Möjliggörande av icke-invasiva ögonstyrda interaktioner i befintliga datorspel

KONSTENIUS, CHRISTIAN
Möjliggörande av icke-invasiva ögonstyrda interaktioner i befintliga datorspel

Enabling non-invasive gaze interactions in existing computer games

Konstenius, Christian

E-postadress vid KTH: ccon@kth.se

Exjobb i: Människa-datorinteraktion

Handledare: Hedman, Anders

Examinator: Lantz, Ann

Uppdragsgivare: Tobii Technology
Enabling non-intrusive gaze interactions in existing computer games

Abstract

Eye tracking have been around for many years now, but until recently the technology have been expensive and the performance too low for the mass market to pay any attention. But times are changing and the cost of eye trackers is going down and performance is getting better. Eye tracking will soon start appearing in more and more homes around the planet.

With the advent of eye trackers in people’s homes, it is time to start thinking about what users would want to use eye tracking for. A popular field that comes to mind is gaming. Adding eye tracking to games could change gaming significantly. Experiences could be enhanced and new genres created.

A deep integration with the games would allow eye tracking to blend smoothly with the game; creating a natural and immersive experience. However, it will take a while for the game developers to make this come true. In the meantime something else is needed to fill the gap between now, when no game had eye tracking support, and the future when all games will have a deeply integrated eye tracking interaction.

The product we came up with is a tool for non-intrusively integrating eye tracking into existing games. This means that the tool is able to add eye tracking support without the need to modify the game in any way.

Möjliggörande av icke-invasiva ögonstyrda interaktioner i befintliga datorspel

Sammanfattning

Ögonstyrning har funnits i många år men nyligen har tekniken varit för dyr och prestandan för låg, för att vanliga konsumenter ska visa något intresse. De tiderna håller på att ändras och priset för eye trackers sjunker och prestandan ökar. Ögonstyrning kommer snart att börja dyka upp i allt fler hem runt om i världen.

I och med ankomsten av ögonstyrning i folks hem så är det hög tid att börja fundera kring vad användarna kommer att vilja använda ögonstyrning till. Ett populärt område idag är datorspel. Att lägga till ögonstyrning i datorspel skulle kunna ändra spelen markant, upplevelser kan bli förstärkta och hela nya genrer av spel skulle kunna skapas.

En djup integration med spelen skulle göra det möjligt för ögonstyrning att smälta in i spelen och då skapa en naturlig och fängslande upplevelse. Men det kommer ta ett tag för spelutvecklarna att göra detta möjligt. Under tiden så behövs något annat för att fylla glappet mellan nu då inga spel har stöd för ögonstyrning och framtiden då alla spel kommer ha en djup integration.

Resultatet av examensarbetet var ett verktyg för att på ett icke-invasivt sätt, integrera ögonstyrning i befintliga spel. Detta innebär att verktyget kan lägga till stöd för ögonstyrning utan att behöva modifiera spelen på något sätt.
Preface

This master thesis has been a very interesting, fun and overall great experience for me. I feel like I want to recognize and give my thanks to a number of people who have had an important role in this thesis.

- **Fredrik Lind** – I want to give Fredrik a big “thank you” for giving me this opportunity. Without him this project wouldn’t have been possible. He was my supervisor, project leader and coworker. This project was his idea from the start. He started this project before I started at Tobii and will continue to lead this project in the future.

- **Tobii Gaming Team** – I also want to thank the entire gaming team, which I was a part of. They all worked with this project in one way or another and together we produced a great product.

- **Tobii OEM Software R&D** – I want to thank the entire OEM Software R&D. They were the ones I’ve spent most of my time with during my time at Tobii. They have all helped me in one way or another and they made my time at Tobii great.

- **Anders Hedman** – Anders was my supervisor at KTH who have supported and helped me in my work through this masters’ thesis.

- **Family and Friends** – Lastly I want to thank my family and friends who have helped me directly or indirectly during this thesis.
# Table of Contents

Glossary ........................................................................................................................................... 1  
Background..................................................................................................................................... 2  
  Problem........................................................................................................................................ 2  
  Purpose......................................................................................................................................... 4  
  Structure of the application ........................................................................................................... 7  
Theory ........................................................................................................................................... 12  
  The movements of the eye ............................................................................................................. 12  
  Unusual ways of gaze interaction .................................................................................................. 13  
  Eye tracking methods ..................................................................................................................... 13  
  Moral and ethical aspects of eye tracking ..................................................................................... 16  
Methodology .................................................................................................................................. 18  
  Personas ....................................................................................................................................... 18  
  Initial sketches ............................................................................................................................... 20  
  Prototype ..................................................................................................................................... 22  
  User tests ..................................................................................................................................... 25  
  Interview with professional mapper .............................................................................................. 28  
  Final sketches ............................................................................................................................... 28  
Results .......................................................................................................................................... 29  
  Initial sketches ............................................................................................................................... 29  
  User tests ..................................................................................................................................... 34  
  Other results ................................................................................................................................. 38  
  Interview with a professional mapper .......................................................................................... 40  
  Final sketches and prototype ....................................................................................................... 41  
Discussion ..................................................................................................................................... 50  
Conclusions .................................................................................................................................... 52  
Bibliography .................................................................................................................................. 54  
Appendix A - Preliminary sketches ................................................................................................. 55  
Appendix B - Iterations of the selected sketch .............................................................................. 59
Glossary

**Application Programming Interface (API)** - A specification about how software components should interact with each other.

**Behaviors** - A set of actions the application performs based on input from the eye tracker, mouse and keyboard.

**Controller** - A type of special key/button event, used to activate and cancel behaviors.

**Calibration** - A procedure required to optimize the eye tracker. In practice this is done by looking at a moving dot for a couple of seconds. This data is used to create a personal model of the users eye.

**DirectX** - A collection of APIs used in graphics and game programming.

**Eye Tracker** - The physical hardware used in eye tracking and gaze interaction.

**Eye Tracking** - The actual process of tracking eyes in 3D space. The word is also to describe the act of tracking the gaze e.g. on a computer monitor.

**Feature** - Small area of the screen used for identifying the current scene in a game. A good feature is a static part of the graphical user interface which only appears in a particular scene.

**First-Person-Shooter (FPS)** - A popular genre of games where you see through the eyes of the game character, walk around in a 3D world and shoot enemies.

**Gaze Point** - The point you are looking e.g. on the computer monitor.

**Glint** - A highlight in the eyes caused by bright lights. In eye tracking it is used to calculate the rotation of the eye ball.

**Illuminator** - A micro projector on the eye tracker who has the purpose of creating glints (highlights) on the cornea. The position of the illuminator decides if the pupil appears black or white; this can be used to help track the pupil by making in more visible.

**Low Pass Filter** - A common category of filters used to eliminate high frequency noise and to smooth a noisy signal. A simple example is the moving average filter which simply takes the average of the last couple of samples as the output.

**Mapping** - A collection of scenes, behaviors, features etc. used to define the interaction in the game.

**Model View ViewModel (MVVM)** - An architectural pattern used in graphical applications which separates the graphical user interface from the underlying data.

**Namespace** - A way to group a series of software component with functionality used in the same context.

**OpenGL** - A multi-platform collection of APIs used in graphics and game programming.

**Pupil Centre Corneal Reflection (PCCR)** - A popular method for eye tracking. It uses relation between the position of the pupil and the position of the corneal reflections (from the illuminators on the eye tracker) to calculate the gaze direction.

**Scene** - A section of the game where you want to use eye tracking to enhance the experience. A scene contains a number of behaviors which can be activated when the scene is active. A scene is active if its features match the image received from the game.

**Source Integration** – The act of modifying the source code of the game to add functionality.

**Triggers** - A set of buttons or combinations of buttons and their states (up, down, clicked, etc.). Triggers are used to activate controllers which in turn activate the behaviors.
In the beginning of 2012, the leading company in the area of eye tracking, Tobii Technology was searching for someone to help them with the design and implementation of a particular software application. They were looking to expand their business into the gaming scene and wanted to create an application that could cater to gamers.

What they had in mind was an application that would allow players to play existing games using their eyes. They didn’t want game companies to have to modify the actual source code of their games; instead Tobii wanted their application to lie as a layer on top of the game. By doing it this way the application would work for almost any game and there would be no need to contact all the major game companies.

To be able to decide when and how to interact with a specific game with your eyes, there is a need to have some kind of tool that would allow a user to apply different “behaviors” to different parts of a game. These behaviors are simply actions that define how to interact with the game with your eyes and what should happen when you do. Two very simple examples of behaviors could be: look at top of the screen to move forward or hold the spacebar to enlarge the mini-map, when you release the spacebar; teleport the game character to where you were looking.

By combining a number of these behaviors together the user could create what could be called a “profile” for a game. Tobii aims to create default profiles for a large number of game titles themselves using the tool, but the tool should also be easy enough for the users to be able to create their own profiles for games.

My role in the project at Tobii will be to design and implement both the tool used to create the profiles and the entire user experience. Besides me there are one more developer (my project leader) whose role will be to create the underlying engine which handles all the interaction with the games. My role can also be described, slightly simplified, as being the one that design and implement the editor which will be used to create the profiles which in turn will be used in the engine, created by my project leader. I will create all the graphical user interfaces and presentation logic while my project leader will handle most of the business logic.

In this thesis I will explore how to best design a user interface that satisfies both target groups requirements while still providing a pleasant user experience. Another question is, how is it possible to show the user what can be done in the designer/editor. Also when during the game, what can be interacted with using the eyes?

By combining these questions I arrive at the main question of this thesis is:

*How can I show the users the possibilities and limitations of the resulting application, while satisfying the needs of both target groups?*

As hinted in the previous section, the application will be divided into two parts:

- A standalone designer/editor that allow users to create unique profiles for games. This editor will be used to create new or modify existing profiles. This includes, but is not limited to, adding and modifying behaviors.

- A service that will run as a layer on-top of the game. This service will be used for all the input and output to and from the game. One major use of this service will be to display feedback and hints to the user. Without this feedback it would be hard for the user to know how and what he can interact with, using his eyes.
Both parts are important, but the majority of the effort in this project will be focused on designing the designer/editor. The reason for this is that the designer is by far the largest part of the two and will require a lot of attention to be executed well.

The Designer

The designer has three main purposes:

- **Manage profiles.** The user should be able to create new or edit existing profiles for games.

- **Manage scenes in the game.** A scene is a part of the game that the user should be able to interact with. Two examples of scenes would be the main menu and when a certain weapon is selected in a first-person-shooter.

- **Manage behaviors.** The designer should enable users to add/remove/modify behaviors in a scene.

Since Tobii wants to create a large number of default profiles themselves, there are two preliminary target groups for the designer.

The primary target group is regular users/gamers that want to create a profile for just one or two of their favorite games. These users will have no idea what to do when they open the designer and will only receive the help that the designer provides. This category of users primary needs are ease of use and simplicity.

The secondary target group is the employees at Tobii that will get a thorough introduction to the designer/editor. They will also be told exactly how everything in the designer works and what they need to do to successfully create a working profile for a game. Since they will be creating possibly hundreds of default profiles for various games, their primary needs will be speed and reliability.

This means that the designer/editor will be used both by the industry that has a great interest in speed and efficiency and by user at home who has a larger need for ease of use. One of the main problems will be, how should the Graphical User Interface (GUI) be designed and what is needed to satisfy both target group’s needs?

The Service

The service is the part of the application that runs when the user is playing a game. It is seemingly invisible to the user until there is a need to send some kind of feedback to the user from the application. The feedback can be in the form of text and images that can be displayed on top of the game. It might also be possible to send feedback in the form of audio. The main focus will lie on the designer but will keep the service in mind even if there won’t be any time to design and test the service.

The service is also in charge of intercepting and simulating mouse and keyboard actions.

One of the main challenges of designing the graphical part of the service will be: when should the service send feedback to the user and how should the feedback look? How should we explain to the user what he/she can do using his/her eyes in a specific game?
Purpose

Historically eye tracking have primarily been used to either understand human behavior or to improve user interfaces for handicapped people. As an assistive technology for handicapped people who have lost their use of their hands, eye tracking is usually used as a single-mode interface where the gaze is the only input available for the user (Tobii, 2013).

Eye tracking has also been used to understand behavior. It is useful to study the gaze since the eyes usually direct towards the information being processed. This has been used for analyzing and optimizing advertisements and interaction design (Tobii, 2013).

The main reason this would be an interesting area to study is because the resulting application will be the first of its kind. There is no similar software on the market today, especially not focused on gaming. The technology and hardware is finally cheap and good enough for mass market, making the results from this masters’ thesis especially important.

It will be a great challenge to design a user interface for the designer, since there aren’t any similar applications on the market to get inspiration from. The vast majority of the users in the target groups are also completely new to eye tracking, this means that there exist no officially established ways of interactions among the population. I will however base many of my design choices and the types of interactions I choose to implement, on previous experiences from the employees at Tobii.

The employees at Tobii are probably the greatest source of knowledge of good eye tracking interaction patterns, methodologies and knowledge that one can find. They have been the world leader in their field for many years now and a very large part of Tobii’s employees is dedicated solely to research and development in everything from hardware and software to user experience and human-computer interaction.

The role of the application in the market

The resulting application from this project will act as the technological bridge between today and the future. Today there are basically no eye tracking enabled games on the market and in the future when eye tracking will be widespread and the majority of all the games created will support eye tracking (see Figure 1). This is of course not guaranteed, but the technology is extremely promising and it is finally mature enough to attract demanding consumer groups such as gamers.

In the future the eye tracking will be integrated a lot deeper in the games. This is ensured by Tobii making development kits available and creating close connections with numerous game developers. This would allow the game developers to create rich and deep experiences for the end user. In the gaming team at Tobii there has been a lot of work trying to figure out how to create the best experience when combining games and eye tracking. Our philosophy about how eye tracking should (to create the best experience) and will be used in computer games in the future is as follows; the best experience will be the one when the user can play the game without thinking or even knowing about the eye tracking. This means that eye tracking shouldn’t be used to e.g. steer the car in a driving game. Instead eye tracking should be used for more subtle things such as causing virtual characters to talk to you if you establish real eye contact with
them. The user should never have to think about how to move his eyes or where to look to perform a certain task. The gaming team at Tobii has concluded, through experimentation, that the important thing for future games is to have a lot of these small and subtle interactions for the user. This will deepen the immersion and enhance the experience for the user which in the end is exactly what we want.

Figure 2 - The difference between integrating the eye tracking into the game or not.

The application we are developing will however not allow for many of these small and subtle interactions. The reason for this is that we want to create a tool for the end users and not an application programming interface (API) or development kit for game programmers. Figure 2 illustrates the difference between using Tobii’s API and using our application. This means that we want the end users themselves to be able to add eye tracking support to their favorite games without the need of any programming knowledge. Because of this, what we can affect in the game will be limited.

Limitations

There is no limit to how big this project could become, so we have decided on a number of limitations to reduce the size of the project and to make it work with most games. These limitations come from both technical problems, administrative problems and various decisions. The major limitations will be:

- The software won’t use source integration of any kind.
- The application will not act as substitute for mouse and keyboard.
- I will focus on the user interface and the overall workflow in this thesis and not on the specifics about the programming.
- The software should not affect the performance of the game noticeably.

By not using source integration I greatly limit what the software will be able to do. Source integration means that we change the code of the actual game or use some kind of API for integrating our application into the game.

With source integration a user would potentially be able to do anything he wants in the game using his eyes. The user would for example be able to make a game character angry if he looks at the character’s wife for too long or highlight enemy planes in a flight simulator just by looking at them.

To do any of this I would have to access the actual code for the game and modify it to add the desired actions and interactions. This means that for every single game, I would have to manually program every change I want to make to the game. This would of course be very powerful, but it would very time consuming and it would be very hard to make a universal tool that would work with any game on the market. If the game requires that level of depth and
Background

complexity when integrating eye-tracking, it would be better if the game companies themselves added the support for eye tracking from the start.

An interesting approach that would be better for this project is to allow our application to lie as a separate layer on top of the game (see figure 3). By doing it this way, the application will act as a kind of barrier, intercepting the input and output devices of the computer.

![Figure 3 - The possibilities of the different ways to integrate eye tracking in games.](image)

By not integrating our application with the game, we can’t access things that are specific to the game such as information about objects in a dynamic 3D scene.

What the application will be able to do is:

- Receive a live image from the game
- Simulate keyboard and mouse actions
- Display text and images while playing the game (read: give feedback)
- Scan the memory for known data, e.g. score, player name, etc. This is done by looking in known locations in the application memory for unknown data or by searching the entire application memory for known data. This data can then be read and/or modified.
- Manipulate the graphic matrices in DirectX/OpenGL. This allows the application to e.g. move the camera (in the game) independently of the player’s position.

With this in combination with eye tracking the application will be able to do things like: click the mouse where the user is looking, create special menus that can be interacted with by looking at them, automatically hold down a keyboard key when you look at a specified area of the screen, etc. By combing these five options in different ways, the application will be able to facilitate many new ways of interaction in existing computer games. It will also allow me to create a universal application that will be able to add these new ways of interaction.

To limit the size of the project further it was decided that the application should not act as a substitute for keyboard and mouse. This means that the game should be playable as normal, but enhanced by the application. This in turn means that the application won’t have to be able to substitute the interaction in every single part of the game. Instead the application will only have to enhance the interaction of certain parts. An example could be to make every menu in a keyboard heavy computer game work with your eyes, but the rest of the game plays as normal using the keyboard.

To add another technical limitation; the application should not affect the performance of the game. This is important since a lot of gamers find the frame rate of the game very important. The application shouldn’t use processor heavy code and must keep things simple. For example the application can’t use advanced image processing since the application is not allowed to use that much processor power. This means that the application should avoid doing demanding image processing tasks such as tracking moving objects on the screen and instead focus on simpler tasks such as detecting the presence of static objects on the screen.

The final limitation has to do with this thesis: I will only talk about the concepts, ideas and results regarding the user experience and graphical user interface. I won’t go into depth in the actual code for the application, the problems that arose when developing this application nor the solutions I found to these problems.
Structure of the application

To understand my initial design choices, there is a need to explain the basic structure and functionality of the application. There are a number of key concepts and features that need explanation to understand. The most vital of which are scenes, behaviors and features.

Scenes

The first concept that might need some explanation is the notion of scenes. A scene is a certain view/state of the game. This is a unique part of the game where the user want to enable some kind of interaction with the eye tracking. A scene could be when the user is in the main menu of the game. Other examples of scenes is when a user is driving a car or is using a certain weapon or spell. Which parts of the game is interesting and should be used as scenes is entirely up to the user.

The reason why the game is divided into scenes is to be able to use different behaviors depending on what the user is doing in the game. If the user for example is in the main menu in the game he/she might want to be able to click on all the menu buttons with his/her eyes. But when he/she leaves the menu, he/she no longer wants to be able to click on where the menu buttons were and instead he/she wants to use other behaviors. Figure 4 illustrates how the application holds several games which in turn hold several scenes.

Behaviors

The behaviors are, as previously mentioned, a set of actions that occur when the user interact with the game using your eyes in combination with mouse and keyboard. An example of a behavior could be a command to move forward in the game if you look at the top part of the screen while holding down the left mouse button. Another example could be a command to simply click on the screen where you are looking, when you press the spacebar.

In the application the behaviors are divided into two categories; stateful and stateless behaviors. A stateful behavior means that the behavior has several states. A state could simply be: activated or not activated. A good example of a stateful behavior is a behavior that enlarges a part of the screen while you hold down a key and when you release the key, it clicks where you were looking on the enlarged portion of the screen. In this behavior there are at least two states: holding down a button (to enlarge the screen) and releasing the button (to execute the click command).

In contrast, stateless behaviors do not have any states. Stateless behaviors can’t have any special states and they can’t be activated for a period of time. A stateless behavior is simple activated when a condition is met. An example of a stateless behavior is the behavior previously mentioned where the application clicks on the screen where the user is looking when the user presses a certain key. When the user presses the specified key, the application instantly clicks where the user is looking. This brings us to the topic of controllers and triggers.

Figure 4 - The structure of the application.
Controllers
The reason the behaviors are categorized into two groups: stateful and stateless, is because they have different requirements when it comes to the ways you activate them. What the application uses is something we call controllers. Some examples of controllers are “Hold”, “Toggle”, “Hold or Toggle”, “Click”, “Release” etc. A behavior may have several different controllers. This makes it possible to activate a single behavior in several different ways.

There are a number of different controllers and all controllers are specific to either stateful or stateless behaviors. The “Hold” controller is used with stateful behaviors and simply activates a “start” state while you hold down a key, and triggers an “execute” state when the button is released. The “Click” controller is a controller used with stateless behaviors and triggers the behavior immediately when a key is pressed.

Triggers
Up till now I have mentioned that controllers are activated by keys, this is true but a slight oversimplification. Using our terminology a controller is activated by one or more triggers. A trigger is a set of keys and actions, specified by the list of keys and corresponding key states (key is up, down, clicked or double clicked). The triggers also allow the use of key combinations such as “Ctrl+Z” and supports delays which can be used for actions such as; hold a key for at least 2 seconds before activation.

To summarize this so far; a behavior contains one or more controllers that specify what action (hold, toggle, instantaneous, etc.) is required to activate the behavior. The controller then in turn contains one or more triggers which specify which keys and key states should activate the controller.

All controllers for stateful behaviors require at least two triggers: one for activation and one for cancelation. The activation trigger specifies how to activate the first state. Cancelation triggers is then used to, while in the first state, abort the behavior. This is used e.g. when you have enlarged a part of the screen and then decide you don’t want to click where you are looking when you release the activation trigger; you could then press the cancelation trigger to abort the behavior.

To add to the complexity somewhat, the application supports behaviors on two levels; on scene level and on game level. Behaviors on scene level only work when a specific scene is active while behaviors on game level will work during any scene.

List of Behaviors
To give an overview of the different behaviors that will be supported in the application, the following section will give a quick description of some of the behaviors which is implemented or is going to be implemented. This list is compiled by my project leader and contains both new behaviors aimed towards gaming and more generic behaviors which can be used in other contexts, such as every day office work (using eye trackers). Do note that it is very easy to add new behaviors later if the need arises.

Continuous Exposé of screen region – Maximizes a small region of the screen to make it easy to gaze click with high accuracy.

Discreet Exposé of objects – Displays a series of gaze clickable objects which can trigger things in the game.

Gaze enabled hotkeys – Teleports the mouse instantly to where the user is looking and presses a hotkey to e.g. trigger a magic spell in a game where the user is looking. It then moves the mouse back to where it was.

Screen free pan mode – Pans the game view to the direction you are looking. For example if you are looking to the right side of the screen, the game pans to the right.
**Screen instant center on click** – Centers the game view where you are looking when you press a button.

**Gaze accuracy feedback lattice** – A type of feedback which show you the current accuracy of the eye tracking for debugging purposes. Basically this shows the raw data from the sensor.

**Gaze region highlighting** – A type of feedback which highlights regions of the screen when the user is looking at them.

**Gaze direct click on region** – Look at a region to click on it.

**Real-time player gaze pattern** – Displays the point where the user is looking. This is basically a heavily filtered version of the raw data from the sensor, displayed as dots on the screen.

**Aggregation and analysis of gaze** – Aggregate and mutate the gaze data to prepare it for analysis.

**Gaze replay** – Record the game and gaze data, save it and allow the user to replay it later.

**Remote gaze live viewing** – In a multiplayer game, send the users gaze position in the game to the other players.

**Gaze free look** – In a first-person-shooter (FPS) game; rotate the view in the direction the user is looking.

**Gaze DPI throttle** – Make the mouse move fast if the user is not looking at it and slow it down (and increase its precision) when he/she is.

**Head-based in-game leaning** – Allow a character in a FPS game to lean based on the players actual head position.

**Concentration based zooming** – Zoom in the view where the user is looking if he/she hold his/her gaze fixed at that location.

**Zoom at gaze point** – Zoom in/out where the user is looking if he/she is using the mouse scroll wheel.

**Off screen region/menu** – Hide a gaze menu outside the screen. The user can then open them by looking outside of the screen.

**Gaze menu at gaze point** – Open a gaze menu at the point where you are looking when you press a hotkey.

**Notifications and warnings at gaze point** – Display warnings or other notifications at the users gaze point to make sure he/she doesn’t miss it.

**Foveal rendering** – Optimize the rendering of the game by increasing the graphical fidelity where the user is looking and decreasing the quality where the user isn’t looking. This can also be used to create proper depth-of-field/out-of-focus effects.

**Cursor adaptations depending on context** – Change the cursor or other graphical elements depending on where you are looking.

**Features**

One problem that arises when dividing the game into separate scenes is; how do you identify which scene is active? If the application would only target a single game and use source integration, this would be trivial. The application could then receive some kind of identifier from the game that specifies which scene is currently active and then activate the correct set of behaviors.

But this application will be universal and the goal is that it should work with almost any existing game, so we can’t receive any kind of identifier from the game since we can’t modify all existing games. To solve the problem of identifying which scene is active, the application instead uses some basic image recognition.
The application tries to find known graphical elements that are unique to a specific scene and uses this information to identify which scene is active. Selection of these unique graphical elements is however not automatic, it is done entirely manual by the user.

We call these graphical elements, features. These features represent a specific area of a screenshot from the scene in the game that will be matched against the live image. The scene whose features match the live image from the game the best is then selected as the activated scene.

The application uses two kinds of features; positive and negative features. Difference between the scenes features and the live image from the game is summed up and if the sum is less than a certain threshold that scene is considered as a candidate for being the current activated scene.

Positive features are features that add the difference in pixel values between the feature and the image to the sum. A large difference between the features and the image results in a high sum meaning that it is less likely that scene is the active scene.

Figure 5 - Description of the functionality of features.

The negative features work similar to the positive features. They have their own sum which accumulates when the difference between the negative features and the live game image increases. The difference between positive and negative feature is that if the sum for the negative features is higher than a certain threshold, the scene is removed from the list of active scene candidates. This can be seen in the image above where the red curve represents the negative sum over time and the green curve represents the positive features. The striped lines represent the positive and negative thresholds. In the beginning the positive sum is below the threshold and the scene isn’t considered being a candidate for being the active scene. Then the positive sum suddenly passes the threshold and is now a candidate. But shortly after that, the negative sum becomes larger than the negative threshold; this removes the scene from the list of candidates.

Profiles and Resolutions

When the application will be released to the public, the application will include a number of default profiles for the most popular games right now. The profiles can be seen as different configurations of the mappings for a game in the application. Another way to understand their practical purpose is to see each profile as a collection of scenes (see figure 6) and when the user change profile it simply change which collection you want to use in the application.
In the application the different profiles will be completely independent, meaning that if you change settings in one profile, it won’t affect any other profiles. But if we want to make the profile system more powerful it would be possible in the future to add the ability to create child profiles of existing profiles (see figure 7). Making changes to the parent profile would update all the child profiles, but updating a child profile won’t update the parent or their siblings in the hierarchy.

Even though the application include default profiles, adding new, editing and deleting profiles will be easy for the end users.

If profiles can be thought of as collections of scenes which you can change, the system for resolutions can be seen as collections of settings for features and behaviors. There are two major differences in how they work compared to profiles: the selected resolution is based on the current screen resolution in the game and all resolutions contain the same features/behaviors.

The reason we require different resolutions to hold different settings for features and behaviors is because the game changes the shape of its GUI based on the screen resolution. This means that e.g. features might need to be moved to match the GUI for a certain resolution. The way I solved this will be presented in the results chapter.
Theory

To get a good overview of how eye tracking have been done previously and how they do it now I did a series of literature studies. I started by reading up on the eye, how it moves and some novel ideas for gaze interaction. Following this I researched how eye tracking has been done historically and finally; how Tobii does their eye tracking right now.

The movements of the eye

To examine the possibilities and limitations of eye tracking as an input modality it is important to understand the movement and behavior of the eye when you are looking at static and/or moving objects. Sundstedt (2012) mentions that there are five different types of movements that the eye makes. She mentions that these movements are usually divided into two categories: stabilizing movements and saccadic movements.

Stabilizing movements are movements that are used to keep the image steady on the retina. The movements in this category include smooth pursuits, vestibular ocular reflexes, optokinetic reflexes and fixations.

Saccadic movements are used to bring the image of the object you are looking at onto the retina. These include saccades and vergence movements.

- **Saccades** are movements where the gaze moves rapidly from one point to another. Once a saccade begins to move the gaze, it is impossible to change the destination. During the movement basically no visual information will be processed. These movements occur reflexively and voluntary and last between 10-100 ms (Sundstedt, 2012).
- **Smooth pursuits** occur when you try to track a visually moving object. It is possible for the eye to accurately track the moving. Of course this depends on the distance and velocity of the object (Sundstedt, 2012).
- **Vergence** movements are movements used to focus a pair of eyes over an object that is far away. This is used to enhance the depth perception (Sundstedt, 2012).
- **Vestibular ocular reflexes** are movements used to keep the gaze locked onto an object. This reflex occurs even if the head rotates or the eyes are closed (Sundstedt, 2012).
- **Optokinetic reflexes** are movements used to compensate for the motion of the visual field. This reflex produces a sense of self-motion. This can be experienced when you are in a stationary train and the train next to you starts moving (Sundstedt, 2012).
- **Fixations** is not technically a type of motion, one might even say it is the opposite of motion. Fixations are the pauses in between eye movements. A fixation last 200-400ms and is used to hold the image still on the retina. The eyes are however never completely still and jitter somewhat. This jitter is sometimes called drift, tremor or microsaccades (Sundstedt, 2012).

All of these types of motions can be used in eye tracking and gaze interaction in one way or other but I found during my time at Tobii that it is saccades and fixations that are the most interesting when using eye tracking as an input modality.

I learnt at Tobii that the fact that the eye is never completely still and jitters somewhat, is one of the major factors that limit the accuracy of eye tracking. This means that even if the hardware is perfect, you will always get a noisy signal. This can be accounted for by using low pass filtering e.g. by averaging together the ten most recent sample points from the eye tracker to receive a more stable output. One thing to remember is that low pass filtering introduces an unwanted delay and makes the system less responsive. Low pass filters isn’t the only type of filter used for
filtering the gaze data; another common filter is Kalman filters which is used to estimate the gaze point’s movement. Another important thing I learnt at Tobii that one must remember is that a gaze point on the screen isn’t an infinitely small point, but is a small circle of uncertainty. This circle corresponds to the part of the image that is projected on the fovea.

Unusual ways of gaze interaction

There has been a lot of research in how you could use eye tracking as an input modality. An interesting way to use eye tracking is to allow the user to perform gestures with their gaze. Rozado et al. (2012) presents two ways to use the gaze movement as gaze gestures. They propose this as a way to write text without the need of having an onscreen keyboard and even possibly speed up the text input. They mention that they considered two different types of gaze movements to be used for gaze gestures: saccadic movement between predefined areas of the screen and smooth pursuits along a predefined line. It is these lines and areas of the screen that is making up the shape of the gestures. They mention that Robinson (1965) had concluded that gliding the gaze along a line is physiologically impossible without performing a smooth pursuit of a moving target. It is however possible to make many, short saccadic movements along a line. Robinson (1965) also interestingly concludes that saccades can happen before, after or during a smooth pursuit. To help the user and give some kind of feedback to the user that he/she is following the line correctly, Rozado et al. (2012) move the mouse cursor to the gaze point. By following these predefined lines with the gaze or by jumping between predefined areas, it is possible for an application to detect what kind of gesture the user is making.

Another interesting way of using the eyes as input modality is to use the size of the pupil. Ekman (2008) presents a possible way to use the change in pupil size to control a computer game. They present a simple game where the user can interact with flowers using their eyes. If the user enlarges his/her pupils the flowers open up and reversely if the pupils get smaller the flowers close. It might sound hard to voluntarily change your pupil size but Ekman (2008) mentions the pupil size changes depending on a number of factors such as lighting conditions, cognitive effort, mental imagery and various forms of stimuli such as images and sound. They also mention that some physical processes can also affect the pupil size, e.g. changing gaze focus, holding your breath or muscular effort.

Eye tracking methods

Eye tracking has been around a lot longer than one might expect. Already in the 1800s there was research being done in the field (Laurence, 1976). From the beginning people were simply watching the eyes of other people, studying how their eyes move when e.g. reading a book. One of the earliest forms of a physical eye tracker was in the form of a lens with a rod pointing in the direction of the gaze (Tobii, 2013). New technology has allowed for more robust, accurate and non-intrusive methods for eye tracking. Some of the methods used today are listed and briefly explained below.

Electro-Oculography

This is an old technique that was first discovered back in 1920s and 1930s. It works by measuring the potential voltage differential between electrodes over/under and to the left/right of the eyes (Laurence, 1976). These voltage differentials were noticed to directly correlate to the sine of the angle of the eye. The voltage differentials caused by the movement of the eyes are however very small (15 to 200 μV), this means that the measurements from the electrodes are noisy from other factors such as voltage differentials caused by other electrical interference from sources.
Corneal Reflection
This is an optical method that exploits the fact that the cornea has a smaller radius of curvature than the overall eye. Because of this, the reflection from a bright light moves about half as much on the cornea than on the rest of the eye, this in turn causes the reflection on the cornea to be displaced opposite to the overall eye movement. This can be used to calculate the direction of the gaze (Laurence, 1976).

Limbus, Pupil and Eyelid Tracking
The limbus (the border between the iris and the sclera) is easy to detect and can be used to calculate the direction of the gaze. The problem with using the limbus is that large a portion of the limbus is usually covered by the eyelids and the border of the eyes (Laurence, 1976). An alternative method is to instead track the pupils position. Since the pupil is smaller it allows for more eye movement and greater field of view without being covered by the eyelids and the borders of the eyes. If we also detect the shape of the eyelids it is possible to combine the two methods to get improved performance.

Contact Lens Methods
One of the most accurate methods used for eye tracking is contact lens methods (Laurence, 1976). These methods use a physical contact lens placed onto the eye that is tracked in one way or another. Normal lenses are placed mainly on top the cornea but this doesn’t work well for eye tracking since the cornea itself can move around slightly relative to the sclera when forces are applied. Instead one has to use special lenses that attach more firmly. Some examples of solutions for how to attach lenses more firmly involve: filling the cavity between the eye and lens with a sodium bicarbonate solution that one lets evaporate creating a vacuum, having a small valve on the lens to extract any fluid or air. These methods can cause discomfort and even pain for the users, some even requiring anesthetics.

Some early contact lens methods dating back to the late 1800s even involved attaching rings of plaster of paris to the eye onto which mechanical linkages were attached. More modern versions include: attaching small reflective surfaces to the lens which can be easily tracked and attaching two small electrical coils that can be tracked in an electromagnetic field.

Head Tracking
Most of the different methods mentioned above are primarily used to measure the direction of the eyes relative to the direction of the head. To get the actual point in the real world where a person is looking it is also required to know the position and rotation of the head. This has been done historically by holding the head in a fixed position, having mechanical linkages attached to helmets, optical sensors watching light emitting diodes or numerous other methods.

Today the primary method, for tracking the heads position and rotation, is a camera based method that tracks the features of the face and the eyes to calculate the 3D position and rotation of the head with respect to the camera. If also the cameras position and rotation relative to e.g. the computer monitor is known, it is possible to calculate the gaze point on the screen.

How Tobii's eye trackers work
Tobii’s eye trackers utilize a variant of pupil tracking in combination with corneal reflection to create an effective method for eye tracking called Pupil Centre Corneal Reflection (PCCR). PCCR is quite simple in theory: it uses one or more cameras and a number of illuminators that have two purposes: change the brightness of the pupil depending on eye color and create glints (highlights) on the cornea. By measuring the center of the pupil and the positions of the glints on the cornea, relative to the pupil, we can calculate the gaze direction. There is quite a bit of math involved, but Guestrin (2006) explains exactly how PCCR works.
Since the eyes are roughly spherical you can see that the glints in the eyes will stay stationary if you only rotate your eyes. To illustrate this, imagine standing in front of a beach ball while shining a flashlight onto the right side of the ball, creating a glint. If we start rotating the ball the glint will remain in the same spot (relative to you) even though we rotate the ball. This means that we can use the glint as a reference point for the position/rotation of the pupil relative to the glint.

Figure 8 – Tobii’s explanation of how their eye tracking works. Image rights: Tobii Technology

To know where on the screen the user is looking two kinds of measurements are needed: the position and rotation of the head relative to the monitor and the position and rotation of the eyes. Calculating the direction of gaze, relative to the head, is done by comparing the location of the pupil with the glints. To calculate the rotation and position of the head, relative the eye tracker, the eye tracker uses image recognition to extract features of the face and the position of the eyes. Figure 8 shows Tobii’s own explanation of how their system for eye tracking works.

This is of course an oversimplification of the actual implementation of the system. In reality everyone’s eyes are a bit different and not completely spherical, requiring more advanced math to create accurate eye tracking. One of the ways Tobii’s software improves the accuracy is to require every new user to run a calibration before using the eye tracker. This calibration creates a model of that specific user’s actual eyes in three dimensions which is then used in all the calculations (Tobii, 2013).

Limitations and problems with optical eye tracking methods and PCCR

There are a number of limitations with PCCR which comes from a combination of technical limitations and human physiology. A technical problem and overall usability problem with PCCR is the fact that the user has to sit at a fixed distance (within some margins) for the eye tracker to work well. The reason for this is that the camera has to receive a good image of the eye to be able to extract the position of the pupil and glints. The tradeoff the hardware designers have to make is between having a wide angle lens and a telescopic lens on the camera. A wide angle lens gives a wide field of view which in turn allows for a lot of head movement but it makes the eye smaller, making the accuracy lower. In contrast a telescopic lens forces the user to keep his/her head in a fixed position but it provides very high accuracy since the camera receives a large image of the eyes.
Another limitation of PCCR is that it currently doesn’t work well outdoors. The reason it doesn’t work is because there are too much interfering light coming from the sun. The interfering light makes it hard to detect the glints in the eyes, making it hard to accurately calculate the gaze position. The way most eye trackers try to eliminate this problem is by using near infrared light instead of regular visible light, this eliminates all the interference from regular visible light. By using near infrared light you minimize the problem of interfering light while indoors since there aren’t many infrared light sources indoors. The sun is however a powerful source of infrared light, which causes a lot of interfering light. Using near infrared light also has the benefit that the user doesn’t have to see the bright light shining directly in their eyes from the eye tracker.

Another thing that causes problems with PCCR is that people’s eyes are all different. Some people have light eyes others have dark and some people keep their eyes fully open at all times while others are squinting. Glasses and contact lenses can also interfere with the eye tracking. The way optical eye trackers usually handle different colored eyes is by changing the illumination of the pupil. The goal is to create as much contrast between the pupil and the iris as possible, making it easy to extract the pupil from the camera image. In practice this is done by having two sets of illuminators: one close to the camera and one further away. By turning on the illuminators close to the camera it creates a red-eye effect which causes a bright pupil, suitable for people with dark irises. In contrast, by turning on the illuminators further away from the camera you get dark pupils, suitable for people with light irises.

Squinting eyes are a problem that is harder to account for. If the top and/or bottom of the pupil are covered by the eyelids, it is still possible to perform eye tracking. It does however mean the accuracy is lowered since all the eye tracker can do is make qualified guesses and estimate the position of the pupil. This estimation is based on the information available to the eye tracker such as the shape of the visible part of the pupil, the shape of the iris (the pupil is usually roughly in the center) and previous positions. Worth remembering is that not only the pupil which is needed for eye tracking as previously mentioned, the glints from the illuminators have to be visible as well, meaning the eyelids cannot obscure to much of the eye, blocking the glints.

Glasses and contact lenses provide their own problems for optical eye tracking techniques. A common problem with glasses with thick glass or special coatings is that they can cause reflections from the illuminators. The bright reflections in the glasses can block parts of the eye making eye tracking impossible. Glasses and contact lenses with large correction can also distort the image of the eye, making it necessary to take this into consideration. Contact lenses have their own problem which is they can move slightly on top of the cornea; distorting the shape of the eye and moving the glints slightly over time. Most of these errors can be corrected for by doing calibrations. But since glasses and contact lenses can move, users with them might have to recalibrate more often than other users.

Moral and ethical aspects of eye tracking

Privacy is an important topic today and large parts of the general public today is concerned about the fact that more and more of their actions is being monitored by different authorities. With the advent of eye tracking, users now not only have to worry about if what they write and say is being recorded; now there is a technology that makes it possible to record where you are looking as well. But eye tracking can not only detect where, but also how you look. Stanley (2013) raises a number of concerns regarding eye tracking on the American Civil Liberties Union’s blog. He mentions that eye tracking could in the future potentially be used to detect a number of traits such as:

- Cognitive disorders
- Drug and alcohol use
• Mental and psychological illness
• HIV/AIDS
• Lie detection
• Intelligence
• Sexuality

All of these traits might not be feasible to detect in practice in the foreseeable future but just the fact that it might be possible worries a lot of people.

Recording where users are looking and depending on how they are looking, gather information about certain traits might seem scary and privacy invading. However Pärnamets (2013) takes this to a scary new level by claiming to be able to manipulate moral decisions made by the users by interrupting them at the right times. Based on the position of the gaze point it was possible to influence the users’ choice when posed a moral question such as “Murder is sometimes justifiable” and given the two alternatives: “Sometimes justifiable” and “Never justifiable”.

In the future it might be possible to analyze how and what a user reads and writes, what a user looks at, what websites he/she is visiting and maybe even what he/she is thinking. This data could then be used to create a very detailed profile about every user using a computer online. I imagine this will continue to be something a lot of people are worried about in the future, especially as the technologies continue to improve.
Methodology

In this chapter I present the methodology I used throughout this project. I started by defining a number of personas from our target groups. I then created a series of initial sketches which was the basis for what followed next; the prototype. The prototype was then evaluated through two user tests. All this together with an interview with our professional game mapper resulted in a set of final sketches and the beginning of a program that will be sold to the public.

Personas

When I started this project I was given two target groups for the designer. These two target groups have very different needs and requirements and the goal is to design a graphical user interface that can satisfy both users.

The first target group for the designer is gamers who want to create a mapping for their favorite game(s) and have no previous experience of the designer and will only receive the help provided with the application or online.

The second target group for the designer is employees at Tobii that will use the designer to map hundreds of games. They will get a crash course in how to use the designer and what to think of when using it.

I also identified a third target group which is gamers who don’t want to map games, but just want to play their favorite games using the eye tracker. They need to have a way to see what they can do with their new eye tracker in their favorite games.

Since my primary goal of this project was to design the GUI for the designer, I primarily focused my efforts on satisfying the first two target group’s needs. However since the third group was also very important I tried to keep its needs into consideration as much as possible when designing the GUI. This was especially important since the first two target groups, more or less, shared the third group’s wish for a good way to show a summary of what can be done using the eye tracker in the games.

To make the target groups more tangible and to personify them, I have created three personas. These personas were then used as the “real” users for whom I designed the application. By having some personas in mind, when designing the application, it was easier to find the different needs and requirements.

Gamer – Wants to map his favorite game

- Erik Andersson
- Man
- 22 years old
- Lives alone in an small apartment
- Studies computer science at KTH
- Owns a computer, Xbox 360 and a series of different handheld consoles
- Plays computer games every day

It is a Friday afternoon and Erik has just come home from long day at school. Earlier in the week Erik bought a new eye tracker which included an application that would allow him to use his new eye tracker to play his favorite game in a new and exciting way. To his big disappointment he finds that his favorite game isn’t supported by the application by default. But
after looking at the application for a short while he notices that it is possible to add new games to the application and he decides to try to add his favorite game to the applications list of games.

Erik has no idea what to do or how to use the application but the application explains exactly what he has to do and he finds it easy to follow the instructions. He starts by identifying and adding a couple of scenes where he wants to use eye tracking. Then he finds the unique features for each scene, making sure that the application can identify which scene is active. Lastly he goes through the list of behaviors and add the behaviors he thinks fits the different scenes. After he is done, he saves and starts his favorite game. He receives a notification that the application has detected the game and that the game is now ready to be played using his new eye tracker.

**Employee**

- Ola Matsson
- Man
- 30 years old
- Lives in apartment just outside of Stockholm with his girlfriend
- Works at Tobii
- Owns a computer and a PlayStation 2
- Plays computer games a couple of times each week after work

It is a Wednesday and Ola has just come to work and is just about to finish his morning sandwich while waiting for his computer to boot up. He knows that today he has to create new mappings for at least twenty games and he needs to do it fast since he have meetings in the late afternoon. As soon as the computer boots up he loads up the first game and starts the application. He is not greeted with any tutorials or hints since he has chosen not to have those any more. He knows exactly what needs to be done. To start he defines the different scenes in the game in no time using keyboard shortcuts and other tricks which the application provides to speed up the process.

When he starts to add behaviors to the scenes he realizes he needs to check which buttons he used in the previous games since he wants to create a similar experience across all games. He quickly opens a new window for that game and looks up the information he needed. Before he is finished with the behaviors, he is suddenly gripped with panic. He remembers that this game requires a different mapping for every resolution. To his relief he remembers that it is possible to copy the behaviors from one resolution to another, this saves him a lot of work and he manages to finish everything well before the meetings.

**Gamer – Just wants to play using her new eye tracker**

- Anna Lundström
- Woman
- 21 years old
- Lives alone in a small apartment
- Works as a IT-consultant
- Owns a computer, PlayStation 3 and Wii
- Plays computer games every day after work

Anna has just come home from a long day at work and is tired and feels like playing a computer game. In her spare time, Anna plays a lot of StarCraft 2 and has recently purchased an eye tracker and decides that today is a good day to give it a try. StarCraft 2 is supported by the
application that comes bundled with the eye tracker and she decides to give it a try. She has no interest right now to change any settings or create a personal profile; instead she just wants to play the game with her new eye tracker. As soon as the game has loaded a notification tells her that the application has detected that the game is running and it is now possible to use the eye tracker.

Confused about what and how she should use the eye tracker, she is happy to notice a message telling her that she can press “F12” any time to get a summary of what she can do in that specific scene. She also notices a small message every time she changes scene, telling her which scene is now active. She quickly learns how to use the eye tracker with the application and begins to play her favorite game in a whole new way; one which she has never experienced before.

**Initial sketches**

Before I started my work, my project leader had produced a sketch (see figure 9) of how he imagined the graphical user interface could look like. It contained all the information about the system on a single view with different areas dedicated to different parts of the system.

From this initial sketch I produced a set of my own sketches (see figure 10). Some of my sketches were similar to my project leaders sketch and contained all the information on a single screen and some designs divided the GUI into several separate views.

![Figure 9 – My Project leaders’ mockup of the graphical user interface](image)

After discussions with my project leader and the usability experts at Tobii, we concluded that the best direction to follow would be to divide the GUI into several smaller views. Doing it this way allows us to not overwhelm the user with a large amount of information and choices at once.
The risk of dividing the GUI into several smaller views with less information on each view is that the GUI will be slow to use if it requires a lot of navigation back and forth. A good example of this is the comparison between how you change settings on a compact cameras compared to how you do it on a professional Digital Single-Lens Reflex (DSLR) camera. DSLRs have lots of buttons all over the camera while cheap compact cameras usually have a single small joystick to navigate through a series of menus.

The DSLR is harder for a beginner to use but is a lot faster and more efficient for a professional who needs to change settings frequently. A compact camera is very user friendly in the sense that a beginner can pick it up and take a decent picture and maybe even change settings, but changing settings takes a lot longer making it less ideal for professionals.

The challenge in designing the GUI for this application is finding a good tradeoff between satisfying beginners who want a GUI that isn’t too complicated and satisfying professionals who would prefer quick access to all the information from a single view, if possible.

The first thing I thought could be a good start was to divide the application into three separate views: scene selection, feature editor and behavior editor. The three concepts (scenes, feature and behaviors) have as mentioned earlier a relation as such; a scene contains a set of features and a set of behaviors. The behaviors and features are only connected in the sense that they both belong to the same scene. With this in mind it makes little sense to display behaviors and features on the same view since they don’t contain any common information that is relevant for the function of the other.

Together with the usability experts I decided to continue working with one of the sketches (see figure 11) that seemed to have the best balance between being easy for beginners and efficient for professionals.
I created a series of iterations of the selected sketch (see figure 12), evaluating pros and cons after each iteration. The evaluation I did myself based on the personas but I also got feedback from my project leader and the usability experts. Some of the iterations involved only changing the size of different components while others introduced completely new ideas and concepts.

With the initial sketches ready I had everything I needed to start creating the prototype. The prototype will be based on the initial sketches and the final sketches will be the base for the final program that will be released to the consumers.

Prototype

I began creating the foundation for the prototype very early in the process. We decided that we would start by creating a high-fidelity prototype. The advantage with doing a high-fidelity prototype is that it will be fully interacting and have the look and feel of the final product (Sharp et al, 2007). The disadvantage is that it is a lot more time consuming than creating a low-fidelity prototype first. My reason for creating a high-fidelity prototype was because it was decided that
this project would result in a product that Tobii would sell to their customers. Because of this there were pressure on me and my project leader to deliver some visible progress continuously throughout the project. This meant that I had to work on the prototype in parallel with doing interviews and user tests. In the beginning I would say that I worked in what Göransson (2002) would consider horizontal prototyping. This means that I worked on only the graphical user interface (GUI) and created a shallow prototype. In the later parts of the program I changed the approach and started working in more of a vertical prototyping way where I worked on the GUI, the logic and the underlying data structure at the same time.

![My project leaders basic GUI](image)

**Figure 13 - My project leaders basic GUI**

Before I started, my project leader had created a very basic GUI that simply displayed a set of scenes in a list and allowed the user to position some features on a screenshot of the scene (see figure 13). I used this GUI as a starting point when I started to create my own prototype from my sketches.

**MVVM**

The prototype uses an architectural pattern called Model View ViewModel (MVVM). This is a pattern that is commonly used in large and extensible enterprise graphical applications. The Model in MVVM is the layer of the application that represents the actual data and content to be displayed in the view. In this projects application some examples of data that is contained in the Model are: scenes, behaviors and features. The model contains all the information and objects that will be saved to disk when the user wants to save his/her progress.

The View in MVVM is what is displayed to the user, it is the actual GUI. In this project we used WPF (Windows Presentation Foundation) to create the GUI for the application. WPF is a powerful graphical subsystem for rendering user interfaces. It allows the developer to create buttons, menus, etc. using simple XML (Extensible Markup Language) and vector graphics, similar to how HTML is used to create webpages. The exact name of the language used in WPF is called Extensible Application Markup Language or XAML for short. XAML is a XML based declarative language created by Microsoft, primarily used in their .NET platform.

The ViewModel is a class used to connect the View with the Model. It has many roles but the primary one is to define what happens when the user interacts with the GUI and how the Model should be updated. Very simplified one can say that the model is the data, the view is the GUI and the ViewModel is the instructions of how to change the data based on the user’s interaction with the GUI.
My role in the development of the prototype was to create all the Views and the corresponding ViewModels. I also created new Models and modified the existing ones to match the functionality I wanted.

Since it is the Model only that is saved to disk (e.g. when we want to save a new behavior or a new feature) I can continue to change the View and ViewModel as much as I want without any risk of making the saved scenes, behaviors, features, etc. incompatible with future versions of the software. As long as we don’t change the model, all saved scenes, features, behaviors, etc. will be compatible with future versions of the application. In figure 14 it is possible to see how the GUI and Model is only connected through the ViewModel.

Since we wanted to start creating mappings/profiles for different games as soon as possible it was important to create a good Model as fast as possible. If we had to change the Model, then all saved mappings/profiles for games would not work anymore since we had changed the underlying data structure.

**Caliburn Micro**

To speed up development and to make following the MVVM pattern easier, we decided to use a framework called Caliburn Micro (CM). CM makes working with WPF/XAML in combination with the MVVM pattern a lot easier by providing a lot of useful functionality.

The main functionality I used was CMs automatic way of connecting the View with the ViewModel. CM did so by using a naming convention where the View is called something like “AnyClassView.xaml” and the ViewModel something like “AnyClassViewModel.cs”. By naming the View and the ViewModel using this convention a connection between the two files was created through CM.

As long as there is a connection between the View and the ViewModel (using CM) it is very easy to e.g. define what happens when a user clicks a button. To illustrate this, here follows a very short example where pressing a button makes the application output “Hello!” to the console:

In the View we first define a button using XAML:

```xml
<Button Content="Click Me" cal:Message.Attach="[Event Click] = [Action SayHello("Hello!")]" />
```

Then in the ViewModel we can define a function called SayHello that takes a string as input:

```csharp
public void SayHello(string msg)
{
    Debug.WriteLine(msg);
}
```

Clicking the button in the View would now call the function SayHello in the ViewModel with the argument “Hello!” making the program output “Hello!” in the debug output.

As you can see in this example it is very easy to connect various actions in the View with functions in the ViewModel. Caliburn helps with a lot of things but the final thing I will mention is that CM helps with connecting values of GUI controls in the View with properties in the ViewModel. This could look something like this:
In the view we have a textbox whose content we “bind” to a string property in the ViewModel:

```xml
<TextBox Text="{Binding TextBoxText}" />
```

In the ViewModel we can then define a string property called TextBoxText as such:

```csharp
private string _textBoxText;

public string TextBoxText
{
    get
    {
        return _textBoxText;
    }
    set
    {
        _textBoxText = value;
        NotifyOfPropertyChange();
    }
}
```

This creates what is called a two-way binding. This means that writing in the textbox in the GUI updates the property in the ViewModel. It also means that changing the property in the ViewModel updates the text in the GUI. The “NotifyOfPropertyChange()” is used to tell any object that uses this property that this has been changed and they should act accordingly. In this case this means that the View should refresh the TextBox with the new value.

By using CM it was possible to greatly increase the speed of development and to create code that is easier to read for other developers.

**User tests**

To evaluate the prototype during different stages of development, I performed a series of user tests. These tests were designed to allow the users to perform a series of tasks in the GUI. I wanted to test everything that was possible to do in the GUI, however during each test I only tested the areas whose design, at the time, was close to the intended design from the sketches.

During the development of the application it was prioritized to get the application feature complete before making it user friendly. The reason for this was that we had a tight schedule and needed to have a working application that an employee could start to use to create profiles/mappings for new games as soon as possible. This meant that we had a relatively well working prototype of the application long before we had a good user experience and it was my job to make sure we also got a good user experience.

The tests were performed on a laptop connected to an extra monitor, mouse and keyboard. I let the users sit and use the extra monitor, mouse and keyboard while I monitored the test from the laptop. I recorded the tests using a free program called Hypercam 2. This software allowed me to record the screen and the audio from the built in microphone in the laptop. The software also highlighted where the user clicked by adding a colored star (blue for the left mouse button and red for the right) over the video where the user clicked. This made it very easy for me to go through the recordings afterwards and analyze the footage.

Before each participant started the test I gave each of them a quick summary of all the terms and concepts they needed to understand before using the application. I explained how a game has several scenes, each scene contained two lists: a list of behaviors and a list of features. I then explained what features and behaviors are and what they do. Each participant got the same quick summary and they also got a chance to ask questions during any time of the test.

The tests themselves were performed by me giving the users a task verbally and they then performed the task in any way that they felt was natural to them. The users were asked to *think*
Methodology

*aloud* during the test to both provide me with a deeper insight to their reasoning and to make the users think about their actions.

All users performed the same tasks and in the same order, this made sure that they all had the same conditions before, during and after each task. Another way to perform the tests would have been to have the different users perform the tasks in different order. This would remove the effects of learning. However I felt that I had too few participants to benefit from having a random order (since it would be hard to draw any significant statistical conclusions regarding learning etc.) and instead opted to have the users perform the tasks in an order that would simulate the actual workflow more accurately. In other words, the test used within group design. This allowed me to compare the results from the different users to find some trends and patterns, even though I had relatively few (12 in total) test subjects.

The first test

The group chosen for this first test was a series of friends and relatives of employees at Tobii. Most of the users were high school students and none of the users would consider them self a gamer. The selection of users was not optimal since it didn’t fit the target groups well, but at the same time it also meant that the GUI had to be even more user friendly. I also didn’t have the time and resources to be too picky about which users I used for this first test since I had such a tight schedule. In the end I got seven participants who performed the test. This was too few participants to do a good statistical and quantitative evaluation of the results so instead I focused on the qualitative results from the tests.

The first test was designed to test some of the most basic functionality and the overall navigation within the software. This test was performed on users with no prior experience of eye tracking and very limited gaming knowledge.

The tasks I asked the participants to do were:

1. Change the name of a scene.
2. Remove a scene.
3. Add a behavior to a scene.
4. Change the settings of a behavior.
5. Add support for a new resolution
6. Change the settings of a scene.
7. Add a feature.
8. Change the size of the feature.
9. Move around the feature.
10. Change the type of the feature.
11. Remove the feature.
12. Add a new profile.
13. Create a copy of the default profile.

These tasks were primarily chosen for two reasons: they are the parts of the application that are the most similar to the sketches and they involve a lot of navigation back and forth between different levels within the application. Especially the navigation was important to see if the users understood correctly since it is such a vital part of the whole experience. If it is difficult and illogical to navigate the application, the whole user experience will be negative.

The drawback of having the tasks in the same order for all the users was that the tasks in the beginning of the test were more likely to get a worse result than the later tasks. This was because the users were new to the application in the beginning while in the later tasks they had learnt how to navigate the application and have gotten to know some of the design choices I had made. This meant that I couldn’t compare different tasks with one and other and say things like “Task 1 was easier than task 2”. What I could do however is to see if the majority of users managed to solve a specific task and in what way they chose to do it.
The second test

The second test was performed a couple of weeks later with the same version of the prototype. This time the users had been selected based on their gaming and overall computer knowledge. The users were recruited from a group of computer engineering students that came to visit Tobii. We (the UX experts and I) asked the students if they wanted to come back in a week or two and participate in some tests. The ones that were interested signed their name and email on a list to be contacted later. In total we got around 20-30 names/email addresses.

We then asked them to fill out a questionnaire with eye tracking and gaze interaction questions. The questions were used to filter out the people that were interesting for both me and the UX experts (who were conducting their own tests). The questions that was relevant for my study was:

- Any previous experience with Gaze Interaction? (Yes, No)
- How often do you play videogames? (Never, Rarely, A couple of times a week, Every day, As much as I can!)
- Do you own any gaming peripherals? (Text box)
- I usually keep up to date with new technology and gadget. (1 – 7, Disagrees - Agrees)

From the questionnaire we received 15 responses out of these we selected the five that met our requirements the best. I wanted the participants to be gamers with good knowledge of computers and be interested in new technologies. The five participants we choose matched my requirements well.

Later when it was time for the test, I had the participants fill out a second form. This time they got to answer these questions:

- How old are you? (Textbox)
- Occupation? (Textbox)
- Gender? (Textbox)
- What is your experience with eye tracking? (Multiple choice)
- How often do you play videogames? (Multiple choice)
- I consider myself an above average computer user. (Scale, Disagree – Agree)
- I usually keep up to date with new technology and gadget (Scale, Disagree - Agree)
- Are you familiar with the game StarCraft 2? (Scale, No – Yes)
- Do you own any gaming peripherals? (Textbox)

When they answered the questions they were told that they could ask me anything if something was unclear or if they wondered anything.

The actual test was performed just like the first test, with the same tasks, in the same order. The main reason for this was because I wanted to see if there would be any noticeable difference between the participants in the first and second test since their gaming and overall computer knowledge was very different. The second group was a lot more experienced with computers and games and matched our target groups well and the first group was a lot less experienced with these topics.

After the tests I asked each participant five questions. We discussed the questions and their answers for a couple of minutes. The questions were:

- What was the application you just tested used for?
- What worked well and was easy to understand?
Methodology

- What was difficult or hard to understand?
- Are scenes, features and behaviors good names?
- Do you have any tips for how to improve the application?

The results from the tests and questions were compiled in a spreadsheet, where I then summarized and analyzed the answers. I then compared the results with the first test to see if there was any noticeable difference.

Interview with professional mapper

Since we knew that the application would be sold as a product by either Tobii or some other company, we also knew that we had to create some default mappings for the most popular games played today. These mappings were going to be bundled together with our application when delivered to the customer when they buy our product.

To create these mappings we employed a man called Rickard Wallin, to work with our application during six weeks. His job would be to map a large amount of games until launch. Rickard would after six weeks be very well acquainted with the application and would definitely be considered the expert in the field of mapping games for eye tracking in our application.

To get his comments, thoughts and requests regarding the application I held an interview with him. This interview was unstructured and involved us discussing various aspects of the application. I tried to keep the questions and discussions focused on increasing the efficiency and experience for the professional mapper who will map a lot of games. I choose to do a unstructured interview since I wanted rich data and didn’t have any need to be able to replicate the interview (Sharp et al, 2007).

Worth noting was that I worked closely with him during these six weeks so we also discussed the application many times during the development.

Final sketches

Due to the fact that the deadlines at KTH and at Tobii isn’t the same I would not be able to release the final version of the application as a result of this thesis instead the result is a prototype or early version of the application together with a set of sketches. The sketches show my goals and ambitions of the final design and how the application will look when it is completed.

The sketches were made in Adobe Photoshop and were based on the results from the tests, comments from coworkers and my own observations. The sketches are slightly stylized and I have intentionally not used very many colors, since the emphasis should lie on the interaction and not on the color scheme. This can otherwise be a problem and lead to a lot of superficial comments regarding the look of the application (Sharp et al. 2007). The choice of colors is of course important. Since it can affect the usability; I have included some colors (e.g. red and green for positive and negative features) where I think it is necessary.
Results

This chapter presents my findings from the user tests, application design meetings and the interview. The resulting prototype and the final sketches is presented and described in the end of this chapter.

Initial sketches

After a series of iterations, the initial sketches resulted in sketches that I decided to use as the starting point and initial ambition for the final application. These sketches would later be improved upon based on the results from the user tests and the discussions, resulting in the final sketches.

Below is a short presentation of the different views of the initial sketches.

Game selection

This view (see figure 15) is what is displayed if you open the designer by clicking on the applications icon. Here you can see a list of all the games that is already mapped by the application.

The list of games could be updated automatically periodically or manually. To update it manually the user presses the button at the top right corner of the application, this downloads the latest “official” game mappings created by Tobii through a cloud service that we will create. We are also thinking about allowing the user to share their game profiles with each other through the cloud service in the future.

![Select Game](image)

Figure 15 - The Game Selection as it looks in the initial sketches.

The different games are presented in a vertical list. Since we plan to support a large amount of games there is also a search bar below the list that quickly allows the user to find the correct game. Clicking on a game in the list opens up the next view which is the scene selection.

To manually add a new game to the list the user have two options, either press the “Add Game” button in the lower left corner. This causes a popup to appear where the user can select the executable file the game uses and give the game a name that will be displayed in the application.

There is also a button called “Options” which opens a popup where the user can change application level settings. The reason I mention that it is application level settings is because we have options on every level such as: application settings, game settings, scene settings and behavior settings.
Scene selection

The scene selection view (see figure 16) consists of a list of scenes that fills column by column and scrolls horizontally. The reason I choose this type of list was that I wanted a list that used the space efficiently and would reduce the need for scrolling regardless if the window is maximized or not.

![Scene Selection View](image)

Figure 16 - The Scene Selection as it looks in the initial sketches.

Each scene is represented in the list with an image of the scene and the scenes name. Pressing either the text or the image opens a popup with more details about the scene.

At the top right there are two combo boxes which allow the user to select which profile and resolution combination he/she wants to edit the settings for. To add a new profile the user can click the button “New Profile” in the lower right corner.

The user has two ways of adding a scene. The first is by clicking the “Add Scene” button in the lower left corner. Clicking this button opens a popup that tells the user that the application is ready to add a scene and that the user should return to the game and press a certain hotkey when the scene in question is active in the game. Pressing the hotkey returns the user to our application were a new scene now has been created. The second way to add a scene is by pressing a different hotkey when playing the game. This creates a new scene directly without the need to open the popup first. This method is faster but less intuitive.

Scene details

When clicking on a scene in the scene selection view, a popup opens where you see some information about the scene as illustrated in figure 17. At the top you see a clickable image of the scene. Clicking the image maximizes it.

![Scene Details View](image)

Figure 17 - The Scene Details as it looks in the initial sketches.
Below the scene image there is a list with information and buttons. There are three things the user can do here:

- The first thing the user can do is rename the scene.
- The second thing is to open the behavior editor
- The third thing is to open the feature editor

In front of the buttons for the feature and behavior editors there is a text showing the number of features and behaviors for the scene.

**Feature editor**

Clicking the “Edit” button next to the “Defining Features” text opens up the feature editor. When opening the feature editor, the first thing you notice is a large image of the scene (see figure 18). If there are any features in the scene there will be colored rectangles on top of the image. These rectangles represent the features and the color represents the type of feature.

![Figure 18 - The Feature Editor as it looks in the initial sketches.](image)

There are a number of things you can do:

- To select a feature the user can either click on a rectangle or select a feature in the combo box in the lower left corner. A selected feature becomes slightly transparent and the border starts to blink slowly to catch the user’s attention.
- To move a feature the user simply clicks on the rectangle to drag it around on the image with the mouse.
- To change the size of a feature the user can click and drag on a small triangle in the lower right corner of a rectangle.
- To change the type of a feature, there is a small square in the upper right corner of the feature rectangles that brings up a dropdown menu that allows the user to change type. Alternatively the user can use the “Type” combo box below the image to change the type.

All the feature specific settings are located in a dark grey area below the image. This is a design that is used in the behavior editor as well to make the workflow similar.
Behavior editor

At first glance the behavior editor is very similar to the feature editor (see figure 19) and this is intentional. The reason is that I want the user to quickly understand that the dark grey area under the image is specific to the selected behavior.

Above the image there is a combo box where you can select the behavior you want to edit. Selecting a behavior changes the content of the dark grey area under the image based on the type of behavior selected.

![Figure 19 - The Behavior Editor as it looks in the initial sketches.](image)

All the different types of behaviors have their own settings panel with completely different controls. If the behavior has any settings that specify an area or location of the screen, this will be displayed on top of the image as editable rectangles or points, similar to the rectangles in the feature editor.

To make the graphical interface easy to understand I have made all the rectangles, points, etc. in the different behaviors look the same in both the settings panel and in the image. For example a rectangle (area of the image) is represented in the settings panel as four text fields (x-position, y-position, width and height) and in the image as a scalable and movable rectangle. All rectangles in all behaviors are presented in this way. Different types of data such as vectors, numbers, points and colors also have their own “standardized” representations in the settings panel and in the image.

Show Description

Users who only are interested in playing the games shouldn’t need to open the full designer and have to navigate through the interface to try to decipher what is possible to do when they are playing the game. The way I’ve solved this is by creating a simple system for writing and viewing descriptions about each game, scene and behavior.

To read the description about a game and its scenes and behaviors, the user can open a special view designed for this purpose (see figure 20).
This view is quite simple in its design. It consists of a horizontally scrolling list of descriptions of the scenes. At the top of each description there is an image of the scene with some overlays. Below the image there is the name of the scene followed by some text description.

In the text there are number icons that correspond to the overlays in the image that makes it easier to describe where to look and what part of the screen is talked about in the text.

There are also hotkeys inlayed in the text. Clicking these hotkey-texts allows the user to quickly change the hotkey to something else. Alternatively the user can click the “Edit Key Binding” button in the top right corner to edit all the bindings.

**Edit Description**

To create and edit descriptions the user can open a description editor (see figure 21) from the scene selection view. The reason the descriptions have to be manually written is because it is very hard to automatically generate easy to understand descriptions. If the descriptions are manually written it can say things like: “Look at the map to teleport to the place you are looking at”. However if the description is automatically generated there is no way for the software to know that the area of interest is a map in the game and that if you click it the character is teleported. Instead it would say something like “Look at the area of the screen and you will automatically click where you are looking” which also works, but is less easy to understand for the user.

In the top left corner of the editor there is a scene image which the user can right click to add numbered icons/markers to. Below the image there is a text field where users can write the description about the scene and what behaviors exist and how they are activated. To the right there are combo boxes for selecting scene and behavior, there are also controls for changing settings and adding overlays to the image and text description.
Tree View and Step-By-Step Instructions

On the left side of each view (see figure 22) there is a grey vertical rectangle with a small icon. Clicking this area opens up a tree view control that shows all the games, their scenes, the scenes behaviors, etc. This tree view is meant for more advanced users who want to quickly jump between different views within the application. Clicking on any object in the tree view makes the application jump to the corresponding view.

![Tree View and Step-By-Step Instructions](image)

Figure 22 - The Tree View and The Step-By-Step Instructions as it looks in the initial sketches.

On the right side of the application there is another grey vertical rectangle with a small question mark. Clicking this area opens up a step-by-step instruction view that guides new users through the process of mapping a game from start to finish. Tasks that are completed are marked with a check mark. The active task is marked with bold text and is explained in detail.

User tests

The main goal with the test was to see what worked, what didn’t and what had to be improved. Quantitative results will be presented from the tests but I will avoid presenting percentages and similar statistics in this thesis as that would require a higher number of participants in the tests. I also received a lot of qualitative data in the form of random comments, discussions and by studying the users.

The first test

The first test consisted of seven users with no or limited technical background. They were selected randomly from relatives and friends among employees at Tobii.

A summarized overview of the results is presented below in figure 23:

![User tests](image)

Figure 23 - A summary of the results from the first test.
In this table each row represents a task and each column corresponds to a user. All the users managed to solve all the tasks but with varying levels of success. The table illustrates this with three different colors:

- A white square means that the user knew what to do and had no problems.
- A grey square means that the users solved the task without any mistakes but had a hard time figuring out where to make the changes.
- A black square means that the user had a hard time figuring out how to perform the task and managed to change something that they shouldn’t.

Things to note from this test are that some tasks are hard and not entirely obvious for the users. Results indicate that parts of the application were easy to understand while other obviously needs improvement while other results are harder to interpret.

**Task 1 – Change the name of a scene**

During the first task I noticed that when I asked the participants to change the name of a scene, one of the participants clicked on the “Scene 1” text since he thought it was the scene name. I had named one of the test scenes “Options Menu”, this caused problems for one participant that thought the scene name was a button to open an options menu.

Several participants also clicked on the actual scene name in both the scene selection view and in the scene popup, to change the scene name instead on the “Rename” button next to the name in the popup.

**Task 2 – Remove a scene**

When I asked the users to remove a scene I noticed that some users had a hard time figuring out where the button to remove the scene was located. Some users, who had the scene popup open, tried going into the scene settings view which didn’t have any button to remove the scene. But in the end all participant found the correct way to remove the scene. Only one participant managed to create unintentional “damage” by removing a feature instead of a scene.

**Task 3 – Add a behavior**

Adding a behavior was an easy task for the majority of the participants but not to all. Two participants instead managed to edit existing behaviors without realizing it at first.

Several participants also said that they were thinking about clicking on the “Behavior” button on the scene selection view (used for behaviors common to all the scenes) but realized that this probably wasn’t related to a specific scene and tried opening the scene description view first, leading them to the correct view.

I also noticed that several participants clicked on e.g. the “4 Behaviors” text to come to the behavior editor, in both the scene popup and in the scene selection view.

**Task 4 – Change the settings of the new behavior**

This task was in the test since I wanted to see if the participants understood that the bottom grey area of the behavior editor was behavior specific. This was apparently quite easy to figure out since the only problem some of the participants had was that the behavior they added in the previous task wasn’t automatically selected and they had to figure out that they needed to select it first in the combo box.

**Task 5 – Add a screen resolution**

One interesting thing I noticed during this task was that when I asked the participants to add a specific new resolution, every single one of the participants first opened the resolution combo box in the scene selection view and then when they clicked “Add Resolution” and the resolution popup appeared they once again opened the combo box.
I asked the users why they opened the combo boxes and some told me that they didn’t know why and some told me that they wanted to see what was there. None of the participants did however think that they should open the combo box to add a new resolution.

**Task 6 – Change the settings of a scene**

The main problem that appeared was the fact that one scene was called “Options Menu” which caused one participant to try to click on the scene name to open the settings.

One participant clicked the “Settings” button on the scene selection view (used to for settings for the entire game) which caused (or would have, if it were implemented in the prototype) the wrong settings panel to appear. This indicates that there either needs to be made clearer which settings panel is which. Alternatively I have to make all the settings buttons open the same settings view, on which it then would be possible to either select application, game or scene settings.

**Task 7 to 11 – Feature Editor**

An example of a task that is hard to interpret is task number eight (change the size of a feature), five of the seven participants manage to do this task without any problem but the two remaining participants had a hard time figuring it out, making some changes to completely different settings.

Another thing I noticed was that none of the participants tried clicking on the image to add a new feature (which is one of the ways to add a feature). I asked the participants about this and most of them told me that it was because they saw the “Add Feature” button instantly so they didn’t realize there were multiple ways to add a feature. But the main problem I noticed still seemed to be that the participants didn’t realize you could click on the image. For example when I asked them to change the size of a feature, the ones that hadn’t learnt that they could click on the image found no way of changing the size and tried to start manipulating the threshold slider instead.

When I asked the participants to move a feature on the screen I noticed that almost half of the participants thought the “change type” square on the feature was used to move the feature. This made the next task extra interesting since the task was to change the type of a feature. Even though almost half the participants had already clicked on the change type square, none actually used it when I asked them to change the type; instead they used the combo box.

**Task 12 and 13 – Add and Copy Profile**

Adding a new profile and copying an existing one was apparently no problem. The only thing worth mentioning again was that the participants all first looked in the combo boxes again.

**The second test**

The second test was performed in the exact same way as the first. A difference between the two tests was the selection of users. Users for the second test were selected to have much experience in both gaming and in overall computer usage. If the users already owned several gaming peripherals and consider themself to be early adopters, it would also be considered a plus since that is one of the major types of consumers we are targeting with this application. We managed to find five participants that met these requirements that were willing to come to Tobii for a couple of hours to participate in a series of tests.

Before the actual test the participants had to fill out a form. The results from this form were as follows:

- All the participants were from KTH and were between 19 and 22 years old.
- They were all men.
- All except one participant had tried eye tracking before the test.
• All participants play computer games at least weekly.
• They all considered themselves advanced computer users (4 or 5 on a five grade scale).
• They are all keeping up to date with the latest technology.
• Only one user had good knowledge about StarCraft 2 (The game used in the test). The fact that only one user had good knowledge about StarCraft 2 didn’t matter too much since the tasks they performed didn’t involve any game specific parts anyway.

During the test I noticed that this new group of participants had a lot less trouble with the application than the first group of participants had. It was apparent that they had more experience with slightly unorthodox graphical user interfaces than the first group. They made just slightly less mistakes (see figure 24), but the difference was that they realized that they made a mistake faster and found the correct way to accomplish the task with fewer tries.

![Figure 24 - A summary of the results from the second test.](image)

The users from this test were also a lot more comfortable with experimenting and trying to click on things to see what they do. For example in the feature editor when I showed them that it was possible to interact with the image, all of the users immediately started moving, change size and removing the features. They all did the tasks 8 – 11 (manipulate the features) without me even having to tell them to do so. This was something the first group didn’t do. The first group seemed scared about breaking something.

After the test I asked the participants the following five questions:

• What was the application you just tested used for?
• What worked well and was easy to understand?
• What was difficult or hard to understand?
• Are scenes, features and behaviors good names?
• Do you have any tips for how to improve the application?

The results were as follows:

• Everyone understood what the application was for and its purpose.
• They found the application easy to use, especially for people with a lot of computer experience.
• There were a number of problems: adding features could be made clearer, right clicking on things in the GUI didn’t do anything, even though some users found it to be the most logical way to e.g. remove a scene or feature.
• Most of the users thought scenes and behaviors were good names for their purpose. But they had two suggestions for a different name for features: overlays or identifiers.
• The users also felt that some kind of tutorial or help system was needed for new users, especially in the feature editor and the behavior editor.

My overall impression was that experienced computer users who had user unconventional user interfaces before had no major problems with the applications. There were a number of areas where the application could be improved or made clearer, but there were no major problems.

Other results

During the development of the application we received orders and news that drastically changed the priorities and features of the application. When I started this project I was told that it would most likely be a research and development project but that there was a small chance that it would turn into a product. This meant that we didn’t have to follow all the correct software development practices meaning that we could develop the application very quickly.

About a third into the project we were told that the project looked promising and that Tobii will make a product out of it and sell it bundled with one of their eye trackers. This was both good and bad news. The good news was that it is fun to see your project be appreciated and be turned into a product. The bad news was that we now had to start following all the correct (and slower) ways of developing the application. This meant we had to make sure the code was easy for other to read, was extensible if we in the future wanted to add features and that there were no bugs. To make sure the quality of the application was high we had to start writing automatic tests to test every part of the application.

About two thirds into the project we got further news that our application would not be sold by Tobii any more, instead Tobii had started collaboration with a large and well known international company (let’s call it X) that would allow Tobii to sell their eye tracker (and our software) to the mass market branded with X’s name. This meant that we now had to change large part of our application to match X’s demands and requests. This meant making sure it was easy to quickly change the look of the application to match X’s overall company graphical design. We also had to prepare for the possibility that we had to integrate our application into their software in one way or another. The collaboration with X also meant that all different business units of Tobii suddenly were interested in our application and what we were doing.

Between these different stages of development we had to have discussions about how/if the different news would affect the applications development and how we were to proceed from there. The discussions main purpose was to decide and agree upon how the application should work and when it should do certain things e.g. save settings to disk. They also contained a lot of programming specific questions and topics such as: How should we write the code so it is easy for others to read? These types of topics are interesting but I feel like this thesis isn’t the right place to present our solutions to these kinds of problems. We did however discuss more general topics that are more interesting for this thesis. Below are some of the more interesting and, to this thesis, relevant topics presented.

Overall workflow and lifecycle of the application

The overall workflow and lifecycle was discussed several times and is something that we haven’t decided how exactly it should work. It is greatly affected by whether the third party company wants to integrate our software in their own software or not. The way we were thinking about doing it before we received the news about the third party company, was like this:

• The application shouldn’t start automatically when starting the computer (unless the user specifically tells the software to do so); instead the user should have to manually start the application by clicking on an icon on the desktop.
Clicking the icon starts the application. When the application starts there won’t be any window or anything, instead a tray icon will appear in the bottom right corner of the screen. A notification that the application has started will appear as a “balloon” above the tray icon for a couple of seconds. Clicking on the tray icon opens a menu where the user can choose to open the designer, pause the application or close it completely.

There was also talk about whether or not we should use the same tray icon as the rest of the Tobii software suite (not the official name). We want to separate our product from the rest of the suite since it is quite different from the other Tobii software. But at the same time it might be good to use the same tray icon, especially since the rest of the Tobii software suite has to be running, for our application to work.

**Overall data management in the application**

We discussed a lot about when and how the application should save and load the data. In the early versions of the application we had to manually save the data for all the scenes by pressing a save button. This allows a lot of control about whether or not you want to save. But it felt a bit strange since we couldn’t do things like e.g. undo changes, anyway. In the end we decided that the best way would be to do it like how a lot of online services and phone/tablet apps do it; we save everything almost continuously. In practice we don’t save everything continuously, but we consider the act of leaving a view as a time to save the changed object or settings.

![Figure 25 - The way we split up the game data into smaller pieces.](image)

This meant that we had to change the way the application manages the data (scenes, features, behaviors, etc.). From the beginning we sent around a large “game object”, which contained all the information about all the scenes, to all the different editors and parts of the application. Worth noting was that this was the “live” game object which our underlying engine used to interact with the game. This way worked, but it was unnecessary and the risks for nasty bugs were too large since every part of the application could manipulate any data. The way we solved this was by first creating a copy of the game object, to make sure that we didn’t use the live version. The next step was to make sure the different parts of the application only held the parts that they should, e.g. the feature editor should only hold a list of features and no information about the behaviors (see figure 25). By breaking up the game object into smaller pieces we were able to eliminate a lot of the potential risks that previously were present.

**Future proof the software and making it easy to add new functionality**

Our hope and ambitions is that this software will be used for several years and that it should continue to improve and expand. To make sure this is possible we had to make sure it was easy to add new features without rewriting large parts of the code.

We were also thinking about the possibility to make it possible for third party developers to add their own plugins and features to the software. To make this possible we had to start by avoiding hardcoding (e.g. defining the list of behaviors in the actual code) as much as possible. We solved most of these problems by using naming conventions, namespaces (containers used to group parts of the source code) and logical folder structures. The naming conventions e.g. allow us to automatically find new behaviors and their graphical views based on just their name
and their namespace without having to change anything in the existing application. This makes the software very flexible and makes it easy to add new functionality.

**Effects of major decisions coming from other parts of the company**

When the project went from a research project to a development of a product to be sold by a third party company, we had to rethink a number of things. Just the fact that the third party company might want to integrate our software into theirs had a number of implications to our priorities. One of the major implications was that our team suddenly grew from two to four (and later five) people, where one of the persons had the important role of handling all communications with the third party company. The news didn’t however change the practical work we needed to do; it mainly changed the order we needed to do things and it changed our priorities.

**Handling the case where the user changes to an odd screen resolution**

One thing we discussed early was how the screen resolution and profile system should work. The problem we noticed is that some games change their GUI based on the resolution. This causes problems for us since we want to use coordinates of interesting areas of the screen and if the GUI changes, these coordinates (even if they are relative instead of absolute) will no longer mark the same areas or points as before.

We solved this by forcing the user to create a mapping for all the common resolutions and aspect ratios (the relationship between the width and height of the screen). If the user then selects an uncommon resolution, our software will try its best to interpolate the values from the mapping with the closest aspect ratio.

![Figure 26 - The principle of the Resolution Dependent Repository](image)

Instead of storing multiple e.g. rectangles separately we have created a “resolution dependent repository” which holds different shapes and objects for different resolutions. With this repository, we can simply send it the resolution the user is using and which type of object we want and it will return an object that matches our preferences (see figure 26).

**Interview with a professional mapper**

The main reason for the interview with our own game mapper, Rickard Wallin, at Tobii was that I wanted to see what functionality he had found to be missing or was needed to speed up the workflow.

In the beginning, before the actual designer part of the application was ready to be used, he was manually editing the XML-files (which contain the saved scenes, behaviors and features). After editing the scenes in this way, his number one requirement of the designer was that it should give interactive feedback to the user. This feedback involves e.g. displaying the rectangles/regions you are editing as an overlay over the scene image instead of just seeing numbers in textboxes. Even better would be the ability to do as much of the editing of the position and size of features/behaviors, graphically on the actual scene image. As he said:
“The interactivity is the only difference between the designer and editing XML-files”

- Rickard Wallin, Professional Game Mapper at Tobii

He also mentions that a good function to add would be to have the rectangles (the ones used for feature and some behaviors in the editors) in different colors and maybe be able to manually set these colors. This would help to identify which rectangle you are currently editing and separate it from the others. Also the ability to add other shapes than rectangles would be very useful. It might even be possible in upcoming versions of the application to let the user draw a mask on the scene image using a brush, which would allow for very good control and would most likely be quite intuitive. Different colors for rectangles is however easy to implement immediately.

Rickard mentioned that he also had found that it was not clear for users what the purpose for the feature editor was. He had sent some screenshots of the application to a coworker, but the coworker didn’t understand what the feature editor was used for and how it works. Rickard mentions that it would be a good idea to try to make this clearer.

Another function he figured was missing in the application was the ability to send advanced combinations of mouse clicks and keyboard presses, instead of just single presses/clicks. This would be useful in many scenarios. One example of one such scenario could be when we want to send voice commands to other players in a game. This is usually done by pressing a series of keyboard buttons to navigate through a menu. If we instead wanted the ability to look at an area of the screen to activate a certain voice command, this would require the ability to send multiple keyboard presses to the game.

I asked Rickard if he thought keyboard shortcuts for adding scenes, features and behaviors would be a good idea and if it would speed up the process. He thought it would be a nice feature, but said it should be quite low priority right now since it wasn’t really necessary.

I also asked him if he thought the navigation was fast and easy enough to be able to work efficiently. He said the navigation was fine and it didn’t seem like there was any reason to e.g. implement the Tree View that I was thinking of.

**Final Sketches and Prototype**

After the user tests, application design meetings and discussion with the professional mapper I made a series of sketches. These sketches showed the final design of the application and illustrated all the main views and popups and how they all fit together.

Note that these were sketches and not the actual graphical design of the application. I intentionally skipped using a lot of colors, textures, etc. in the sketches because of two reasons: the first reason is the focus should lie on the interaction and overall workflow and not on the selection of colors and textures. The second reason is that the final colors, texture, fonts, etc. have not been determined. The selection of the colors and overall graphical style is very much up to Tobii and the third party company to decide. If the application will be branded Tobii, then it should match Tobii’s other applications and if it will be branded the third party company then the look and color scheme will be very different.

Below is a brief presentation of the views and how I intended for them to work. I also present the prototype and compare it with the sketches to see how far the prototype has to go before it can be considered ready for release. As previously mentioned, the prototype is an early version of the final application. It is not fully functional but all the necessary and important parts of the application work. It is my goal that the final application should look and work like I intended in the final sketches.
Common Changes

There have been a number of changes to create an even more unified design between the different views. Some of the changes are:

- Added question buttons on every view. Hovering over these will show a quick hint, pressing it will take you to your standard browser and to our online help system. The help system will explain every view thoroughly and there will be a possibility for users to ask questions.

- Tried to make it easier for users to find the controls they are looking for faster. I did this by making the design of the “Game Selection” view and the “Scene Selection” view very similar to each other. I also made the feature editor and behavior editor look resemble each other even more. These changes mainly involved rearranging controls.

- Renamed buttons in the application. The old design had a problem with naming conventions and e.g. mixed together names like “options” and “settings”.

- Added icons for even more buttons to further help the user.

Game Selection

I have changed this view a bit since the initial sketches. There are at least three major functional changes from the initial sketches:

- The ability to see which games are running and are currently connected with our application. This is illustrated with a big pause sign on top of the game image in the list. Pressing the pause sign eliminates the connection with the game, removing all eye tracking capabilities. The pause sign then turns into a play sign instead, which when pressed enables the eye tracking capabilities in that game again.

- The possibility to choose between reading the description (if you only want to play the game) and editing the mapping for the game. This choice is made by pressing the “Description” or “Edit” buttons. The selection can also be made by clicking the game object in the list which opens a popup where you can select the action to take.

- Changing the active profile, this is done by either pressing the “Profile: …” text on the game control or by pressing anywhere else to bring up the popup where you can change profile. By changing the profile, the application changes which scenes and behaviors are used.

![Select Game](image)

**Figure 27 - The Game Selection as it looks like in the final sketches.**

There have also been a couple of graphical changes (see figure 27). There is now an image in front of each games name which fades between the different scenes of the game. The purpose of this is to give an overview of the scenes in the game and also to add some level interactivity to an otherwise static view. To get the exact number of scenes quickly there is text for each game.
that tells you how many scenes there are in the game. A game that is running is recognized by a big pause sign on top of the image. Pressing the pause-sign disables the eye tracking for that game temporarily.

**The prototype**
This view works but has only basic functionality. It is e.g. only possible to open the designer (and not read the description) for the different games. It is also possible to add new games which is done in a popup that opens when pressing “Add Game” and selecting the running game from a list of processes.

**Scene Selection**
From the discussion I found that some users tried to click things that weren’t clickable, e.g. the “4 Behaviors” and “3 Features” texts for each scene to go to the feature- and behavior editors. Some users also made the incorrect assumption that the scene names were buttons when the scenes were called things like “Options Menu”. Figure 28 shows the new design.

![Scene Selection](image)

*Figure 28 - The Scene Selection as it looks in the final sketches.*

The changes I have made to this view are:

- Made the “4 Behaviors” and “3 Features” text into buttons that take you to the editors.
- Removed the scene number text/header (e.g. “Scene 1”) and instead enlarged the scene name to make it the header for each scene.
- Added buttons for settings and common behaviors for all scenes.
- Removed the ability to change resolution from this view since it doesn’t change anything.
- Added a red exclamation mark behind the behavior/feature texts if the scene doesn’t have any behaviors/features. This warns the user that this scene has no purpose since it can’t function without at least one feature and one behavior.

**The prototype**
This view (see figure 29) visually matches the sketches, but some functionality is missing. For example the description, settings, behavior and add scene buttons don’t have any function yet. It is also not possible to click on the feature and behavior texts to go directly to the editors. It might sound like most of the functionality is missing, but the two most important functions still work: selecting/editing profiles and adding/removing/editing scenes.
Scene Details

This view (see figure 30) hasn’t changed much from the initial set of sketches. There is only one big functional change; it is now possible to change the image of a scene if the game for some reason updates their GUI (meaning the features and behaviors might need to be updated). The way this is done is by right clicking the scene image to get the option to change the image. A popup appears, telling you to go in to the game and press a certain hotkey to capture a new image, when you select the option. Pressing the hotkey in the game returns you to our application.

Figure 30 - The Scene Details as it looks in the final sketches.

Other minor updates include: added a magnifying glass icon on top of the image when you hover over it, added a help button and removed the “Scene 1” text.
**The prototype**

This view (see figure 31) works just like the sketch, except for two things: you can’t edit the settings and the possibility to change image is not yet implemented.

![Figure 31 - The prototype of the Scene Details.](image)

**Feature Editor**

The feature editor’s largest change is that I have renamed features to “identifiers” (see figure 32). After talking to users it was apparent that the name wasn’t very good at explaining what the features purpose were. The name “identifiers” was a suggestion from a user and I feel that it is a better word. Identifiers are used to identify the scene is active in the game.

![Figure 32 - The Feature Editor as it looks in the final sketches.](image)

Another change is that the list of features (identifiers, but I will continue calling them features when talking about them in this text) has been removed. The reason for this decision is that the users had a hard time drawing the connection between the object in the list and the overlay on the scene image. Instead of having a list of features, I have made the feature overlays on the image more apparent by increasing their saturation and making them less transparent. The selected feature becomes very transparent but its border starts blinking to get the users attention. I have also improved the interactivity of the scene image:

- Features now have markers on all four corners and all four sides which can be used to change the size of the feature.
• Right clicking on a feature brings up a context menu with the choices: “Change Type” and “Remove Feature”.

• Right clicking anywhere else on the scene image brings up a context menu with the choices: “Add Feature” and “Clear All Features”

• Scrolling with the mouse wheel zoom the image in or out.

The prototype

This view (see figure 33) works exactly the same in the prototype as in the sketch, with the exception that there is no zoom functionality in the prototype.

![Figure 33 - The prototype of the Feature Editor.](image)

Behavior Editor

The behavior editor looks largely the same as in first sketch as well, but there are some changes (see figure 34). The largest change is the addition of a button for editing the hotkeys (triggers and controllers). This button, when pressed, opens a new popup where the user sees a list of which buttons activate and cancels the behaviors. The user also has the possibility to add new hotkeys.
The views for the behaviors have also changed, but their design isn’t finalized so I won’t present them here. The main difference with their design is that they now try to fill the space horizontally instead of vertically. This is needed since there isn’t much room in the behavior settings panel vertically but a lot of space in the horizontal direction. This in turn allows the user to see more of the data at once, without the need to scroll.

**The Prototype**

This view is fully implemented in the prototype with the exception of the possibility to zoom. Another limitation with the prototype is that some behaviors’ settings can’t be edited.
Figure 35 shows the behavior editor while editing a Screen Expose Behavior. On top of the image there are two red rectangles, corresponding to the two areas of the behavior. In the bottom of the screen it is possible to edit these two areas manually using numbers.

**Show Description**

The only change on this view is that I have added the ability to change to profile in the upper right corner (see figure 36). This allows the user to quickly browse through different profiles and compare their scenes and behaviors. I haven’t tested this view with users since I haven’t had time to implement it in the prototype.

![Figure 36 - The Description View as it looks in the final sketches.](image)

In the view in figure 36 you can read the description of three scenes. To view more scenes the user can scroll to the right. Each scene is described with an image and some text. The text and image have numbered markers that represent the behaviors. These markers can be also be used to e.g. indicate where to look.

**Edit Description**

This view has changed to match the design of the feature editor and behavior editor. The main difference is that I have moved all the controls, for editing the description, to the bottom of the screen (see figure 37). I’ve also added to possibility to change the profile. This view isn’t implemented in the prototype yet.

![Figure 37 - The view for editing descriptions as it looks in the final sketches.](image)
As can be seen in the image above, this view looks very much like the feature editor or behavior editor. This is intentional since I want the interaction in these views to be very familiar to the user. Once he/she has learnt to use one of them, the other two should be easy to learn.

**Overview and workflow**

The overall workflow is focused around five main views: Game Selection, Scene Selection, Scene Details, Feature Editor and Behavior Editor. These five views are represented in Figure 38 below by gray rectangles and are centered vertically.

![Diagram](image.png)

**Figure 38 - All the views in the application and the overall workflow**

The main workflow is illustrated by thick arrows pointing between the five main views. This path is the one that the user follows when he wants to map a game from start to finish. From each of the main views there are one or more child views.

The child views are either in the form of popups or they change the current windows view. They are represented by a solid black border. Some of the child views have their own child views which are represented by a dotted border.

All these views have their own designs and require to be tested by actual users in the future. During this project I tested the five main views, the profile editor and the resolution editor. From the tests I noticed that the users found this navigation to be quite understandable, even though Figure 38 might look complicated. The majority quickly understood e.g. if they were on the game level or scene level.

**The Prototype**

Not all views and child views are implemented in the prototype. So far I have implemented roughly half of them. The ones that are implemented in the prototype are represented in Figure 38 by their text being bold. The implemented views are not finished yet as mentioned above, but their basic functionality work.
Discussion

The work during this thesis has been largely focused on two parts; finding out how to best design the graphical user interface and developing the actual program. Before I started the project I was given two target groups: professionals who map a lot of games and the average user who only want to map one or a few games. The primary target group is the average users since they represent our consumers and the secondary target group is the professionals. I also added a third target group which is the gamers who just want to play the games with eye tracking.

I tried during the development to keep all the target groups need into consideration. The problem I decided to tackle was:

*How can I show the users the possibilities and limitations of the resulting application, while satisfying the needs of both target groups?*

This problem definition was the combination of several parts, which together formed the final problem definition. The parts were:

- How do I design an application that satisfies the targets group’s needs?
- How do I provide a pleasant user experience?
- How do I show what can be done in the designer?
- How do I show what can be done when playing the games using eye tracking?

I tried to find the answers to these questions through a process consisting of several steps that was focused on the users.

The process

The process started when I received the first sketch from my project leader. I treated this sketch as a starting point and produced a series of alternative sketches with varying designs. These designs where later evaluated by my project leader and the user experience experts at Tobii. One of the sketches was selected as the one to move forward with.

This sketch was then used as the initial design for the actual software application. At this stage of the project I started with the programming, which ran in parallel with the following user experience (UX) and interaction design (ID) parts of the project. The application had a working engine and only a very basic version of the feature editor, when I started. These were used as the foundation for the application that was developed.

Working in parallel with programming and UX/ID was an effective way of making progress as long as the programming didn’t halt. For the most part throughout the project, the programming ran smoothly and rapidly, but at the later stage when it was decided that the application would become a product, a lot of work had to go into making sure the code was up to production quality and a lot of time was put in to programming that didn’t result in any visual change to the application. This in turn resulted in that it was hard to perform the iterative development cycle which I intended to use from the beginning. The iterative development cycle would have involved user tests, followed by changes to the prototype, followed by user tests, followed by changes, and so on. To be able to get valuable data even when the GUI of the application didn’t change, I had design meetings with my project leader, discussed my sketches and application with the UX experts and talked to our professional game mapper. I received a lot of suggestions, comments and constructive criticism from these persons which was very helpful when trying to improve the sketches.
I also performed two series of user tests of the actual prototype before the development of the visual side of the program slowed down. I tried to make sure I had two different types of users for the tests. The first test was conducted on users with limited computer/technological background and the second test was performed on engineering students (studying IT) with large computer/technological background and who played a lot of computer games. This worked out well since I noticed clear differences between the two groups and the things that was unintuitive or unclear for beginners was easy to spot. It was good to be able to see which concepts worked and which didn’t. For example things like dragging a rectangle in the lower right corner to change the size was clear to the experienced users but not to the beginners. The user tests was important because of many reasons, but a large reason was that I didn’t want take it for granted that the users understood some of the basic concepts which I used throughout the application.

A drawback of the tests was that they were performed on the same version of the prototype. This meant that I couldn’t compare two different versions of the applications which would have given me a different kind of data. Optimally I would have liked to have performed multiple tests on several versions of the application, with multiple groups of groups of users for every version. Another thing that might be considered a drawback is the fact that the users had almost no knowledge of eye tracking and gaze interaction prior to the tests. This was apparent when I explained what e.g. behaviors are.

One thing that was left out of the entire process was the interaction design of the service, the part of the application that will run and give feedback when playing the game. The reason for this was that it was harder to program, some necessary parts of the engine wasn’t implemented yet and it would have taken a long time, time which I found was more valuable to spend on the designer part of the application. It would have been interesting to see how to best design this part of the application, but I’ll leave that to future exploration.

I feel like the tests managed to give me a good feedback on how to make the application user friendly for new users. The interview/discussion with our professional game mapper also helped to give a good indication to what he found was good and what needed to be improved or added to the designer and overall application. Worth noting is that even though I had a dedicated discussion with him, I got even more valuable information from him during the weeks he was at Tobii, helping out with our project.

I also did some research about the basics of eye movement, how the eye works, some novel ways of gaze interaction and the theory behind eye tracking. I used this information indirectly, meaning that I kept it in mind when designing the application. The reason for some of the research was to understand the types of interactions that are used with gaze interaction at e.g. Tobii. And also why they do some types of interactions but skip others; interactions that might seem good in theory but turn out to be quite bad in practice, e.g. clicking by blinking.

Throughout the entire project I have learnt a lot about eye tracking and gaze interaction from the employees at Tobii. This information was some of the most valuable information I found during the project. The employees really knew a lot about what works and what doesn’t, when it comes to eye tracking.

It is possible to say that resulting application from this project is a tool that enables users to add a new input modality to existing application. More specifically; it adds eye tracking to existing applications. This enables eye tracking to be used to a much larger extent than before, helping spread the modality. It would, even though that it is not our primary goal with the application, be possible to make games accessible for handicapped people or to create entirely new interactions with possibilities.

The tool enables eye tracking in applications were there was no intention, or even possibility, to have eye tracking support from the start. This could be considered this thesis primary contribution to the field of Human–Computer Interaction.
Conclusions

The main question I asked in this thesis was:

How can I show the users the possibilities and limitations of the resulting application, while satisfying the needs of both target groups?

In the final sketches I’ve tried to create an experience that is both efficient for the professionals that will map hundreds of games and easy to understand for the average users who will only map very few games. I feel like I’ve managed to accomplish this.

Answering the main question

I’ve made decisions during this project with the goal of satisfying the target group’s needs. These decisions and changes are primarily intended to aid a specific group, but will most likely benefit all types of users either directly or indirectly.

To satisfy the professional users’ needs, I have made a series of decisions, some of which are:

- Made sure that it is quick to navigate back and forth within the applications many views.
- There are also shortcuts and hotkeys to speed up common actions.
- Make all the information needed to perform tasks visible to the user without having to open menus or leave the views.

To satisfy the average users’ needs, I have also taken a number of steps:

- Divided the application into several smaller, task oriented views and removed all options and controls that weren’t relevant when performing a certain task.
- Created a uniform design where the controls are placed in predictable places.
- Created a help system which displays tips within the application or entire guides in the browser.

For all the target groups I’ve also created two simple, interactive and graphical editors (the behaviors and feature editors) which allow the users to perform the more complicated tasks with minimal need for text input. This allows the users to move objects, sliders and press buttons to accomplish most tasks, instead of writing in textboxes. If the users want to be more precise it is still possible to manually edit textboxes as well. They then receive instant feedback in the graphical editors. This is an example of another goal of mine, I’ve have tried to make it possible to perform several common task in multiple different ways, e.g. it is possible to go to the feature editors either directly from the “scene selection” view or through the “scene details” view.

For the users who only want to play the games I’ve created a simple description system which allows the user to quickly get a summary about how they can use this application together with the eye tracking in their favorite game.

The way I show the possibilities and limitations of the application is by dividing the application into task oriented views that perform a certain task. The way the final applications workflow works is like this:

Select Game - Select Scene – Edit Behaviors or Features

This workflow illustrates how the underlying system is built and the different concepts are presented one by one. At any stage in the workflow it is also possible to click the help buttons in the application to get detailed description.
Recommendations for future work

I’m quite certain that the resulting sketches will be a good foundation for further development of the actual application. Once all the changes are implemented and the prototype/application updated, it needs to be tested, both to see if the changes made the interaction and user experience better, and also to find things that I might have missed in the first two tests.

During this project there was a number of things that I wanted to do, but didn’t have time to do. I have compiled these things into a list of recommendations for future work with this application. These are ideas that have come to my attention either through discussions with employees at Tobii, when drawing the sketches or from the user tests.

To further improve the application there are several things that can be done.

- Invent more behaviors and design views for all behaviors
- Research how the feedback inside the game should best be designed
- Design and implement the cloud service for sharing profiles with other users. Maybe allow the users to share profiles with friends by sharing URLs or similar text strings.
- Perform the tests with significantly more people who fit the target groups better to get more accurate quantitative data
- Do another interview with the game mapping professional when he have mapped a hundred games in the updated application
- Create video tutorials for every view to complement the help system.
- Add support for game controllers. Maybe there is an interest in replacing the mouse/keyboard with game controller and eye tracking in games.
- Improve the feature detection algorithm to support dynamic/changing GUIs in games
- Add support for users with programming knowledge to add their own behaviors.
Bibliography


Detailed article about different ways to implement gaze gestures in practice.


Article about a project where they use the pupil size to control a game.


Article about how the Pupil Center and Corneal Reflection method for eye tracking works.


Book used in human-computer interaction courses at KTH.


Article about influencing a user’s decision based on gaze data.


Old but thorough study about the saccadic and smooth pursuit eye movement.


Well written book about human-computer interaction (HCI) and interaction design. Used in HCI courses at KTH.


An article about the potential privacy problem caused by eye tracking.


A recent book about eye tracking control in computer games by Veronica Sundstedt from Blekinge Institute of Technology


Tobii’s own description about how their eye tracking works.


Tobii’s summarized version of the history of eye tracking.


Collection of old methods for measuring the movement of the eyes.
Appendix A – Preliminary Sketches

Starcraft 2

Scene 1 - “Menu”
2 Behaviours

Scene 2 - “Ingame”
5 Behaviours

Scene 3 - “Ingame Menu”
3 Behaviours

World of Warcraft

Scene 1 - “Menu”
1 Behaviours

Scene 2 - “Raid”
7 Behaviours

Scene 3 - “Spells”
2 Behaviours

Scene 4 - “Mour”
3 Behaviours

Name: “Ingame”

Defining Features:
1
2

Behaviours:
3 Expose
4 Scroll
5 Shift

+ Add Scene

Add Game

Scene 1
Scene 2
Scene 3
### Appendix A – Preliminary Sketches

<table>
<thead>
<tr>
<th>Game</th>
<th>Starcraft 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scene</td>
<td>&quot;Ingame&quot;</td>
</tr>
<tr>
<td>Behaviour</td>
<td>&quot;Minimap&quot;</td>
</tr>
</tbody>
</table>

**Scene: "Ingame"**
- Defining Features
  - Feature 1
  - Feature 2
  - Feature 3
- Add Feature

**Behaviour: "Minimap"**
- Type: Exposed
- Active Region: Select
- Behaviour Triggers
  - Trigger 1
  - Trigger 2
- Add Trigger
Appendix B

- Iterations of the selected sketch
Appendix B – Iterations of the selected sketch
Appendix B – Iterations of the selected sketch
Appendix B – Iterations of the selected sketch

### Scene 1 - "Menu"
1. Look at any menu item to select it.

### Scene 2 - "Ingame"
1. Press Mouse3 for an Expose of the minimap.
2. Look at the side of the screen to scroll the view.
3. Press K to move the cursor where you are looking.

### Scene 3 - "Protoss"
1. Press Mouse to deploy a psion storm where you are looking.
2. Look at an area and press B to blind selected troops in that area.

---

### Editor

**Starcraft 2**

- **Scene:** Scene 1 - "Menu"
- **Behaviour:** Random

- Hide this behaviour
- Marker: Place Delete
- Show active region

- Look at any menu item to select it.
Appendix B – Iterations of the selected sketch

Starcraft 2 - Description

Scene 1 - “Menu”
1. Look at any menu item to select it.

Scene 2 - “Ingame”
1. Press Mouse3 for an Expose of the minimap.
2. Look at the side of the screen to scroll the view.
3. Press K to move the cursor the where you are looking.

Scene 3 - “Protoss”
1. Press Mouse1 to deploy a position where you are looking.
2. Look at an area and press B to bin selected troops in that area.

Edit Description

Starcraft 2
Scan: [Scene 1 - “Menu”]
Behaviour: [Behaviour 1]
[Hide this behaviour]
Marker: [Place] [Delete]
[Show active region]

Look at any menu item to select it.
Appendix B – Iterations of the selected sketch
Appendix B – Iterations of the selected sketch

Starcraft 2 - Description

Scene 1 - “Menu”
1. Look at any menu item to select it.

Scene 2 - “Ingame”
1. Press Mouse3 for an Expose of the minimap.
2. Look at the side of the screen to scroll the view.
3. Press K to move the cursor when you are looking.

Scene 3 - “Protoss”
1. Press Mouse to deploy a psi storm where you are looking.
2. Look at an area and press B to blind selected troops in that area.

Edit Description

Look at any menu item to select it.
Appendix B – Iterations of the selected sketch
Appendix B – Iterations of the selected sketch

[Diagrams showing various iterations and sketches related to Starcraft 2, with notes and comments on behaviors and settings.]
Appendix B – Iterations of the selected sketch