer with a smoothing factor to determine how to move it according to gaze location.

Parameters

point	

Returns

Smoothed location of the gaze pointer

3.12.1.5 UpdateGazePointerPosition()

```
virtual void EyegazeGeneralPointer.UpdateGazePointerPosition (
            GazePoint gazePoint )  [virtual]
```

Updates the pointer position in the world space, projects from screen plane to plane in world Smoothes pointer position on movement (Remove jagged movement)

Parameters

gazePoint	

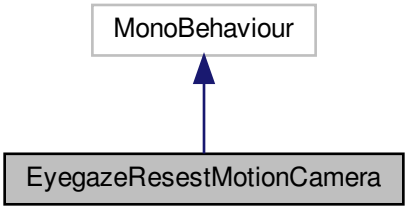
Reimplemented in EyegazeBrowserPointer.

The documentation for this class was generated from the following file:

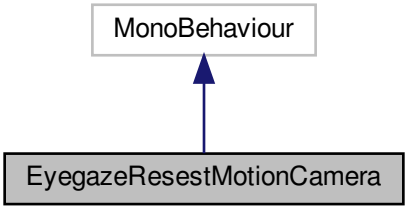
- EyegazeGeneralPointer.cs

3.13 EyegazeResestMotionCamera Class Reference

Inheritance diagram for EyegazeResestMotionCamera:



Collaboration diagram for EyegazeResestMotionCamera:

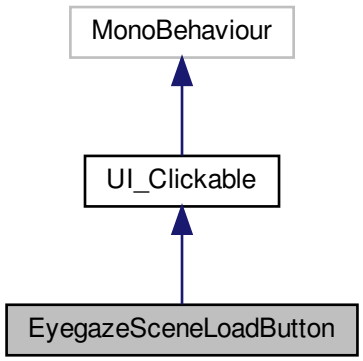


The documentation for this class was generated from the following file:

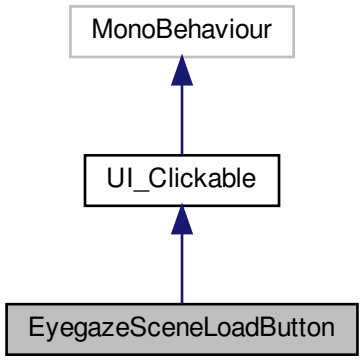
- EyegazeResestMotionCameraButton.cs

3.14 EyegazeSceneLoadButton Class Reference

Inheritance diagram for EyegazeSceneLoadButton:



Collaboration diagram for EyegazeSceneLoadButton:



Public Member Functions

- void Click ()

  Accesses scene loader's LoadScene(String s) with given string
- override void OnClickEnter ()

  When clickable UI element is gazed upon enter this function.

Public Attributes

- string **sceneToLoad**

3.14.1 Member Function Documentation

3.14.1.1 Click()

```
void EyegazeSceneLoadButton.Click ( )
```

Accesses scene loader's LoadScene(String s) with given string

3.14.1.2 OnClickEnter()

```
override void EyegazeSceneLoadButton.OnClickEnter ( )  [virtual]
```

When clickable UI element is gazed upon enter this function.

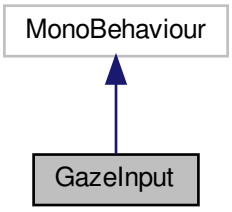
Implements UI_Clickable.

The documentation for this class was generated from the following file:

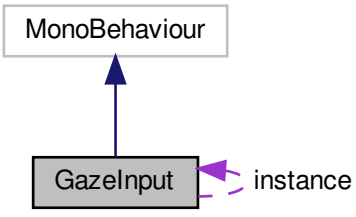
- EyegazeSceneloadButton.cs

3.15 GazeInput Class Reference

Inheritance diagram for GazeInput:



Collaboration diagram for GazeInput:



Static Public Attributes

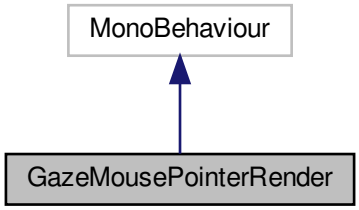
- static GazeInput **instance** = null

The documentation for this class was generated from the following file:

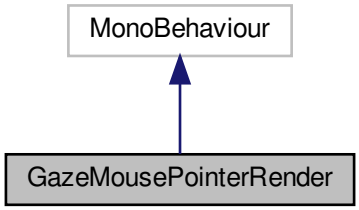
- GazeInput.cs

3.16 GazeMousePointerRender Class Reference

Inheritance diagram for GazeMousePointerRender:



Collaboration diagram for GazeMousePointerRender:

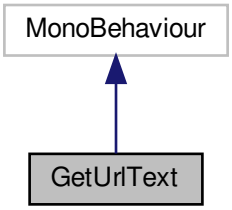


The documentation for this class was generated from the following file:

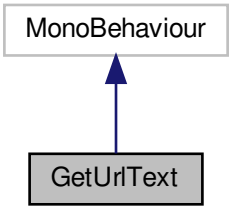
- GazeMousePointerRender.cs

3.17 GetUrlText Class Reference

Inheritance diagram for GetUrlText:



Collaboration diagram for GetUrlText:



Public Attributes

- TextMesh **meshUrlText**
- Text **urlText**

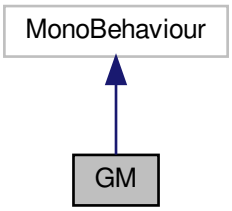
The documentation for this class was generated from the following file:

- GetUrlText.cs

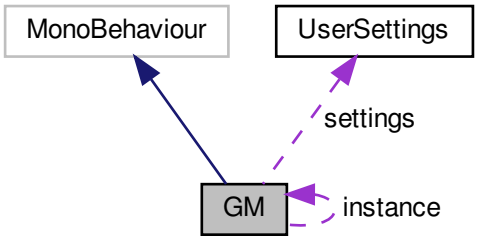
3.18 GM Class Reference

Top level manager, loads singletons and holds settings. Singleton class

Inheritance diagram for GM:



Collaboration diagram for GM:



Public Member Functions

- void LoadMainMenu ()

  Loads the main menu
- void ExitApplication ()

  Exits the application

Public Attributes

- float **clickTimer** = 1
- int **fontSize** = 14
- UserSettings **settings**
- bool **pauseable** = true

Static Public Attributes

- static GM **instance** = null

3.18.1 Detailed Description

Top level manager, loads singletons and holds settings. Singleton class

3.18.2 Member Function Documentation

3.18.2.1 ExitApplication()

```
void GM.ExitApplication ( )
```

Exits the application

3.18.2.2 LoadMainMenu()

```
void GM.LoadMainMenu ( )
```

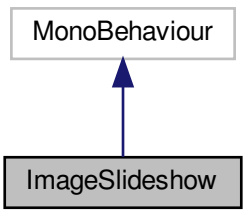
Loads the main menu

The documentation for this class was generated from the following file:

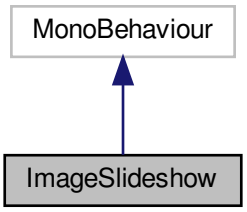
- GM.cs

3.19 ImageSlideshow Class Reference

Inheritance diagram for ImageSlideshow:



Collaboration diagram for ImageSlideshow:



Public Attributes

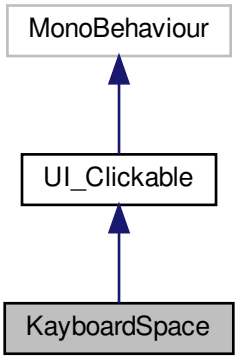
- List< Sprite > **bgImages**

The documentation for this class was generated from the following file:

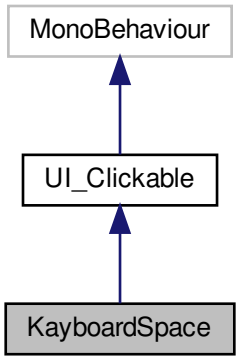
- ImageSlideshow.cs

3.20 KayboardSpace Class Reference

Inheritance diagram for KayboardSpace:



Collaboration diagram for KayboardSpace:



Public Member Functions

- override void OnClickEnter ()

  When clickable UI element is gazed upon enter this function.

3.20.1 Member Function Documentation

3.20.1.1 OnClickEnter()

```
override void KayboardSpace.OnClickEnter ( )  [virtual]
```

When clickable UI element is gazed upon enter this function.

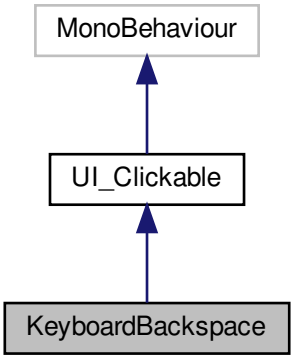
Implements UI_Clickable.

The documentation for this class was generated from the following file:

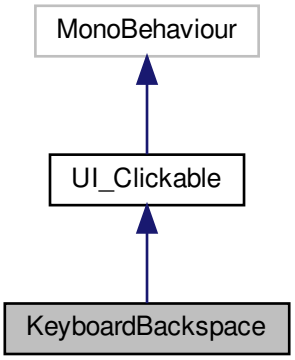
- KayboardSpace.cs

3.21 KeyboardBackspace Class Reference

Inheritance diagram for KeyboardBackspace:



Collaboration diagram for KeyboardBackspace:



Public Member Functions

- override void OnClickEnter ()

  When clickable UI element is gazed upon enter this function.

3.21.1 Member Function Documentation

3.21.1.1 OnClickEnter()

```
override void KeyboardBackspace.OnClickEnter ( )  [virtual]
```

When clickable UI element is gazed upon enter this function.

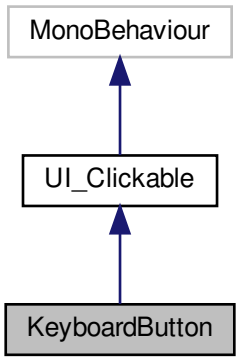
Implements UI_Clickable.

The documentation for this class was generated from the following file:

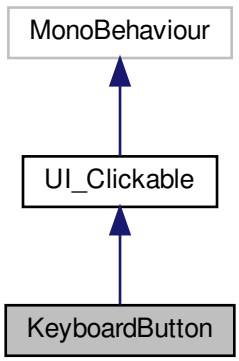
- KeyboardBackspace.cs

3.22 KeyboardButton Class Reference

Inheritance diagram for KeyboardButton:



Collaboration diagram for KeyboardButton:



Public Member Functions

- void Initialize (char character)

  Initializes a keyboard button with a given character.
- override void OnClickEnter ()

  When clickable UI element is gazed upon enter this function.

3.22.1 Member Function Documentation

3.22.1.1 Initialize()

```
void KeyboardButton.Initialize (
            char character )
```

Initializes a keyboard button with a given character.

Parameters

character	Keyboard character

3.22.1.2 OnClickEnter()

```
override void KeyboardButton.OnClickEnter ( )  [virtual]
```

When clickable UI element is gazed upon enter this function.

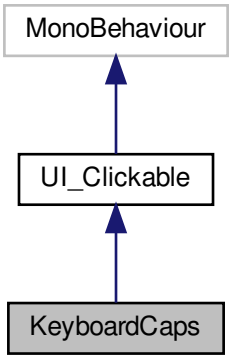
Implements UI_Clickable.

The documentation for this class was generated from the following file:

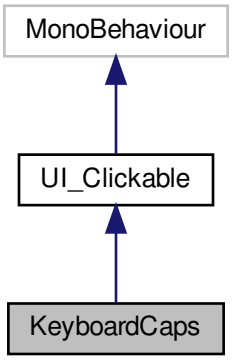
- KeyboardButton.cs

3.23 KeyboardCaps Class Reference

Inheritance diagram for KeyboardCaps:



Collaboration diagram for KeyboardCaps:



Public Member Functions

- override void OnClickEnter ()

  When clickable UI element is gazed upon enter this function.

3.23.1 Member Function Documentation

3.23.1.1 OnClickEnter()

```
override void KeyboardCaps.OnClickEnter ( )  [virtual]
```

When clickable UI element is gazed upon enter this function.

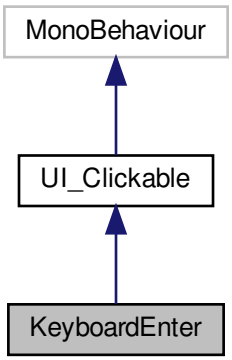
Implements UI_Clickable.

The documentation for this class was generated from the following file:

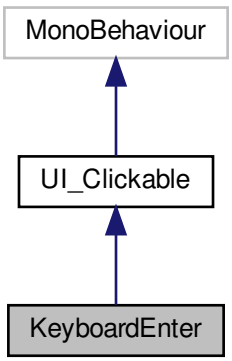
- KeyboardCaps.cs

3.24 KeyboardEnter Class Reference

Inheritance diagram for KeyboardEnter:



Collaboration diagram for KeyboardEnter:



Public Member Functions

- override void OnClickEnter ()

  When clickable UI element is gazed upon enter this function.

3.24.1 Member Function Documentation

3.24.1.1 OnClickEnter()

```
override void KeyboardEnter.OnClickEnter ( )  [virtual]
```

When clickable UI element is gazed upon enter this function.

Implements UI_Clickable.

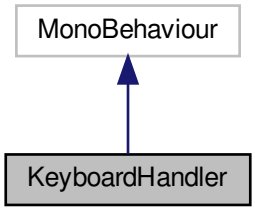
The documentation for this class was generated from the following file:

- KeyboardEnter.cs

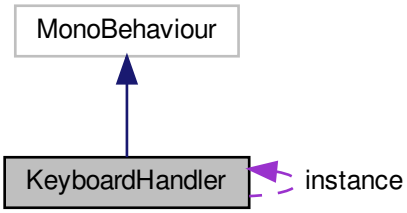
3.25 KeyboardHandler Class Reference

This is the keyboard class, it will handle creating, displaying, and functionality of the keyboard.

Inheritance diagram for KeyboardHandler:



Collaboration diagram for KeyboardHandler:



Public Member Functions

- void HideKeyboard ()

  Hides keyboard
- void ToggleKeyboard ()

  toggles keyboard on or off
- void Write (string key)

  Adds a string to the string currently being edited.
- void Backspace ()

  Removes a character from the string currently being edited.
- void ToggleShift ()

  Toggles shift mode.
- void ToggleCaps ()

  Toggles caps lock.
- void SetString (Text text)

  Changes which textbox is being edited.

Public Attributes

- float **distanceFromCenter** = 50

Static Public Attributes

- static KeyboardHandler **instance** = null

3.25.1 Detailed Description

This is the keyboard class, it will handle creating, displaying, and functionality of the keyboard.

3.25.2 Member Function Documentation

3.25.2.1 Backspace()

```
void KeyboardHandler.Backspace ( )
```

Removes a character from the string currently being edited.

3.25.2.2 HideKeyboard()

```
void KeyboardHandler.HideKeyboard ( )
```

Hides keyboard

3.25.2.3 SetString()

```
void KeyboardHandler.SetString (
            Text text )
```

Changes which textbox is being edited.

Parameters

text	a unity Text object to be edited.

3.25.2.4 ToggleCaps()

```
void KeyboardHandler.ToggleCaps ( )
```

Toggles caps lock.

3.25.2.5 ToggleKeyboard()

```
void KeyboardHandler.ToggleKeyboard ( )
```

toggles keyboard on or off

3.25.2.6 ToggleShift()

```
void KeyboardHandler.ToggleShift ( )
```

Toggles shift mode.

3.25.2.7 Write()

```
void KeyboardHandler.Write (
            string key )
```

Adds a string to the string currently being edited.

Parameters

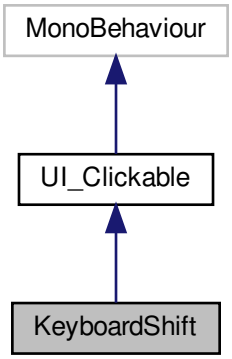
key	The string to be added

The documentation for this class was generated from the following file:

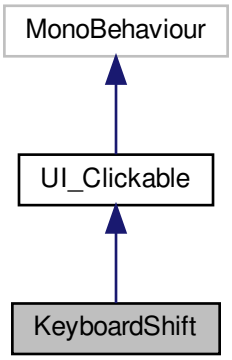
- KeyboardHandler.cs

3.26 KeyboardShift Class Reference

Inheritance diagram for KeyboardShift:



Collaboration diagram for KeyboardShift:



Public Member Functions

- override void OnClickEnter ()

  When clickable UI element is gazed upon enter this function.

3.26.1 Member Function Documentation

3.26.1.1 OnClickEnter()

```
override void KeyboardShift.OnClickEnter ( )  [virtual]
```

When clickable UI element is gazed upon enter this function.

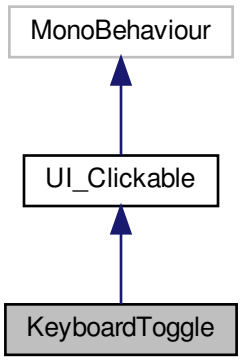
Implements UI_Clickable.

The documentation for this class was generated from the following file:

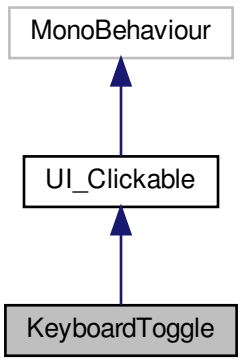
- KeyboardShift.cs

3.27 KeyboardToggle Class Reference

Inheritance diagram for KeyboardToggle:



Collaboration diagram for KeyboardToggle:



Public Member Functions

- override void OnClickEnter ()

  When clickable UI element is gazed upon enter this function.

3.27.1 Member Function Documentation

3.27.1.1 OnClickEnter()

```
override void KeyboardToggle.OnClickEnter ( )  [virtual]
```

When clickable UI element is gazed upon enter this function.

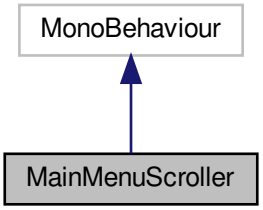
Implements UI_Clickable.

The documentation for this class was generated from the following file:

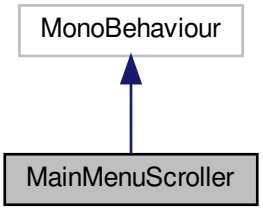
- KeyboardToggle.cs

3.28 MainMenuScroller Class Reference

Inheritance diagram for MainMenuScroller:



Collaboration diagram for MainMenuScroller:



Public Member Functions

- void scrollUp ()

  Changes content of panel by moving backwards by 1 in array/vector

- void scrollDown ()

  Changes content of panel by moving forwards by 1 in array/vector

Public Attributes

- List< Button > **funcButtons** = new List<Button>()

3.28.1 Member Function Documentation

3.28.1.1 scrollDown()

```
void MainMenuScroller.scrollDown ( )
```

Changes content of panel by moving forwards by 1 in array/vector

3.28.1.2 scrollUp()

```
void MainMenuScroller.scrollUp ( )
```

Changes content of panel by moving backwards by 1 in array/vector

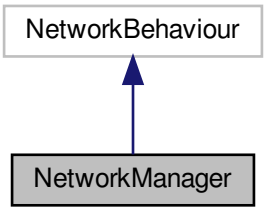
The documentation for this class was generated from the following file:

- MainMenuScroller.cs

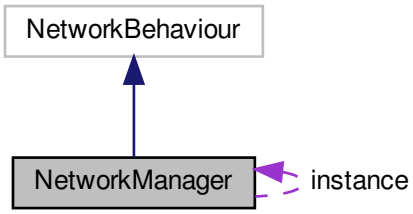
3.29 NetworkManager Class Reference

Handles communication with server, singleton class.

Inheritance diagram for NetworkManager:



Collaboration diagram for NetworkManager:



Public Member Functions

- WWW Post (string postURL, string jsonString)

  Return page information with json request. Takes in URL to send the request to, and returns the result as a page.

Static Public Attributes

- static NetworkManager **instance** = null

3.29.1 Detailed Description

Handles communication with server, singleton class.

3.29.2 Member Function Documentation

3.29.2.1 Post()

```
WWW NetworkManager.Post (
            string postURL,
            string jsonString )
```

Return page information with json request. Takes in URL to send the request to, and returns the result as a page.

Parameters

postURL	The URL to Poll.
jsonString	Information to send. (use JsonUtility.ToJson(string)) to generate this easily

Returns

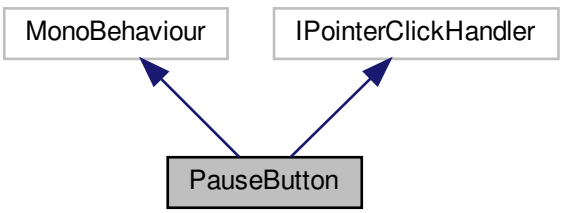
Page information from request (UnityEngine.WWW)

The documentation for this class was generated from the following file:

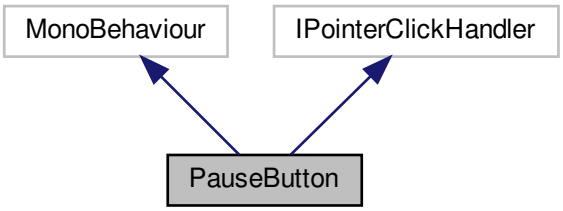
- NetworkManager.cs

3.30 PauseButton Class Reference

Inheritance diagram for PauseButton:



Collaboration diagram for PauseButton:



Public Member Functions

- void OnPointerClick (PointerEventData eventData)

  Activates pause menu panel, sets in-game timescale to 0

Public Attributes

- GameObject **pauseMenu**

3.30.1 Member Function Documentation

3.30.1.1 OnPointerClick()

```
void PauseButton.OnPointerClick (
            PointerEventData eventData )
```

Activates pause menu panel, sets in-game timescale to 0

Parameters

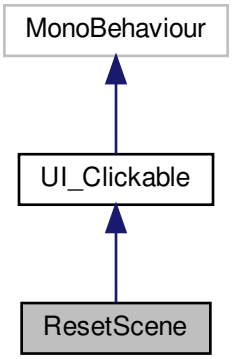
eventData	

The documentation for this class was generated from the following file:

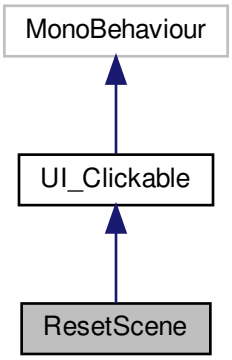
- PauseButton.cs

3.31 ResetScene Class Reference

Inheritance diagram for ResetScene:



Collaboration diagram for ResetScene:



Public Member Functions

- override void OnClickEnter ()

  When clickable UI element is gazed upon enter this function.

3.31.1 Member Function Documentation

3.31.1.1 OnClickEnter()

```
override void ResetScene.OnClickEnter ( )  [virtual]
```

When clickable UI element is gazed upon enter this function.

Implements UI_Clickable.

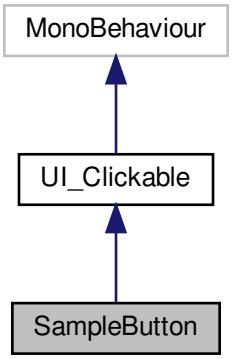
The documentation for this class was generated from the following file:

- ResetScene.cs

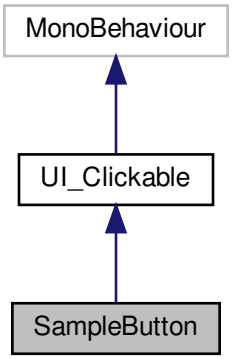
3.32 SampleButton Class Reference

Example button.

Inheritance diagram for SampleButton:



Collaboration diagram for SampleButton:



Public Member Functions

- override void OnClickEnter ()

  Button function here.

3.32.1 Detailed Description

Example button.

3.32.2 Member Function Documentation

3.32.2.1 OnClickEnter()

```
override void SampleButton.OnClickEnter ( )  [virtual]
```

Button function here.

Implements UI_Clickable.

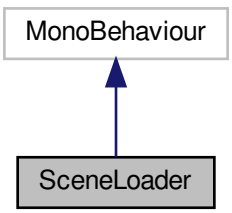
The documentation for this class was generated from the following file:

- SampleButton.cs

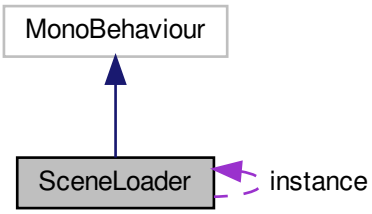
3.33 SceneLoader Class Reference

Handles loading between scenes. Singleton Class

Inheritance diagram for SceneLoader:



Collaboration diagram for SceneLoader:



Public Member Functions

- void LoadScene (int sceneIndex)

  Loads a new scene by scene index (as listed in unity build settings)
- void LoadScene (string sceneName)

  Loads a new scene by scene name
- void ResetScene ()

  Reloads the currently active scene.

Static Public Attributes

- static SceneLoader **instance** = null

3.33.1 Detailed Description

Handles loading between scenes. Singleton Class

3.33.2 Member Function Documentation

3.33.2.1 LoadScene() [1/2]

```
void SceneLoader.LoadScene (
            int sceneIndex )
```

Loads a new scene by scene index (as listed in unity build settings)

Parameters

sceneIndex	Index of Scene to be loaded

3.33.2.2 LoadScene() [2/2]

```
void SceneLoader.LoadScene (
            string sceneName )
```

Loads a new scene by scene name

Parameters

sceneIndex	Name of Scene to be loaded

3.33.2.3 ResetScene()

```
void SceneLoader.ResetScene ( )
```

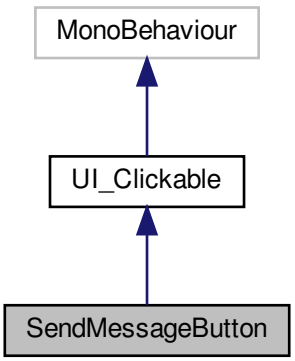
Reloads the currently active scene.

The documentation for this class was generated from the following file:

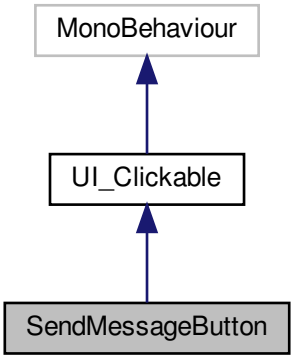
- SceneLoader.cs

3.34 SendMessageButton Class Reference

Inheritance diagram for SendMessageButton:



Collaboration diagram for SendMessageButton:



Public Member Functions

- void **Awake** ()
- override void OnClickEnter ()

  When clickable UI element is gazed upon enter this function.

3.34.1 Member Function Documentation

3.34.1.1 OnClickEnter()

```
override void SendMessageButton.OnClickEnter ( )  [virtual]
```

When clickable UI element is gazed upon enter this function.

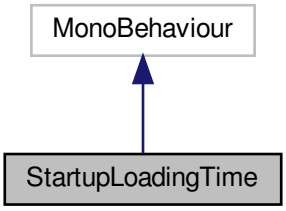
Implements UI_Clickable.

The documentation for this class was generated from the following file:

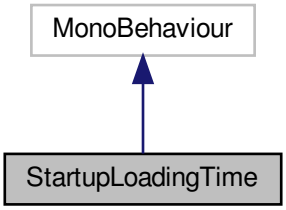
- SendMessageButton.cs

3.35 StartupLoadingTime Class Reference

Inheritance diagram for StartupLoadingTime:



Collaboration diagram for StartupLoadingTime:

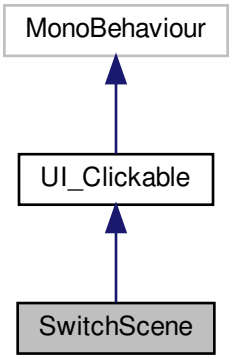


The documentation for this class was generated from the following file:

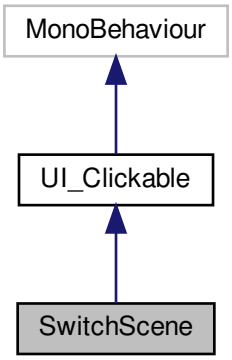
- StartupLoadingTime.cs

3.36 SwitchScene Class Reference

Inheritance diagram for SwitchScene:



Collaboration diagram for SwitchScene:



Public Member Functions

- override void OnClickEnter ()

  When clickable UI element is gazed upon enter this function.

3.36.1 Member Function Documentation

3.36.1.1 OnClickEnter()

```
override void SwitchScene.OnClickEnter ( )  [virtual]
```

When clickable UI element is gazed upon enter this function.

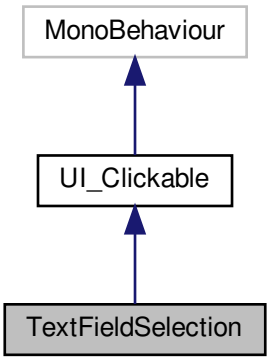
Implements UI_Clickable.

The documentation for this class was generated from the following file:

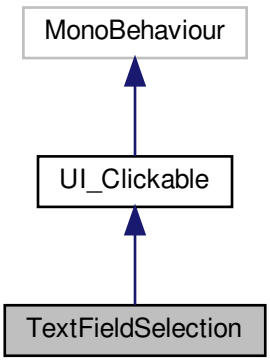
- SwitchScene.cs

3.37 TextFieldSelection Class Reference

Inheritance diagram for TextFieldSelection:



Collaboration diagram for TextFieldSelection:



Public Member Functions

- override void OnClickEnter ()

  When clickable UI element is gazed upon enter this function.

3.37.1 Member Function Documentation

3.37.1.1 OnClickEnter()

```
override void TextFieldSelection.OnClickEnter ( )  [virtual]
```

When clickable UI element is gazed upon enter this function.

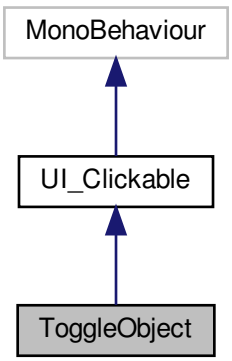
Implements UI_Clickable.

The documentation for this class was generated from the following file:

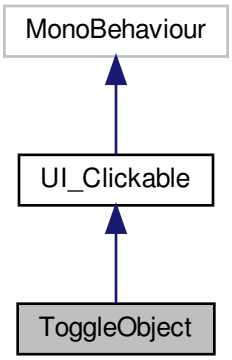
- TextFieldSelection.cs

3.38 ToggleObject Class Reference

Inheritance diagram for ToggleObject:



Collaboration diagram for ToggleObject:



Public Member Functions

- override void OnClickEnter ()

  When clickable UI element is gazed upon enter this function.

3.38.1 Member Function Documentation

3.38.1.1 OnClickEnter()

```
override void ToggleObject.OnClickEnter ( )  [virtual]
```

When clickable UI element is gazed upon enter this function.

Implements UI_Clickable.

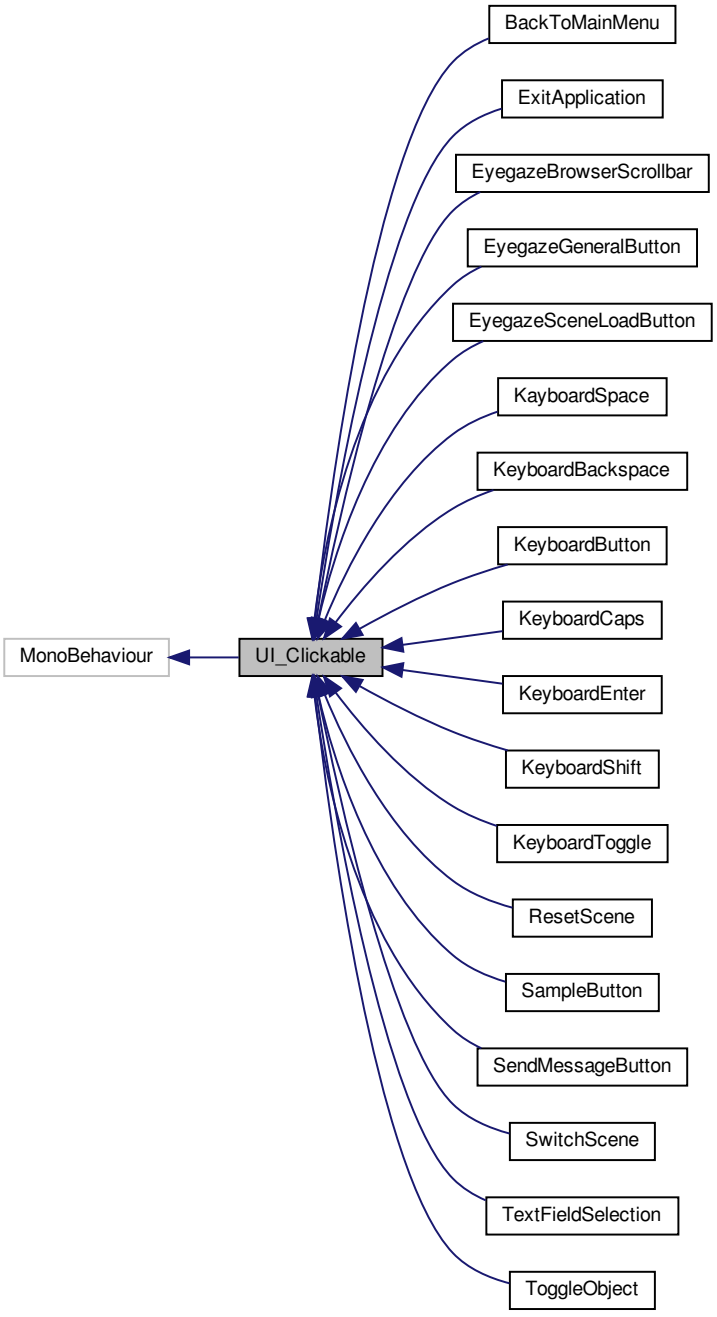
The documentation for this class was generated from the following file:

- ToggleObject.cs

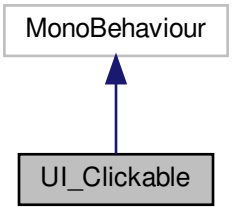
3.39 UI_Clickable Class Reference

Parent class for any clickable/selectable element in the ui.

Inheritance diagram for UI_Clickable:



Collaboration diagram for UI_Clickable:



Public Member Functions

- abstract void OnClickEnter ()
  When clickable UI element is gazed upon enter this function.

3.39.1 Detailed Description

Parent class for any clickable/selectable element in the ui.

3.39.2 Member Function Documentation

3.39.2.1 OnClickEnter()

```
abstract void UI_Clickable.OnClickEnter ( )  [pure virtual]
```

When clickable UI element is gazed upon enter this function.

Implemented in EyegazeBrowserScrollbar, EyegazeGeneralButton, EyegazeSceneLoadButton, KeyboardButton, SendMessageButton, SampleButton, SwitchScene, ToggleObject, KayboardSpace, KeyboardBackspace, KeyboardEnter, KeyboardToggle, BackToMainMenu, ExitApplication, ResetScene, KeyboardCaps, KeyboardShift, and TextFieldSelection.

The documentation for this class was generated from the following file:

- UI_Clickable.cs

3.40 UserSettings Class Reference

Serializable object for XML I/O, holds user settings.

Public Member Functions

- UserSettings ()

  creates a usersettings object with default values
- UserSettings (float clickTimer, float clickResetTimer, int fontSize, float keyboardRadius, float uiScale, string username)

  creates a usersettings object with given values

Public Attributes

- float **clickTimer**
- float **clickResetTimer**
- int **fontSize**
- float **keyboardRadius**
- float **uiScale**
- string **username**

3.40.1 Detailed Description

Serializable object for XML I/O, holds user settings.

3.40.2 Constructor & Destructor Documentation

3.40.2.1 UserSettings() [1/2]

```
UserSettings.UserSettings ( )
```

creates a usersettings object with default values

3.40.2.2 UserSettings() [2/2]

```
UserSettings.UserSettings (
            float clickTimer,
            float clickResetTimer,
            int fontSize,
            float keyboardRadius,
            float uiScale,
            string username )
```

creates a usersettings object with given values

Parameters

clickTimer	time for a click to register
clickResetTimer	reset cooldown for clickable objects
fontSize	ui fontsize
keyboardRadius	keyboard distance
uiScale	scale of UI
username	online username

The documentation for this class was generated from the following file:

- XMLManager.cs

3.41 XMLManager Class Reference

Handles XML parsing, and save/load of XML files.

Public Member Functions

- void WriteXmlFile (string fileName, UserSettings settings)
  Writes a usersettings object to an xml files
- UserSettings LoadXmlFile (string fileName)
  Loads usersettings from a file

3.41.1 Detailed Description

Handles XML parsing, and save/load of XML files.

3.41.2 Member Function Documentation

3.41.2.1 LoadXmlFile()

```
UserSettings XMLManager.LoadXmlFile (
            string fileName )
```

Loads usersettings from a file

Parameters

fileName	file to read from

Returns

usersettings loaded

3.41.2.2 WriteXmlFile()

```
void XMLManager.WriteXmlFile (
            string fileName,
            UserSettings settings )
```

Writes a usersettings object to an xml files

Parameters

fileName	name of file to be created
settings	usersettings to be saved

The documentation for this class was generated from the following file:

- XMLManager.cs

Index