Author: Michael Edgar

Supervisor: Robert Sheehy

Second Reader: Paul Collins

An Evaluation of the use of an adaptive controller with an eye tracker to facilitate accessibility in gaming

B.Sc. (Hons) in Computing with Games Development

Table of Contents

[1 Abstract 3](#_Toc41825194)

[2 Introduction 4](#_Toc41825195)

[3 Video Games and Accessibility 5](#_Toc41825196)

[3.1 Introduction 5](#_Toc41825197)

[3.2 Video Games 5](#_Toc41825198)

[3.3 Accessibility and Technology 7](#_Toc41825199)

[4 Case Studies 9](#_Toc41825200)

[4.1 Case Study 1: Xbox Adaptive Controller 9](#_Toc41825201)

[4.2 Case Study 2: Logitech G Adaptive Gaming Kit 10](#_Toc41825202)

[4.3 Case Study 3: Tobii Eye Tracker 4C 10](#_Toc41825203)

[4.4 Case Study 4: Final Fantasy XIV 11](#_Toc41825204)

[5 Methodology 12](#_Toc41825205)

[5.1 Research Undertaken 12](#_Toc41825206)

[5.2 Research Question 12](#_Toc41825207)

[5.3 Proposed Project Implementation 12](#_Toc41825208)

[5.4 Functional Specification - User Stories 13](#_Toc41825209)

[5.4.1 Eye Tracking Game Control 13](#_Toc41825210)

[5.4.2 Voice Control Game Control 13](#_Toc41825211)

[5.4.3 Adaptive Controller Game Control 14](#_Toc41825212)

[5.4.4 Eye Tracking Cursor Control 14](#_Toc41825213)

[5.4.5 Multiple Voice Command Profiles for Different Games 15](#_Toc41825214)

[5.5 Use Cases 16](#_Toc41825215)

[5.5.1 Eye Tracking 16](#_Toc41825216)

[5.6 Prototype 18](#_Toc41825217)

[6 Implementation 23](#_Toc41825218)

[6.1 Voice and Eye control in Windows 23](#_Toc41825219)

[6.2 Raspberry Pi Research 24](#_Toc41825220)

[6.3 Python/C# Eye Tracker Implementation 25](#_Toc41825221)

[6.4 VoiceBot Speech Recognition 27](#_Toc41825222)

[6.5 Eye Tracker Mouse Movement 28](#_Toc41825223)

[6.6 Final Implementation 32](#_Toc41825224)

[7 Analysis and Conclusions 36](#_Toc41825225)

[7.1 Analysis 36](#_Toc41825226)

[7.2 Conclusions and Recommendations 37](#_Toc41825227)

[8 References 38](#_Toc41825228)

[9 Table of Figures 42](#_Toc41825229)

# Abstract

Video games are a form of interactive media that involve a player interacting with an artificial world on a computer. Accessibility is a measure of how accessible such video games are to all users, which can be linked with people that have disabilities. In video games, accessibility can relate to the design of the controller, the use of flashing images, the inclusion or exclusion of subtitles. Technology has provided several options to facilitate accessibility in video games. This includes voice to text conversions and vice versa, colour/high contrast filters and adjustable icons or text. Accessibility in relation to video games has been an issue for several decades despite many attempts to improve their accessibility. These solutions were often inaccessible due to high costs or lack of public knowledge.

This study evaluated several hardware devices working in conjunction with software applications, in order to improve the experience of video game players with reduced motor function. The study evaluated devices specifically created to solve an accessibility issue, as well as devices that were not. Additionally, hardware devices paired with accessibility software were accessed. These included an eye tracker with voice controls to improve the experience of players with reduced motor function. The implementation utilised the Tobii Eye Tracker 4C and the Xbox Adaptive Controller, along with the VoiceBot software application. C# programming scripts were developed to provide functionality to these devices. These were tested in the context of the online video game Rocket League.

The study has concluded that, while accessibility supports have become more accessible to video game players, it remains inaccessible due to their high cost. Similarly, the implementation that was created in this study concluded that, while the suggested setup worked within the video game Rocket League, the performance achieved hindered players as opposed to helping them. With further advancements of these technologies, along with a reduction of the costs, these accessibility supports could benefit video game players that do not have the support required to be able to play these video games.

# Introduction

This study aims to outline accessibility issues in video games, with a focus on previous attempts at improving accessibility within video games and an evaluation of their success or failure. To achieve this, an explanation of both video games and accessibility will be provided. Following from this, the study will outline the use of technology to improve accessibility, providing examples of their use. This will then be followed by an inspection of accessibility as it relates to video games; with a focus on the issues that exist and attempts to resolve them. Finally, the study will examine an attempt by the author to utilise existing technologies to create a controller setup for use in video games that aims to improve the experience of players with less motor function.

# Video Games and Accessibility

## Introduction

Video games are a form of interactive media that involve a player interacting with an artificial world on a computer. Accessibility is defined as how easy something is to reach, enter, use, see, etc. (Oxford University Press, 2019). Accessibility can be linked with people that have disabilities. To ensure accessibility for as many people as possible, many countries have laws in place to prevent the exclusion of those with disabilities.

In video games, accessibility can relate to the design of the controller, the use of flashing images, the inclusion or exclusion of subtitles, etc. Accessibility is something that is often overlooked in the design of video games. An example of where this is the case is referred to by Kevin Bierre et al. They present the example of a young boy with cerebral palsy who requires a specific adapted controller to play video games, but the boy in question struggles to find suitable video games as accessibility information is difficult to find (Bierre, et al., 2019).

## Video Games

A video game can be defined by the goal(s), the rules, and the environment of the game. Video games are often categorised into various genres such as puzzle, first-person shooter, couch co-op, etc.

The first known video game was implemented by Josef Kates and was called ‘Bertie the Brain’ (Smith, 2014). Kates, who at the time was a consulting engineer in Toronto 1950, made the game as an exhibit for attendees of the 1950 Canadian National Exhibit (Simmons, 1975). The game was a simple Tic Tac Toe game in which the player would attempt to beat the computer by getting three consecutive O’s in a horizontal, vertical, or diagonal line. This game was created to demonstrate power of computers (Smith, 2014).



Figure 3.1 Life magazine photo of Danny Kaye in front of Bertie the Brain at the Canadian National Exhibit in 1950 (Hoffman, 1950)

There were many games that followed Bertie the Brain, but the first video game created for entertainment purposes was ‘Tennis for Two’ (Smith, 2014). William A. Higinbotham created the game in 1958 for the Brookhaven National Laboratory (BNL) annual ‘Visitors Day’ to showcase the current research and development projects in the laboratory. Similar to Bertie the Brain, this game was purely an exhibition piece and was not released to the public. Tennis for Two ended up being dismantled after the 1959 Visitors Day (Nitray, 2015).

One of the first video games to be released to the public was Pong; based on the popular sport Ping Pong. The game consisted of two player-controlled panels and a ball that would travel back and forth between the two panels. The objective of the game was to make the ball object move passed the other players’ panel. It was released in 1972 for arcades (Wolf, 2007). An example of Pong is shown in figure 3.2 below.

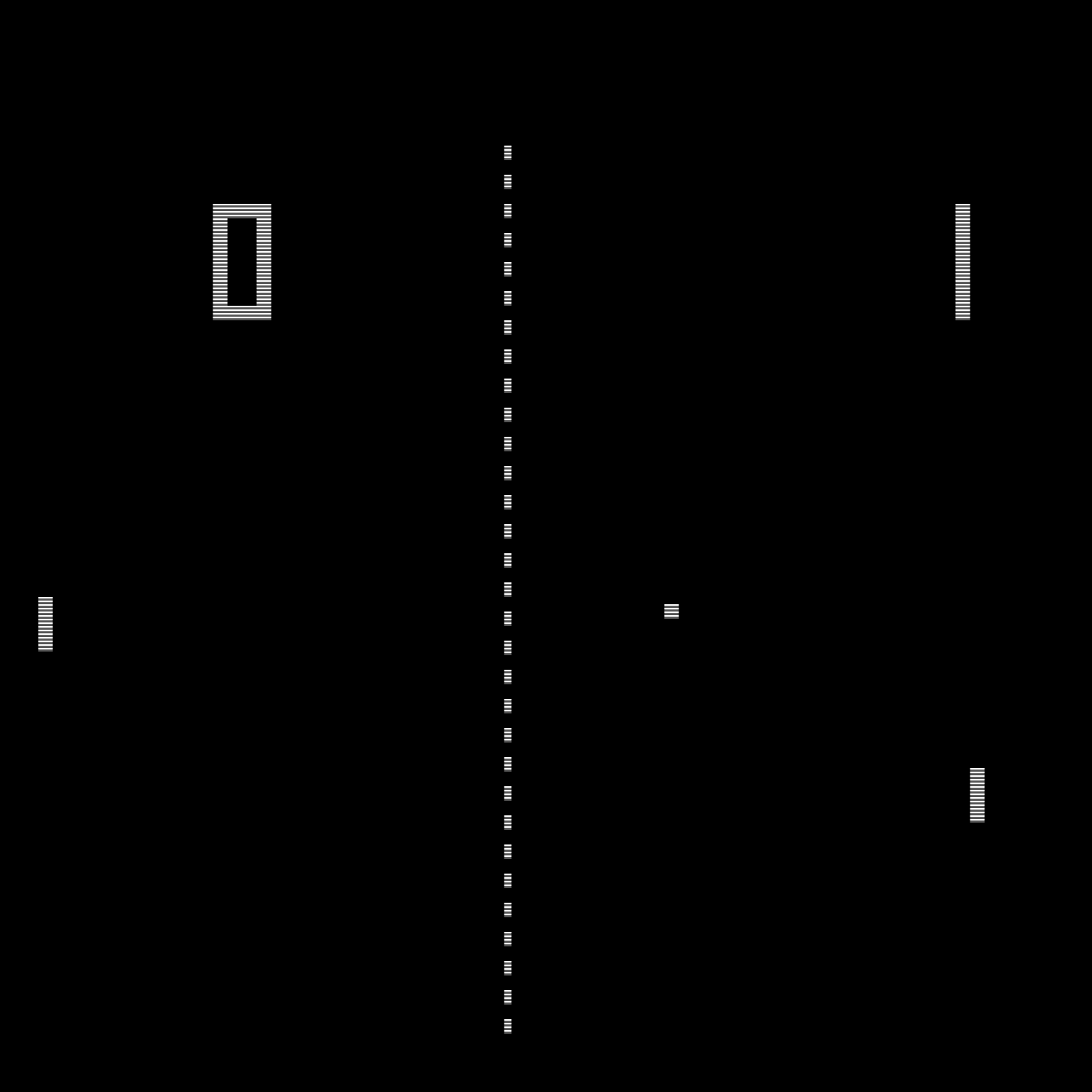


Figure 3.2: Pong Game

## Accessibility and Technology

Technology has shown a significant growth in the last thirteen years with the introduction of many “smart” devices such as the smart phone, smart television, and smart watch. With this growth, the desire to cater to accessibility needs has been highlighted. To cater to these needs, a number of accessibility options were introduced to various pieces of technology. This includes options such as voice to text conversions and vice versa, colour and high contrast filters and the ability to adjust the size of icons or text as desired. All of these options were introduced to help make technology more accessible to the general public.

The European Parliament and the Council of the European Union released a directive titled “Directive (EU) 2016/2102 of the European Parliament and of the Council of 26 October 2016 on the accessibility of the websites and mobile applications of public sector bodies”. This directive states that any website or app owned by a public body must be accessible to any person with a disability (National Disability Authority, 2017).

For as long as video games have existed, the issue of accessibility has been present for the people who play them (Wilds, 2020). Video games presented accessibility issues not only in the hardware configuration, but also in the software setup itself. One example of this would be Super Mario Bros from 1985, which lacked subtitles or audio control options; presenting issues for people who are deaf or hard of hearing. To combat these issues, solutions have been created at different levels of the development of video games. Many companies have accessibility options built into the games themselves. One of the first examples of this is the video game Real Sound: Kaze no Regret by Warp Inc. released in 1997 for the Sega Saturn video game console. What made this game unique is that it was marketed entirely towards blind players. The game relied entirely on the audio to guide players. Other companies add accessibility options as updates to existing games. This was the case for the game Days Gone by Bend Studio for the PlayStation 4 video game console, which was released in April 2019, which added multiple accessibility options in an update. This included the ability to replace repeated button presses with a button hold (Bend Studio, 2019). This option allows players that may have difficulty repeatedly pressing a button an easier alternative. Some users modify the games themselves to make the games more accessible. An example of this is the blindness accessibility mod that was created for Grand Theft Auto V. The mod was first showcased in March of 2020 in a YouTube video by the mod’s author, Liam Erven. The mod reads information from the game and uses audio to relay this information to the player. The mod adds a number of controls that tells the player information about their current environment, including the direction the player character is facing, their location and a list of nearby vehicles/characters and their location in relation to player character (Erven, 2020).

# Case Studies

## Case Study 1: Xbox Adaptive Controller

The Microsoft Xbox Adaptive Controller is a hardware device released in September 2018 that can be used to control video games on both the Xbox One and Windows PC. The controller was designed to cater for people who play video games with limited mobility. The controller meets accessibility needs with two main features: its configurability and the Co-pilot mode. The controller consists of 2 large programmable buttons, a D-pad, Xbox, View, Profile and Menu buttons, 2 USB ports, a 3.5mm headset jack, USB-C and DC power ports, a connect button and 19 3.5mm jacks for external inputs. The USB ports can be used to map the left and right analog stick inputs respectively from an external device, such as a joystick. The 19 3.5mm jacks correspond to each of the button inputs on a regular Xbox One controller; not including the movement of an analog stick. However, the controller does contain 2 extra inputs that can be mapped to any input of the controller including an analog stick direction input. The Xbox Accessories application can be used to modify inputs, while allowing for multiple configurations. The other main accessibility feature of the controller is the Co-pilot mode. This mode allows a player to connect a second Xbox One controller that would work alongside the Xbox Adaptive controller. This allows a person to use any of the inputs from a regular Xbox One controller while using the Adaptive controller; as well as allowing a second person to assist someone using the Xbox Adaptive controller.

The Xbox Adaptive Controller costs 89.99 Euro, 30 Euro more than a standard Xbox One Wireless Controller. This pricing makes it less accessible to lower income video game players that require this support to play video games. The Xbox Adaptive Controller’s main feature, its ability to be modified with extra inputs, is also one that makes it less accessible from a cost perspective. To make use of the Xbox Adaptive Controller as a full controller, users would be required to purchase additional peripherals, without which, the Xbox Adaptive Controller can only make use of less than half of the available inputs of a regular Xbox One Controller.

In summary, the Xbox Adaptive Controller is evidence of the growth in the field of video game accessibility, but it also highlights an important issue that needs to be overcome to make video games accessible for all users. The potential uses for this controller are dramatically reduced for video game players that cannot afford to pay several hundred Euro to invest into their video game controller.

## Case Study 2: Logitech G Adaptive Gaming Kit

The Logitech G Adaptive Gaming Kit is a set of buttons and triggers; intended for use alongside the Xbox Adaptive controller. The kit contains 4 light touch buttons, 2 variable triggers, 3 small buttons and 3 large buttons. Each of these buttons and triggers can be individually connected to an Xbox Adaptive Controller to create individual controller setups that suit the user. The kit also contains a number of mats and stickers to increase ease of use and allow the user to recognise each button/trigger more easily. The Adaptive Gaming Kit was released in November 2019 (Perez, 2019). The Adaptive Gaming Kit costs 99.99 Euro. This has a similar effect as the Xbox Adaptive Controller, as the cost would make the Adaptive Gaming Kit less accessible for video game players.

To make effective use of the Adaptive Gaming Kit and the Xbox Adaptive Controller, users would also require a device(s) to control the input of the left and right analog sticks. This means that users would need to spend 189.98 Euro, alongside the cost of the device(s) that control the analog sticks. This price is over three times that of a regular Xbox One Wireless Controller, which highlights how much extra a video game player with reduced motor function would have to play to have a gaming experience comparable to that of video game players that do not require an accessibility controller.

## Case Study 3: Tobii Eye Tracker 4C

The Tobii Eye Tracker 4C is an eye tracking device released in October 2016 (Tobii Tech, 2016). The eye tracker can be used to control a number of games that support the eye tracker, as well as for general navigation in the Windows operating system. The eye tracker is marketed towards people who desire additional functionality during video games. They advertise the ability to “look around” within a game using the motion of the player’s head, along with a number of other features that aim to improve the video game experience. Alongside its ability to be used as a peripheral for video games, Microsoft has added accessibility supports that allow eye tracking to be used in Windows 10 with the Tobii Eye Tracker 4C (Microsoft Accessibility Blog, 2017).

The eye tracker connects to any PC or laptop using a USB 2.0 connection and proprietary eye tracking software, Tobii Eye Tracking. The Tobii Eye Tracking software can be used to calibrate the eye tracker and change a number of settings. Once connected and calibrated, the Tobii Eye Tracker 4C requires no further setup to be used for Windows navigation, Windows Eye Control or for use in video games.

## Case Study 4: Final Fantasy XIV

Final Fantasy XIV is an online multiplayer game that was released by publisher Square Enix in 2010. The game allows users to connect to servers of between 6,000 and 15,000 active players (Price, et al., 2020). This game is an example of how video games can become more accessible with updates and changes. In 2013, the game was awarded the Mainstream Game of the Year Award, for being the most accessible mainstream game at the time. They were given this award by The AbleGamers Foundation, a non-profit charity that works to improve accessibility in video games. The award was given three years after the game was initially launched, due to changes that had been made to the game to increase accessibility in-game (Grubb, 2014). This trend continued for Final Fantasy XIV for years after receiving this award. In May 2018, the developers of Final Fantasy XIV released a patch that included extensive color filtering options to make the game more accessible to colorblind users (Square Enix, 2018). These settings allowed players to specify the type of colorblindness they suffered from and adjust the level at which the color filter was applied. A screenshot of these settings can be seen in the figure below.

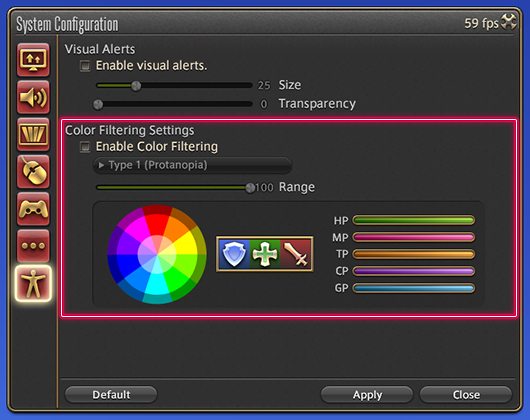


Figure 4.1 Final Fantasy XIV Color Filter Settings (Square Enix, 2018)

# Methodology

## Research Undertaken

The research undertaken for this project centred around accessibility issues and previous attempts to overcome these issues. Attempts around technology in general and the world of video games to overcome accessibility issues were researched; both to examine the accessibility issue posed and the success of the proposed solution. These solutions were grouped by their proposed usage: those aimed at improving general technology use and those aimed specifically for use with video games.

In relation to video game-based solutions, the source of the solution was also examined. These solutions fell into three categories in that respect: those created by a company to act as a peripheral for their existing video game console/video game, those created by a company to work in conjunction with another company’s existing video game console/video game and those created by a user to facilitate their use of a video game console/video game.

The last topic researched was the use of existing peripherals for video games as a means of solving an accessibility issue. This aimed to identify existing resources and how they can be applied to alternative situations. An example of this is the Tobii Eye Tracker 4C, a consumer eye tracking device that is marketed for use in video games, but can be used to enable Windows’ Eye Control; which allows users to control their PC or laptop with their eyes alone.

## Research Question

Can eye tracking and voice control be implemented alongside existing accessibility peripherals for use in a video gaming context. An evaluation of existing technologies, how they may work alongside each other and how beneficial they may be to players with less motor function ability.

## Proposed Project Implementation

This project aims to use a number of hardware devices working together with software applications in order to improve the experience of players with less motor function ability. This will also require software that communicates with the hardware devices and translates their inputs into inputs a video game may understand.

## Functional Specification - User Stories

### Eye Tracking Game Control

**Description:**

As a player, I need the ability to have my eye movements tracked so that I can control the game.

**Priority:** Must Have

**Estimated Completion Time:** 7hr

**Acceptance Criteria:**

The eye movements of the player are received by the eye tracker.

The data containing the eye movements is sent from the eye tracker to a script.

The script receives the data containing the eye movements and sends it to the game as an input.

The game receives this input and uses it to affect the game in a meaningful and noticeable manner.

### Voice Control Game Control

**Description:**

As a player, I need the ability to use voice commands so that I can control the game.

**Priority:** Must Have

**Estimated Completion Time:** 2hr

**Acceptance Criteria:**

The microphone of the device running the game receives audio from the user.

The user audio data is sent from the microphone to the software controlling voice commands.

The software controlling voice commands checks if the user audio contains a voice command registered for the game.

The software controlling voice commands sends an input matching the given voice command to the game.

The game receives this input and uses it to affect the game in a meaningful and noticeable manner.

### Adaptive Controller Game Control

**Description:**

As a player, I need the ability to use the Xbox Adaptive Controller so that I can control the game.

**Priority:** Must Have

**Estimated Completion Time:** 1hr

**Acceptance Criteria:**

The Xbox Adaptive Controller connects to the device running the game.

The Xbox Adaptive Controller receives inputs from the player.

The player inputs are sent from the Xbox Adaptive Controller to the game.

The game receives the player inputs and uses them to affect the game in a meaningful and noticeable manner.

### Eye Tracking Cursor Control

**Description:**

As a user, I need to be able to control the device’s cursor with the eye tracker, so that it can navigate the operating system without a mouse.

**Priority:** Should Have

**Estimated Completion Time:** 4hr

**Acceptance Criteria:**

The eye movements of the player are received by the eye tracker.

The data containing the eye movements is sent from the eye tracker to a script.

The script receives the data containing the eye movements and uses it to modify the cursor’s position.

### Multiple Voice Command Profiles for Different Games

**Description:**

As a player, I need the ability to have separate profiles of voice commands for different games so that I can control multiple games with voice commands.

**Priority:** Should Have

**Estimated Completion Time:** 1hr

**Acceptance Criteria:**

The software controlling the voice commands has multiple profiles for different games, containing unique voice commands.

The voice commands from a profile, do not affect a game outside of the game’s profile.

The profiles can each send the same input for their given game.

## Use Cases

Note: The game itself is considered an actor and not part of the system as the system interacts with the game but is not run by the system

### Eye Tracking

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | **Eye Tracking** | |
| **Use Case Id** | 1.1 | |
| **Priority** | High | |
| **Source** | User | |
| **Primary Business Actor** | User | |
| **Other Participating Actors** | Game, Eye Tracker | |
| **Description** | This function takes the position of the user’s gaze in relation to the display and uses it to determine a new position for the cursor | |
| **Preconditions** | The Eye Tracker is connected | |
| **Trigger** | No | |
| **Typical Scenario** | **Actor Action** | **System Response** |
|  | **Step 2:** The eye tracker connects to the system  **Step 4:** The user gazes at a point on the display.  **Step 6:** The eye tracker sends the user’s eye tracking data to the system.  **Step 9:** The game receives the message from the system and uses that movement to perform some action in the game, e.g. getting a character to turn in the direction specified | **Step 1:** The system requests a connection to the eye tracker.  **Step 3:** The system confirms the eye tracker is connected.  **Step 5:** The system requests the user’s eye tracking data from the eye tracker.  **Step 7:** The system uses the user’s eye tracking data to determine which direction the user is looking.  **Step 8:** The system sends a message to the game that simulates a mouse movement in the direction determined in Step 7. |
| **Alternate Scenarios** | **Actor Action** | **System Response** |
| **Eye Tracker Not Connected** |  | **Step 1:** The system requests a connection to the eye tracker.  **Step 2:** The system fails to connect to the eye tracker.  **Step 3:** The system displays an error message and exits. |
| **Eye Tracking Data is Null** | **Step 2:** The eye tracker connects to the system  **Step 4:** The user gazes at a point on the display.  **Step 6:** The eye tracker sends a null response. | **Step 1:** The system requests a connection to the eye tracker.  **Step 3:** The system confirms the eye tracker is connected.  **Step 5:** The system requests the user’s eye tracking data from the eye tracker.  **Step 7:** The system displays an error message and exits. |
| **Conclusions** | The game has used the user’s gaze position to control some aspect of the gameplay | |
| **Post conditions** | The user can control the game with their gaze | |
| **Business Rules** | None | |
| **Implementation Constraints** | None | |

## Prototype

|  |  |  |
| --- | --- | --- |
| **Prototype Number** | **Start Date** | **Finish Date** |
| 1 | 04/12/2019 | 09/12/2019 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Install Unity Hub | Complete |
| 2 | Install Unity 2019.2.6f1 | Complete |
| 3 | Install Rider 2019 IDE | Complete |
| 4 | Open Unity Hub | Complete |
| 5 | Create new 3D Unity Project | Complete |
| 6 | Download Tobii SDK from Unity Asset Store | Complete |
| 7 | Import Tobii SDK from Unity Asset Store | Complete |
| 8 | In Tools click “Add Tobii SDK Demo Scenes to Build” | Complete |
| 9 | Run Sample Scenes to learn about the Tobii SDK | Complete |
| 10 | Create “Scripts” and “Prefabs” folders | Complete |
| 11 | Create “AsteroidController”, “EyeTracking”, “GameManager” & “PlayerController” Scripts | Complete |
| 12 | Read documentation at <https://tobii.github.io/UnitySDK/scripting-api> | Complete |
| 13 | Populate Scripts with code shown in Figures 5.1 – 5.4 | Complete |
| 14 | Create 3D Sphere with the properties shown in Figure 5.5 and save as a prefab named “Asteroid” | Complete |
| 15 | Make the default Directional Light a child of the default camera object. Rename the camera to Player and save as a prefab with the properties shown in Figure 5.6 | Complete |
| 16 | Create a UI Text object named “Exit Text” | Complete |
| 17 | Move the Exit Text Anchor and Position to Bottom, Left and change the Text to “Press ‘Esc’ to Quit” | Complete |
| 18 | Create an UI Image object name “Current Eye Position” | Complete |
| 19 | Add the EyeTracking script as a component of the Current Eye Position | Complete |
| 20 | Run the Game and test it works | Complete |

Using the Tobii Eye Tracker, along with Microsoft Services Voice, a game can be controlled without any physical input from the player, e.g. button presses. This project provided a clear use case for the eye tracker in video games; particularly how the eye tracker can be used without the need of a physical input device. The benefit of this is the ability of players without the ability to move their arms can be given the opportunity to play video games. This shaped the rationale for the rest of this project and provided the idea for the Eye Tracking Windows Control shown below.

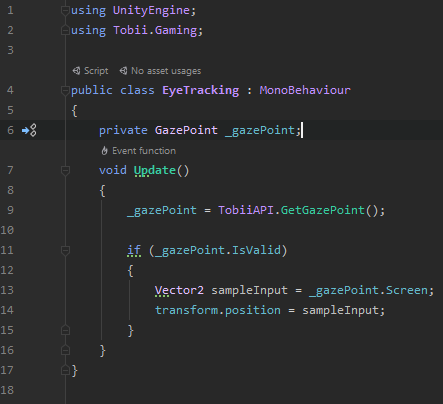


Figure 5.1 EyeTracking Class

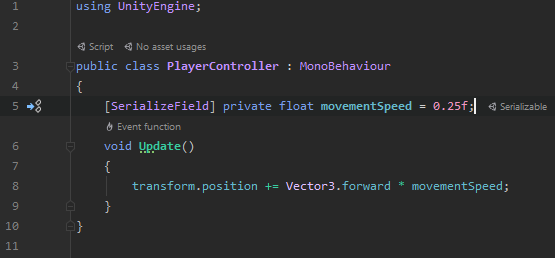


Figure 5.2 PlayerController Class

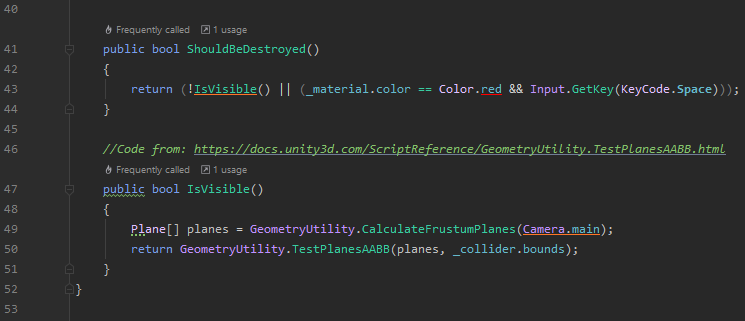


Figure 5.3 AsteroidController Class

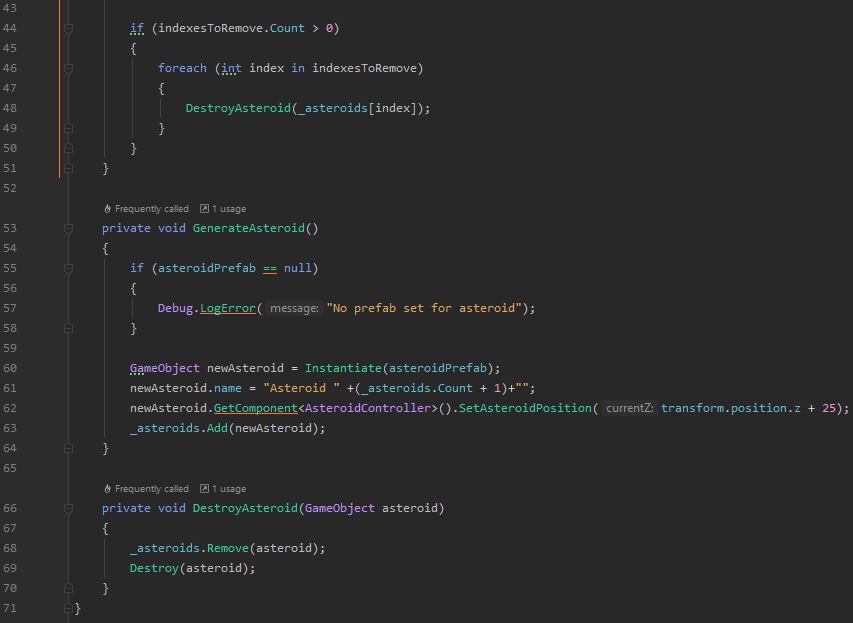
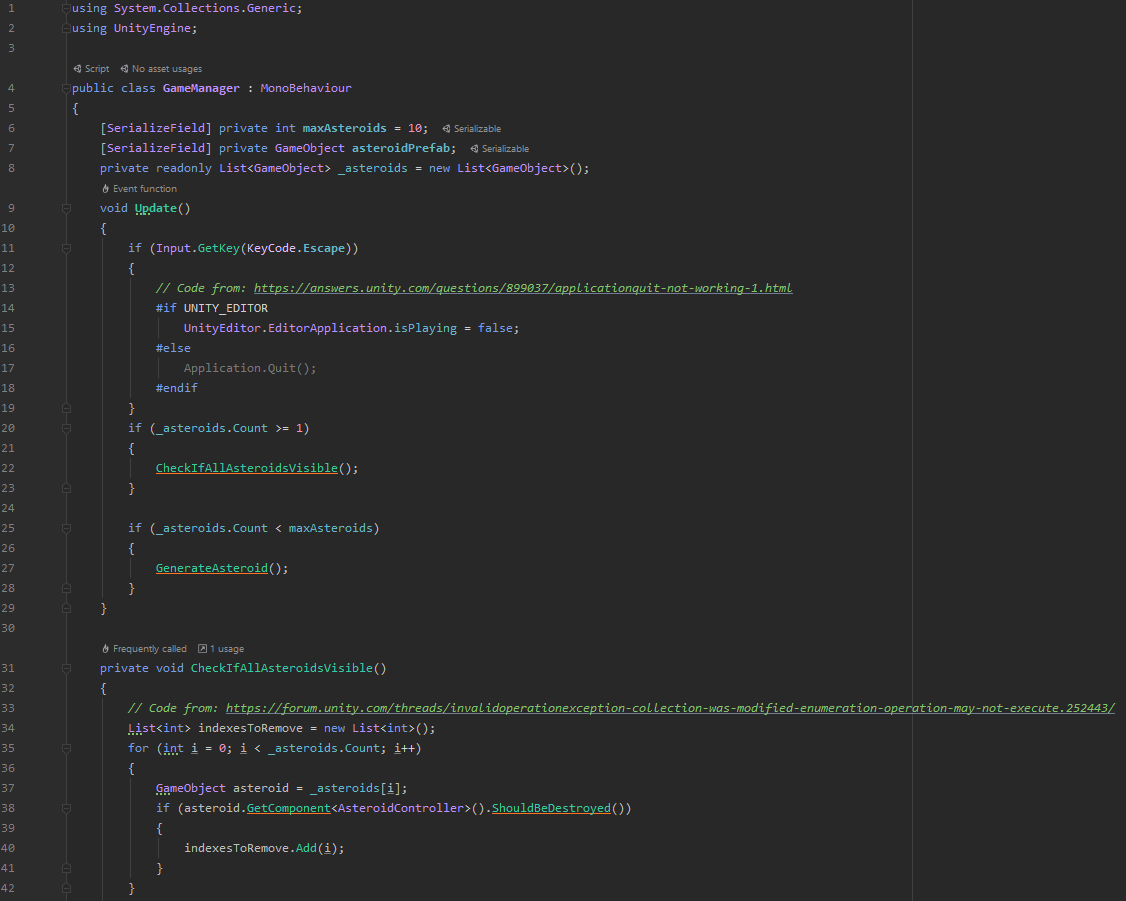


Figure 5.4 GameManager Class

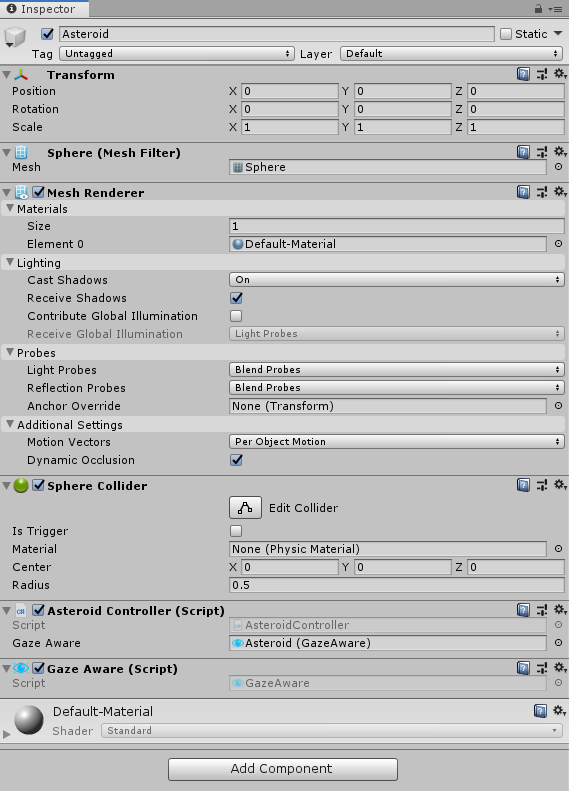


Figure 5.5 Asteroid Prefab Properties

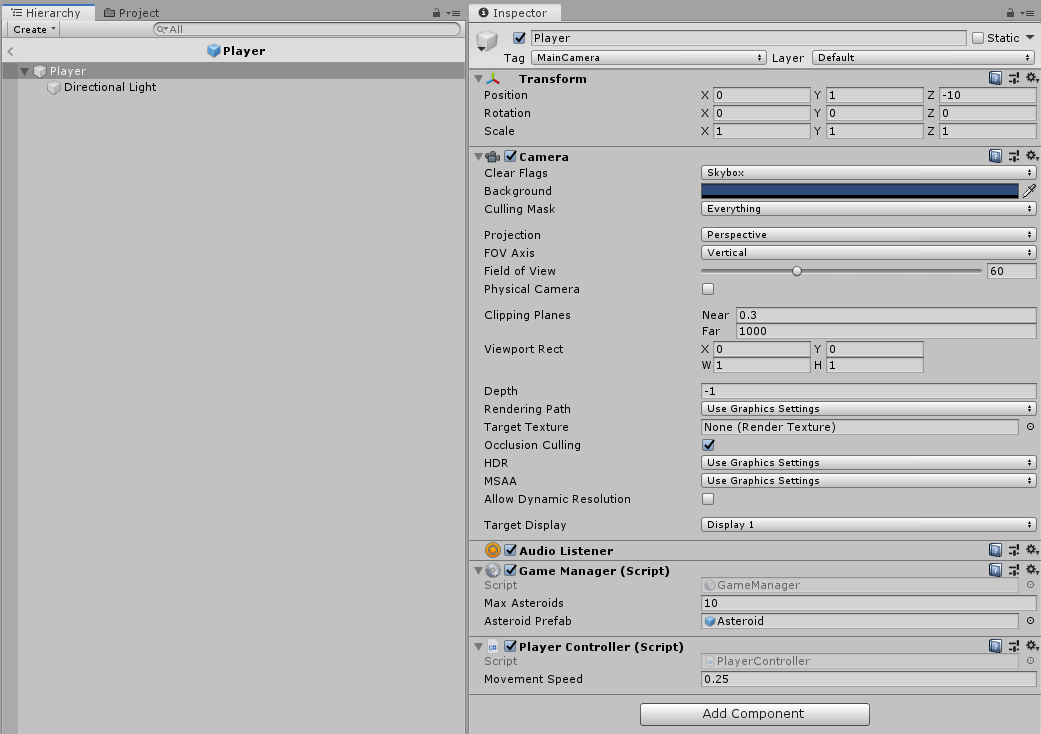


Figure 5.6 Player Prefab Properties

# Implementation

## Voice and Eye control in Windows

|  |  |  |
| --- | --- | --- |
| **Sprint Number** | **Start Date** | **Finish Date** |
| 1 | 02/02/2020 | 07/02/2020 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Install Tobii Eye Tracking Application | Complete |
| 2 | Configure Eye Tracker | Complete |
| 3 | Enable Windows Eye Control | Complete |
| 4 | Experiment with Windows Eye Control to learn functions and possible uses within project | Complete |
| 5 | Enable Windows Speech Recognition | Complete |
| 6 | Experiment with Windows Speech Recognition to learn functions and possible uses within project | Complete |
| 7 | Configure Xbox Adaptive Controller | Complete |
| 8 | Connect Tobii Eye Tracker to Xbox Adaptive Controller and test functionality | Completed with Errors |

This sprint attempted to evaluate the current voice and eye tracking functionality within Windows 10, along with the functionality of the Xbox Adaptive Controller. This was to assist shape where progress was required in terms of determining what functionality would be incorporated and what functionality would have to developed. It was determined that the Xbox Adaptive Controller could not directly process the inputs from the Tobii Eye Tracker. To solve this issue, it was decided to process the inputs from the Tobii Eye Tracker and use an external device to feed these inputs directly to the Xbox Adaptive Controller. From discussions with my FYP supervisor, it was determined to research the Raspberry Pi as a possible solution for this issue. The Raspberry Pi would process the inputs from the Tobii Eye Tracker and then feed those inputs into the relevant input within the Xbox Adaptive Controller, in a manner similar to a switchboard.

## Raspberry Pi Research

|  |  |  |
| --- | --- | --- |
| **Sprint Number** | **Start Date** | **Finish Date** |
| 2 | 11/02/2020 | 21/02/2020 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Research Raspberry Pi and potential sources within the college of obtaining one | Completed |
| 2 | Research local companies that work with Raspberry Pis | Completed |
| 3 | Contact IMaR to arrange meeting to discuss FYP and Raspberry Pi integration | Completed |
| 4 | Meet with ImaR Strand Leader of RFID & Internet of Things | Completed |

This sprint focused on the possible implementation of a Raspberry Pi for processing the inputs from the Tobii Eye Tracker for use in a video game that would not support it natively. After many discussions, it was concluded that the Raspberry Pi would only be necessary were the project to be introducing this setup on an environment other than PC, e.g. Xbox One Console. For this reason, alternative implementations were researched. Following on from a meeting with IMaR’s Strand Leader of RFID & Internet of Things, a python script was identified that would run from the command line and process the data from the Tobii Eye Tracker. This would form the basis of the next sprint.

## Python/C# Eye Tracker Implementation

|  |  |  |
| --- | --- | --- |
| **Sprint Number** | **Start Date** | **Finish Date** |
| 3 | 09/03/2020 | 21/03/2020 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Fork Tobii Eye Tracker project from GitHub (sajidbaloch, 2018) | Completed |
| 2 | Install GitHub Desktop | Completed |
| 3 | Clone Tobii Eye Tracker project | Completed |
| 4 | Run Python Script from Command Line | Completed with Errors |
| 5 | Troubleshoot Python Script Error | Completed |
| 6 | Research C# Tobii Implementation | Completed |
| 7 | Create C# Console Application in Rider | Completed |
| 8 | Import Tobii Interaction NuGet package | Completed with Errors |
| 9 | Troubleshoot NuGet Rider import issue\* | Completed |
| 10 | Install Visual Studio 2019 | Completed |
| 11 | Create C# Console Application (.Net Core) in Visual Studio | Completed |
| 12 | Import Tobii Interaction NuGet package | Completed with Errors |
| 13 | Troubleshoot NuGet Visual Studio import issue (  <https://stackoverflow.com/questions/38714898/visual-studio-2015-cant-find-nuget-package-references-and-dlls-in-packages-fold>  <https://docs.microsoft.com/en-us/nuget/quickstart/install-and-use-a-package-in-visual-studio>  ) | Completed |
| 14 | Create C# Console Application (.Net Framework) in Visual Studio | Completed |
| 15 | Import Tobii Interaction NuGet package | Completed |
| 16 | Complete following tutorial for a Console Application: <https://developer.tobii.com/consumer-eye-trackers/core-sdk/getting-started/> | Completed |
| 17 | Run application to ensure it works | Completed |

This sprint focused on the task of parsing the data from the Tobii Eye Tracker in its simplest form. This means parsing the X and Y coordinates on the screen where the Eye is looking. At first, this was attempted by using the Python script which was obtained in the previous sprint. However, this proved to be an unsuitable approach as, after troubleshooting the issue with task number 8, it was established that in order to parse the data from the eye tracker that was being using, a special license would be required, which required payment to use ([Tobii], 2018). From this, it was established that a similar approach that would not require the license could be implemented, requiring the use of C# instead of Python. This caused a further issue when attempting to import the required NuGet package. Neither Rider nor Visual Studio seemed to recognise the package after it was imported. It was identified that this was caused by the project template that was used. Originally, a .NET Core Console Application was used when a .NET Framework Console Application was required to use NuGet packages. Once this issue was overcome, a tutorial was used to retrieve the required data from the Tobii Eye Tracker.

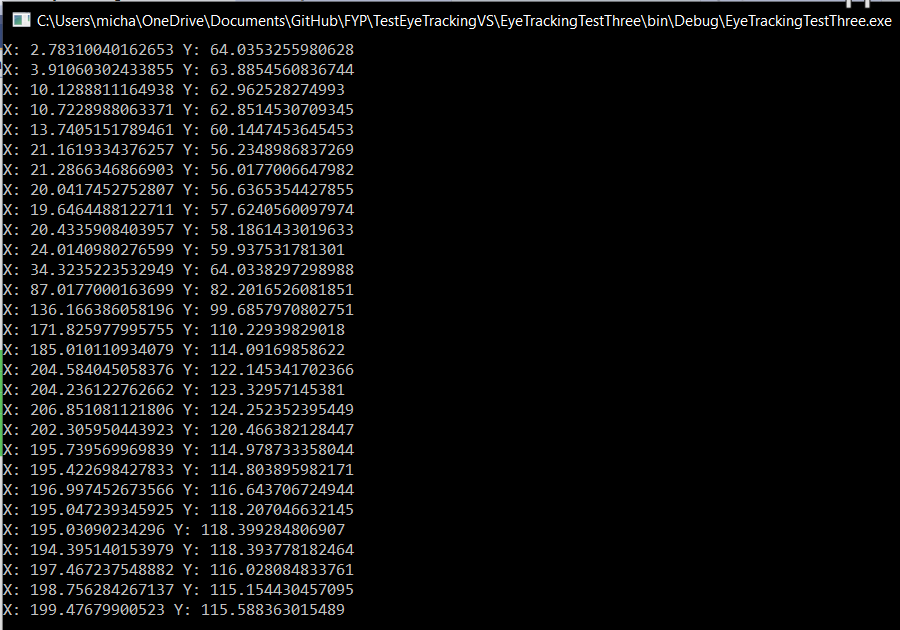


Figure 6.1 Tobii Eye Tracker Gaze Data Coordinates

\*Note: troubleshooting from the following websites:

<https://www.jetbrains.com/help/resharper/Finding_Exploring_and_Installing_NuGet_Packages.html>

<https://www.jetbrains.com/help/rider/Reference_Windows_NuGet.html>

<https://stackoverflow.com/questions/3304741/getting-type-or-namespace-name-could-not-be-found-but-everything-seems-ok>

## VoiceBot Speech Recognition

|  |  |  |
| --- | --- | --- |
| **Sprint Number** | **Start Date** | **Finish Date** |
| 4 | 23/03/2020 | 27/03/2020 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Go to <https://www.voicebot.net/Download/> and download VoiceBot Installer. | Complete |
| 2 | Install VoiceBot by running installer and follow steps | Completed |
| 3 | Open VoiceBot and choose download profile | Completed |
| 4 | Download Rocket League Profile | Completed |
| 5 | Test Rocket League Profile in game | Completed |
| 6 | Click Edit Profile and Add button in dialog box | Completed |
| 7 | Name command “Boost” and ensure Use name as command checkbox is set to true. | Completed |
| 8 | Click Add Group button and name group “New” | Completed |
| 9 | Click Add, then Press Mouse Button and then open the dropdown menu and choose Left Mouse Click and then OK | Completed |
| 10 | Press OK again and test new command in game | Completed |
| 11 | Ensure profile runs on correct application only | Completed |

In this sprint, the VoiceBot application was investigated for its potential use within this project. For licensing reasons, the owners of the application were contacted to get permission for its use. Following this, a profile was tested for the game Rocket League. It was established that, after a small adjustment period, the application was perfectly suited for this project. An individual profile could be set up for any game that was played, which would hijack the controls of the game based on the voice commands that were set.

## Eye Tracker Mouse Movement

|  |  |  |
| --- | --- | --- |
| **Sprint Number** | **Start Date** | **Finish Date** |
| 5 | 12/05/2020 | 21/05/2020 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Retrieve eye tracking script from previous script | Complete |
| 2 | Modify script to move cursor based on GazePoint position | Completed with Errors |
| 3 | Troubleshoot GazePoint data inaccuracy and “jumping” | Completed |
| 4 | Modify script to move cursor based on EyePosition | Completed with Errors |
| 5 | Troubleshoot EyePosition data inaccuracy | Completed |
| 6 | Modify script to move cursor based on HeadDirection | Completed with Errors |
| 7 | Trouble HeadDirection missing data | Completed |
| 8 | Modify script to move cursor based on filtered GazePoint position | Completed with Errors |
| 9 | Test script within Rocket League game | Failed |

In this sprint, the script created in Sprint Number 3 was used to move the mouse cursor. This would then allow the user to replace mouse controls in game with movements recorded by the eye tracker. It was decided to keep the GazePointStream from the previous implementation and change it to override the cursor position based on the position of the gazes it recorded (<http://www.blackwasp.co.uk/MoveMousePointer.aspx>). However, this implementation had two main issues; firstly, it was not accurate, i.e. the position it was moving the cursor to was not the position of the gaze, as shown in Figure 6.2 below. The second issue was that the cursor would jump rapidly from one position to another due to the frequency of checks by the eye tracker. While troubleshooting these issues, two possible alternatives were located in a Tobii sample program that could resolve these issues (<https://github.com/tobiitech/core-sdk-docs/blob/master/samples/Streams/Interaction_Streams_103/Program.cs>). The first was to replace the GazePointStream with an EyePositionStream and use an average of the LeftEyePosition and RightEyePosition to determine the cursor position. However, this resulted in the cursor jumping to a random point on the screen due to the EyePosition not reflecting where the user was looking. This is illustrated in Figure 6.3 below. The other solution was to use a HeadPoseStream to determine the cursor position by the direction of the user’s head. In terms of accuracy, this solution would fit best with the implementation seen in games that natively support the eye tracker. This implementation would allow the user to ‘look around’ in game. The issue with this implementation was that the head direction recorded was mostly NaN, which is the equivalent of null, despite movement of the user’s head position and direction, as shown in Figure 6.4. After troubleshooting, this was found to be a bug with the eye tracker not detecting the user’s head while the script was running. This was resolved by recalibrating the eye tracker; however, it was found that the head direction values returned by the eye tracker were all less than zero. This was an issue as the Cursor position was in the form of an integer value for the X and Y co-ordinates. To resolve this, the head direction value was multiplied by a factor of ten to make them integer values. This led to cursor movements being seemingly unrelated to the direction of the user’s head, e.g. tilting the user’s head upwards may cause the cursor to move towards the right or downwards. For this reason, the GazePoint method of moving the cursor was utilised. To fix the ‘jumping’ issue, a check was put in to ensure that the new position was more than 50 pixels left/right or up/down from the current cursor position. After this, a preliminary test was undertaken of the eye tracking cursor movement in Rocket League. This failed due to the cursor movements having no effect in the gameplay, despite the cursor being moved by the GazePoint in menus. The gameplay was tested with a regular mouse to see if this was an issue with the game, but the mouse movements worked during gameplay. In the next sprint, this issue will be troubleshooted and an attempt to make the GazePoint more accurate will be made.

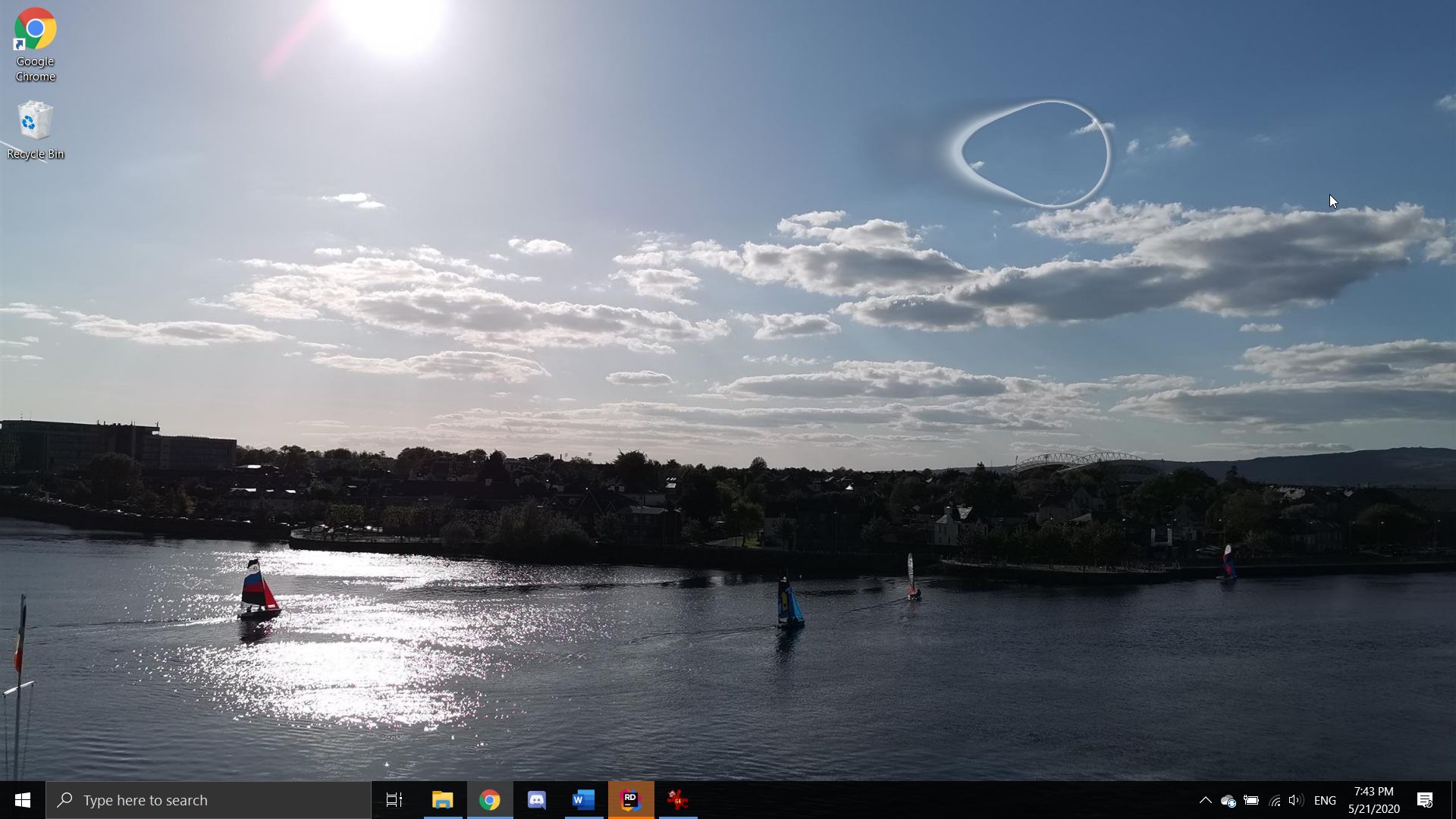


Figure 6.2 Example of difference between GazePoint position from script and actual GazePoint position. Note: Cursor shows GazePoint position recorded by script while circular selection shows actual GazePoint position.

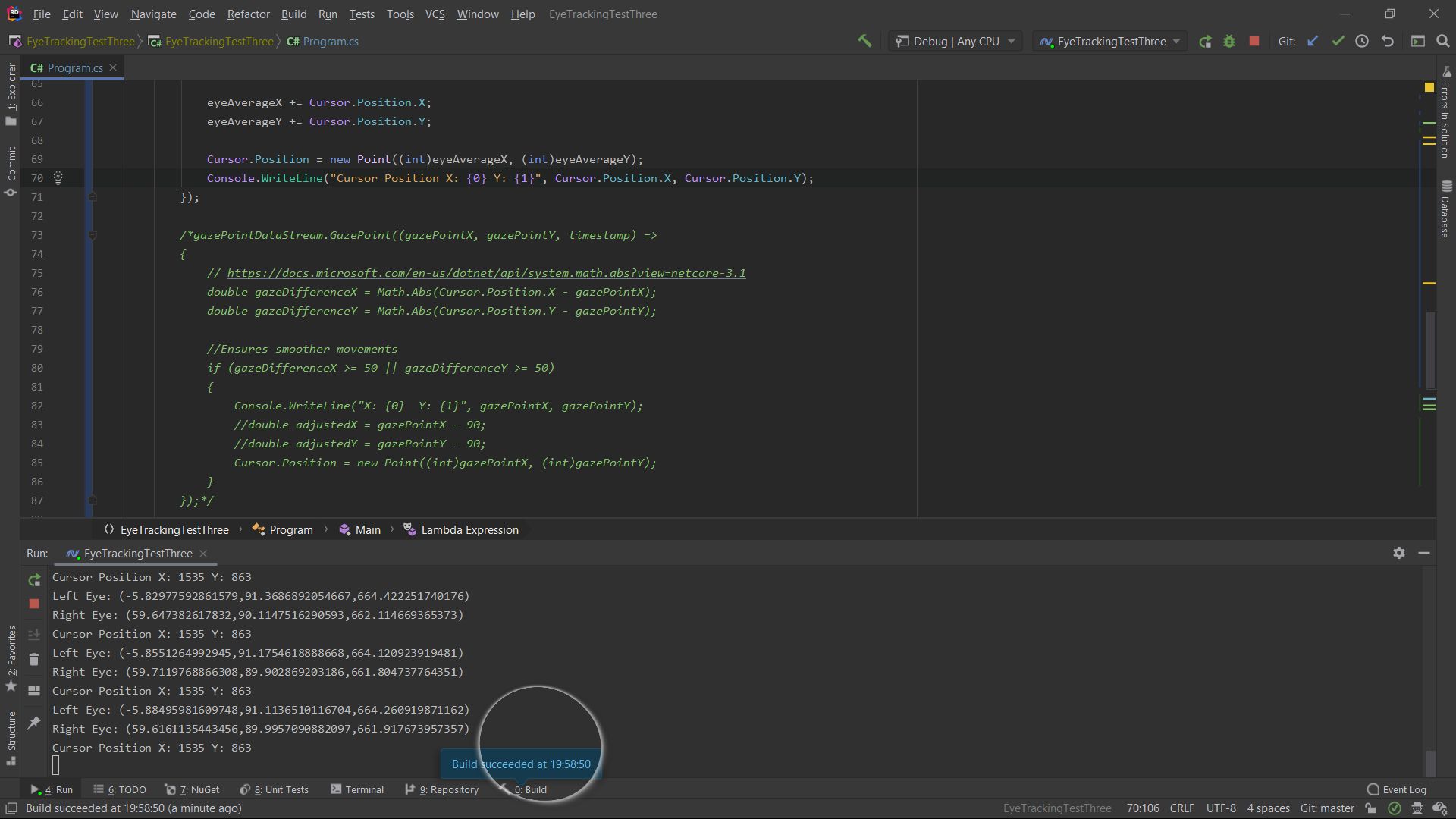


Figure 6.3 Example of problem with use of EyePosition for cursor movement

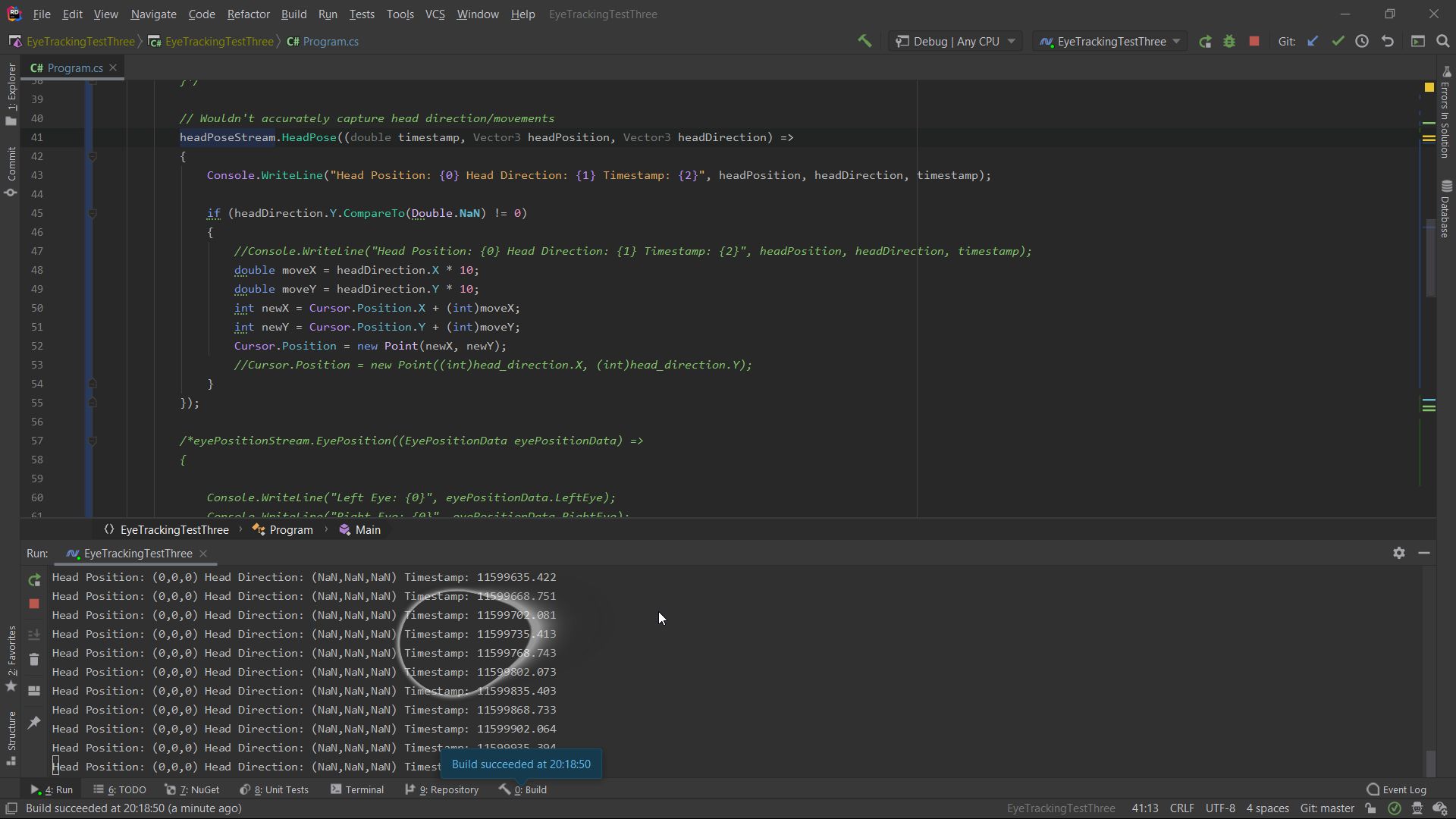


Figure 6.4 Example of HeadPoseStream not recording the user's head pose

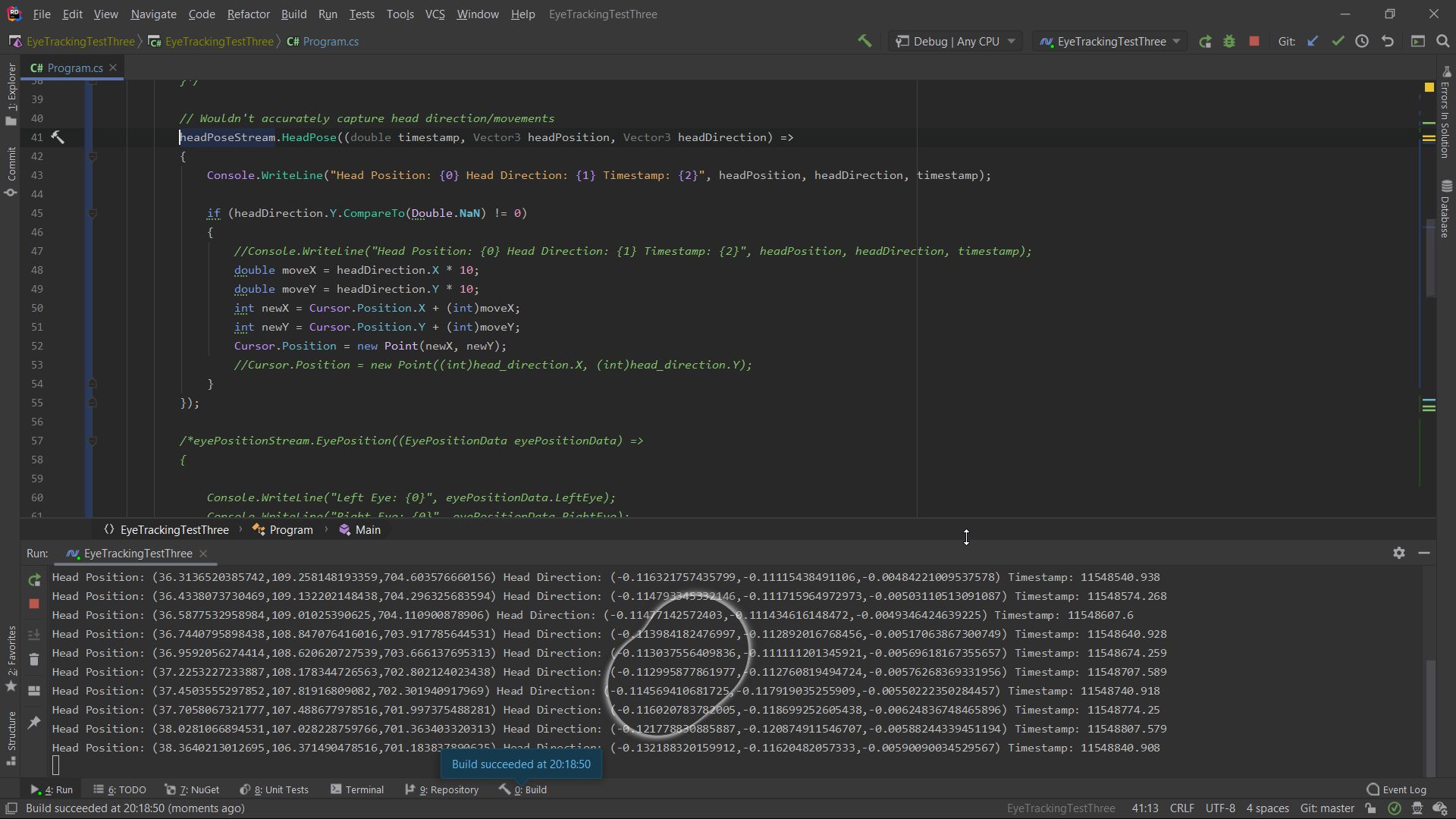


Figure 6.5 Example of the HeadPoseStream after recalibration

## Final Implementation

|  |  |  |
| --- | --- | --- |
| **Sprint Number** | **Start Date** | **Finish Date** |
| 6 | 22/05/2020 | 29/05/2020 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Click Run Tab on Toolbar, Click Edit Configurations, click project name and ensure use external console is checked, Click Apply and Ok | Completed |
| 2 | Modify code to use infinite while loop as shown in Figure 6.6 | Completed |
| 2 | Research alternative to Cursor.Position that can be recognised in game | Completed |
| 3 | Create MouseOperations class to control mouse movements event and populate it with the code in the comment by Keith in the answer (<https://stackoverflow.com/questions/2416748/how-do-you-simulate-mouse-click-in-c>) | Completed |
| 4 | Replace Cursor.Position implementation with MouseOperations.SetCursorPosition | Completed |
| 5 | Revert to the GazeDataStream method of collecting Gaze position | Completed |
| 6 | Add a call to MouseOperations.MouseEvent with the Move flag | Completed with Errors |
| 7 | Implement checks for gaze direction movement and replace Move Flag with LeftDown, LeftUp, RightDown and RightUp flags | Completed with Errors |
| 8 | In the MouseOperations class, add a MouseEvent method that accepts the MouseEventFlags and the x and y values of the mouse as parameters as shown in Figure 6.7 | Completed |
| 9 | Replace MouseOperations.SetCursorPosition and MouseOperations.MouseEvent calls with calls to the new MouseOperations.MouseEvent method | Completed with Errors |
| 10 | Troubleshoot Incorrect movement and disappearing mouse cursor with MouseEvent calls | Completed |
| 11 | Redesign the Main method as shown in Figure 6.8 | Completed |
| 12 | Test new implementation in OS | Completed |
| 13 | Test new implementation in Rocket League | Completed |
| 14 | Test new implementation with VoiceBot in Rocket League | Completed |
| 15 | Test new implementation with Xbox Adaptive Controller in Rocket League | Completed |
| 16 | Test new implementation with Xbox Adaptive Controller and VoiceBot in Rocket League | Completed |

This sprint aimed to complete the implementation as laid out by the design that was outlined in the design phase of the study. To complete this, a number of objectives needed to be completed along with several challenges that had to be overcome. The sprint began with the refactoring of the Main method from a GazeDataStream loop to an infinite while loop with a break condition. This aimed to make the class cleaner and more concise, but in reality, caused inaccuracies with the data returned from the eye tracker. It was decided that the initial method would be used as the accuracy of the data was more important to the study. Following on from this, research was undertaken to replace the Cursor.Position call with an implementation that would be recognised within a video game, such as Rocket League. This research led to the discovery of the mouse\_event function, “which synthesizes mouse motion and button clicks” (Windows Dev Center, 2018). This function was identified alongside a sample class of its functionality, MouseOperations.cs. This class was introduced into the project to replace the Cursor.Position. At first, the SetCursorPosition was used to replace the Cursor.Position, but this incurred the same issue of not being able to recognise this movement in-game. For this reason, the MouseEvent method was investigated. This method took a MouseEventFlags object as a parameter and made a call to the mouse\_event function with the current position of the mouse being sent along with the MouseEventFlags object. At first, this was called following the SetCursorPosition call with the Move MouseEventFlag. However, this caused the cursor to flicker when the program was running. It was then decided to refactor the Main method of the program so that it would make a call to the MouseEvent method depending on the direction the gaze was compared to the cursor, e.g. if the gaze was above and to the left of the cursor, it was call the MouseEvent method with the LeftUp flag. This had the same issue as the Move flag call, but also caused random clicks of the mouse. After further research of these flags, it was found that these flags were not related to movement but to clicks (<https://stackoverflow.com/questions/30145724/c-sharp-mouse-drag-function-using-mouse-event-flags> & <https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-mouse_event#return-value>). However, this led to a new method of achieving successful eye tracker control in-game. The Main method was once again refactored to no longer move the cursor to the gaze position but instead set the cursor to the center of the display and moves up/down and/or left/right depending on the direction the user is gazing. To ensure the user could continually simulate left/right or up/down mouse movement, the cursor was repositioned to the center of the display once it had moved a large enough distance away. This method worked as planned when run on its own. However, due to the previously mentioned inaccuracy between the gaze position recorded and the user’s actual gaze position, a large enough boundary box for the mouse movement was determined to be necessary. As well as this, to reduce the difficulty for the user, a minimum distance between the gaze position and the cursor was used so that the user needed to be looking a distance equal to or greater than this minimum difference for the movements to be made. The program was then tested on its own, in Rocket League and with VoiceBot and the Xbox Adaptive controller. The tests of the program with VoiceBot and the Xbox Adaptive controller were performed first with each of the components separately and then together once they were confirmed to work as desired. Once this implementation was deemed to work to the specification described in the design phase, it was determined that this project was completed successfully.

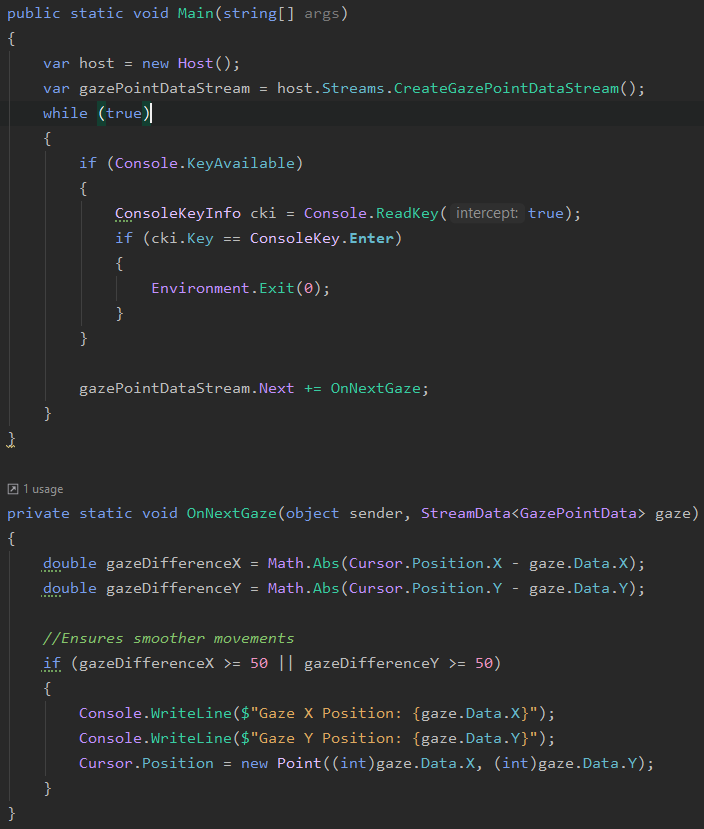


Figure 6.6 Modified Eye Tracking Class

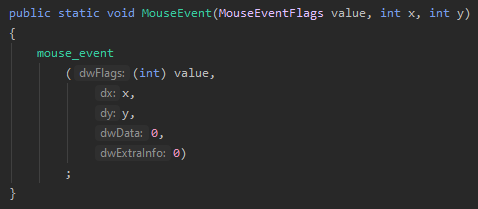


Figure 6.7 MouseEvent method

# Analysis and Conclusions

## Analysis

This study aimed to identify and analyse different accessibility technologies and their effectiveness, with a focus on technologies used within video games. By investigating both the subject of video games and accessibility, alongside their history, the study was able establish an objective view of what each subject was, how they relate to each other and their importance. The final part of this study aimed to implement a controller setup to be used in video games by video game players with reduced motor function abilities. It was found that it was possible to create this controller setup using a number of hardware devices and software applications in conjunction with each other. However, there was a number of obstacles that would prevent this solution from appealing to the general public.

The main issue with this setup is the cost; this setup utilised the Xbox Adaptive Controller, the Tobii Eye Tracker 4C and the VoiceBot software. While the VoiceBot software was free to use, it did have a premium version that would cost between 13.62 and 22.71 Euro depending on how it was to be used. This would be the smallest cost, with the Xbox Adaptive Controller and the Tobii Eye Tracker priced at 89.99 and 169 Euro, respectively. When compared to a traditional Xbox One Controller costing 69.99 Euro, this highlights how expensive this solution would be to a user. These costs fail to take into account the cost for additional triggers and buttons that would make the Xbox Adaptive Controller more adaptable, which would cost upwards of 100 Euro.

The next issue with this setup is the difficulty of setup. Both VoiceBot and the Xbox Adaptive Controller can be used without requiring much calibration. However, the Tobii Eye Tracker 4C required a software application download, calibration for each user and supports a relatively small number of games. Were a project similar to the C# project that was created during this study to be implemented, a user would require sufficient knowledge of computing and coding, along with the ability to troubleshoot any issues that may arise.

The last issue that would prevent this setup being widely adopted by video game players is the performance of the setup. In particular, the Tobii Eye Tracker 4C proved to be a hinderance to gameplay when used in a game that did not natively support it with the C# project created in this study. It is possible that, with modification of in-game settings and further research and refactoring of the C# project, the performance could be improved. However, as it is currently implemented, this setup would not help a user control a game that did not natively support this eye tracker.

## Conclusions and Recommendations

This study has shown a growth in the field of accessibility in video games. There have been many recent attempts at making video games more accessible through software and hardware. These attempts have been more successful than their predecessors as there has been a greater public visibility and adoption of them. However, this study has also highlighted how this accessibility support is not yet accessible to everyone. In particular, hardware-based accessibility supports have a number of barriers for users. This includes the high cost of these devices, the lack of support for these devices in video games, their performance and devices being ineffective without working with other devices.

There are a number of areas into which the field of accessibility in video games could be improved. Currently, accessibility software is much more accessible for users than its hardware counterparts. Bridging that gap will vastly improve the experience of users, as there are certain barriers that software cannot gap. This can be achieved by finding ways to reduce their cost so that they more closely reflect the price of other peripherals. Similarly, devices such as the Xbox Adaptive Controller should be paired with triggers and buttons within the box as opposed to requiring users to buy them separately. This will serve to make these devices more accessible for lower income video game players. Lastly, the support for these devices in video games needs to be universal. To realistically achieve this, these devices need to be paired with proprietary software that translate between the device input and the game.

In conclusion, while there are many barriers to accessibility in video games, the field is one that is growing and improving with each new hardware peripheral and software application that aim to improve accessibility for everybody.

# References

[Tobii], G., 2018. *[Solved] No Gaze data received?.* [Online]   
Available at: https://developer.tobii.com/community/forums/topic/no-gaze-data-received/  
[Accessed 12 March 2020].

Bend Studio, 2019. *Days Gone Patch 1.30 Notes Discussion.* [Online]   
Available at: https://www.reddit.com/r/DaysGone/comments/cikfip/days\_gone\_patch\_130\_notes\_discussion/  
[Accessed 15 May 2020].

Bierre, K. et al., 2019. *Game Not Over: Accessibility Issues in Video Games.* 1st ed. San Francisco: Games Accessibility Special Interest Group.

Bishop, M., 2018. *Brief History of Disability and Web Accessibility in the United States.* [Online]   
Available at: https://www.ebsco.com/blog/article/brief-history-of-disability-and-web-accessibility-in-the-united-states  
[Accessed 20 November 2019].

DougDoug, 2020. *Can you bike down GTA's Mountain, using ONLY your voice?.* [Online]   
Available at: https://www.youtube.com/watch?v=UNSPeyf\_DN0  
[Accessed 19 March 2020].

Erven, L., 2020. *Taking a look at my Grand theft Auto V Addon.* [Online]   
Available at: https://www.youtube.com/watch?v=CU05lQQx784  
[Accessed 30 May 2020].

Grubb, J., 2014. *AbleGamers: Final Fantasy XIV was the most-accessible mainstream game of 2013.* [Online]   
Available at: https://venturebeat.com/2014/01/28/final-fantasy-xiv-a-realm-reborn-was-the-most-accessible-mainstream-game-of-2013-according-to-ablegamers/  
[Accessed 30 May 2020].

Hoffman, B., 1950. *Danny Kaye beats Bertie the Brain at Tic-Tac-Toe.* Toronto: Life Magazine.

Microsoft Accessibility Blog, 2017. *From Hack to Product, Microsoft Empowers People with Eye Control for Windows 10.* [Online]   
Available at: https://blogs.microsoft.com/accessibility/from-hack-to-product-microsoft-empowers-people-with-eye-control-for-windows-10/  
[Accessed 14 May 2020].

National Disability Authority, 2017. *EU Web Accessibility Directive.* [Online]   
Available at: http://nda.ie/Publications/Communications/EU-Web-Accessibility-Directive/  
[Accessed 4 December 2019].

Nitray, K. J., 2015. *Tennis for Two.* [Online]   
Available at: https://sexedi.ga/09174746.pdf  
[Accessed 4 December 2019].

Nitsche, M., 2008. *Video Game Spaces : Image, Play, and Structure in 3D Worlds.* 1st ed. Cambridge, Mass: The MIT Press.

Oxford University Press, 2019. *Definition of accessibility noun from the Oxford Advanced Learner's Dictionary.* [Online]   
Available at: https://www.oxfordlearnersdictionaries.com/definition/english/accessibility?q=accessibility  
[Accessed 20 November 2019].

Perez, D., 2019. *Logitech G Adaptive Gaming Kit Creates New Possibilities for Gamers with Accessibility Needs.* [Online]   
Available at: https://www.businesswire.com/news/home/20191118005308/en/Logitech-Adaptive-Gaming-Kit-Creates-New-Possibilities  
[Accessed 14 May 2020].

Price, J., Reid, P., Hillier, M. & (Padraig), C., 2020. *FFXIV Census.* [Online]   
Available at: https://ffxivcensus.com/  
[Accessed 30 May 2020].

sajidbaloch, 2018. *Tobii-Eye-Tracker-4C.* [Online]   
Available at: https://github.com/sajidbaloch/Tobii-Eye-Tracker-4C  
[Accessed 09 March 2020].

Simmons, M., 1975. *Bertie the Brain programmer heads science council.* Ottawa: The Ottawa Citizen.

Smith, A., 2014. *Tennis Anyone?.* [Online]   
Available at: https://videogamehistorian.wordpress.com/2014/01/28/tennis-anyone/  
[Accessed 4 December 2019].

Smith, A., 2014. *The Priesthood at Play: Computer Games in the 1950s.* [Online]   
Available at: https://videogamehistorian.wordpress.com/2014/01/22/the-priesthood-at-play-computer-games-in-the-1950s/  
[Accessed 3 December 2019].

Square Enix, 2018. *FFXIV Patch 4.3 Under the Moonlight Patch 4.3 Notes.* [Online]   
Available at: https://na.finalfantasyxiv.com/lodestone/topics/detail/13e322580acb8e9861160a6e08ccabfaff09eeee  
[Accessed 30 May 2020].

The European Parliament and the Council of the European Union, 2016. *Directive (EU) 2016/2102 of the European Parliament and of the Council,* Strasbourg: Official Journal of the European Union.

Tobii Tech, 2016. *Tobii Releases Next Generation Gaming Eye Tracker.* [Online]   
Available at: https://www.tobii.com/group/news-media/press-releases/2016/10/tobii-releases-next-generation-gaming-eye-tracker/  
[Accessed 14 May 2020].

Wilds, S., 2020. *For All The Players: A History Of Accessibility In Video Games.* [Online]   
Available at: https://www.ladbible.com/technology/gaming-for-all-the-players-a-history-of-accessibility-in-video-games-20200124  
[Accessed 29 January 2020].

Windows Dev Center, 2018. *mouse\_event function.* [Online]   
Available at: https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-mouse\_event#return-value  
[Accessed 24 May 2020].

Wolf, M. J. P., 2007. *The Video Game Explosion: A History from PONG to Playstation and Beyond.* illustrated ed. Westport, Connecticut: ABC-CLIO, 2008.

# Table of Figures

[Figure 3.1 Life magazine photo of Danny Kaye in front of Bertie the Brain at the Canadian National Exhibit in 1950 (Hoffman, 1950) 6](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/FYP/Michael_Edgar_T00194492_FYP.docx#_Toc41825601)

[Figure 3.2: Pong Game 6](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/FYP/Michael_Edgar_T00194492_FYP.docx#_Toc41825602)

[Figure 4.1 Final Fantasy XIV Color Filter Settings (Square Enix, 2018) 11](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/FYP/Michael_Edgar_T00194492_FYP.docx#_Toc41825603)

[Figure 5.1 EyeTracking Class 19](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/FYP/Michael_Edgar_T00194492_FYP.docx#_Toc41825604)

[Figure 5.2 PlayerController Class 19](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/FYP/Michael_Edgar_T00194492_FYP.docx#_Toc41825605)

[Figure 5.3 AsteroidController Class 20](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/FYP/Michael_Edgar_T00194492_FYP.docx#_Toc41825606)

[Figure 5.4 GameManager Class 21](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/FYP/Michael_Edgar_T00194492_FYP.docx#_Toc41825607)

[Figure 5.5 Asteroid Prefab Properties 22](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/FYP/Michael_Edgar_T00194492_FYP.docx#_Toc41825608)

[Figure 5.6 Player Prefab Properties 22](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/FYP/Michael_Edgar_T00194492_FYP.docx#_Toc41825609)

[Figure 6.1 Tobii Eye Tracker Gaze Data Coordinates 26](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/FYP/Michael_Edgar_T00194492_FYP.docx#_Toc41825610)

[Figure 6.2 Example of difference between GazePoint position from script and actual GazePoint position. Note: Cursor shows GazePoint position recorded by script while circular selection shows actual GazePoint position. 30](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/FYP/Michael_Edgar_T00194492_FYP.docx#_Toc41825611)

[Figure 6.3 Example of problem with use of EyePosition for cursor movement 30](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/FYP/Michael_Edgar_T00194492_FYP.docx#_Toc41825612)

[Figure 6.4 Example of HeadPoseStream not recording the user's head pose 30](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/FYP/Michael_Edgar_T00194492_FYP.docx#_Toc41825613)

[Figure 6.5 Example of the HeadPoseStream after recalibration 31](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/FYP/Michael_Edgar_T00194492_FYP.docx#_Toc41825614)

[Figure 6.6 Modified Eye Tracking Class 35](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/FYP/Michael_Edgar_T00194492_FYP.docx#_Toc41825615)

[Figure 6.7 MouseEvent method 35](https://d.docs.live.net/7dbab720b972a6b6/Documents/GitHub/FYP/Michael_Edgar_T00194492_FYP.docx#_Toc41825616)