Project to Explore the Use of Gaze Tracking in Competitive First-Person Shooters

COMP 8037 – Major Project Proposal

BW – A00000000
11-23-2020

Table of Contents

1. Student Background 2
   1.1. Education 2
   1.2. Work Experience 2
2. Project Description 2
3. Problem Statement and Background 2
4. Scope and Depth 3
5. Test Plan 4
6. Methodology 5
7. System/Software Architecture Diagram 6
8. Innovation 7
9. Complexity 9
10. Technical Challenges 10
11. Development Schedule and Milestones 10
12. Deliverables 12
13. Conclusion and Expertise Development 13
14. References 13
15. Change Log 14

Commented [DV1]: Please remember to update the table prior to saving and printing your document for review.

1. Student Background

BW is a student with a strong passion for video games, especially exploratory games such as traditional role-playing games and massive-multiplayer games, and competitive first-person shooters. He is also curious about advancements in computer hardware technology and how they affect games development.

1.1. Education

British Columbia Institute of Technology

- Computer Systems Diploma – Digital Processing
  - Graduated May 2017 with Distinction
- Bachelor of Technology – Games Development
  - September 2019 - Current

1.2. Work Experience

Company 1

- Worked on Reactor, a web browser-based game of brain training minigames.

Company 2

- Worked on TalkBox Assistant, a browser-based virtual chat assistant for e-commerce websites.

Company 3

- Worked on Ambit, an iOS board game

2. Project Description

The goal project is to use a gaze tracking device such as Gazepoint to provide input and assist in aiming and improve reaction time inside a competitive first-person shooter. The gaze tracker will use eye tracking data to simulate mouse movement. The aim is to demonstrate that the use of a gaze tracker in combination with traditional input devices can improve a player's performance inside a competitive shooter in a measurable way without requiring the game to guess and "lock on" a target. In this way, the physical skill gap that exists between players in competitive shooters can be lowered, allowing more inclusivity in such games while maintaining the competitive integrity of the game by not having the software guide the player to a target.

3. Problem Statement and Background

Traditionally, there has been a very large skill gap in competitive first-person shooting games. These games are defined as shooters where:

- the enemy player or non-player character hitboxes are divided up into sections worth different values

- weapons in the game have recoil when firing
- time-to-kill (the required damage to kill an enemy) is very small

These games favor people with low reaction times, enhanced peripheral vision, excellent hand-eye coordination, and fine-tuned motor skills. There are many reasons for people to have lower performance than others in these games, such as:

- a player being too young to be able to display fine-tuned control with a keyboard and mouse
- a player being older and having declining motor skills or concentration levels
- a player not having the capability for fine-tuned motor skills due to injury or illness
- the inability for fine-tuned accuracy using input such as a controller

This skill gap creates a problem of toxicity in the gaming communities of competitive first-person shooters due to the existing skill gap between players on the same team, which can result in further decline in performance. The toxicity can make players feel demoralized and lose interest in these games, creating an issue of exclusivity.

Attempts at using a simple webcam or headgear to track eye movement have been made, but there are fundamental flaws with these approaches.

Webcams:

- are too low resolution to capture the details of eye movement. They can track head movement at best and reduce the accuracy of gaze detection to quadrants of a screen
- don't work well in environments with poor lighting, which causes detail of the image to be missed
- have frame rates too low to provide fast enough communication on change of direction with the eyes
- have problems working on gaze detection for people with glasses, depending on lighting

Headgear:

- is heavy to wear, which limits play time before it gets too uncomfortable to use
- is incompatible for people who wear glasses

Many games also allow for the use of aim-assist for consoles due to the difficulty of fine-tuned aiming using a controller. This is a software implementation of assisted aiming for high value sections of the hitbox when a player can position the crosshair close enough to the head using a controller. Such an implementation would not be used in a competitive first-person shooter because it ruins the integrity of the game by reducing the impact of player skill on a successful kill.

4. Scope and Depth

The scope of this project is to develop a game that simulates a competitive first-person-shooter that integrates the usage of a gaze detection device to enhance traditional controls such as a keyboard-and-mouse combo, or a controller. Extra functionality, time permitting, would be a Windows 10 program that allows the use of the gaze tracker to control the mouse in any game without in-game support with

the same fine-tuned controls as the simulation game. For depth, the game will feature multiple levels for variety, AI opponents for the player, benchmarking functions to gather statistics on player performance, a GUI overlay to provide a heatmap for player vision, and different options for gaze tracking integration such as a toggle option for reduced cursor movement and toggle of a vision focusing mode that will slow down cursor movement for accuracy.

The user control adjustment features will show how players can use gaze tracking as a control mechanism in competitive games to increase the skill floor of the players, and the GUI overlay and benchmarking tools will assist in overcoming the technical challenges in translating the user control features into a more integrated experience for the user.

Outside of scope is developing a multiplayer mode for the player to play against other people, or networking capability.

The feasibility of the project should be high. There is an API to help interpret the data recorded by the Gazepoint tracking device, and software to assist in user calibration. The lack of multiplayer for this game reduces the length of the game development to an obtainable goal for a timeframe of 360 hours, including research and testing. The Gazepoint device provides multiple data points for analysis, not just direction of the user's gaze, so there should be enough data to differentiate when a player is focused on a single point and when they are scanning the area. There should also be increased accuracy over other products such as the Tobii eye tracker due to the increased number of sensors.

5. Test Plan

Unit testing will be used to verify recoil physics, Gazepoint data interpretation, and any helper functions. For example, recoil physics testing will test the vector of the projectile's travel path in relation to the direction of the player. A pass/fail would be whether the degree of difference between the projectile's travel path and the player's travel path differs by more than the recoil's set degree of variance. A pass/fail for the Gazepoint data would be to compare the values parsed by the API developed to interpret the Gazepoint data with a saved XML file of the same Gazepoint data. A fail would be the result of different values between the API and the XML file.

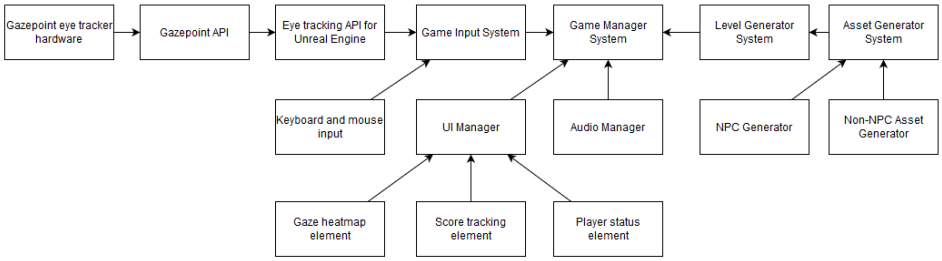
Testing for the rendering performance of the game would be a measure of the frame rates. The minimum frame rate for the game rendered at 1080P resolution is 100 FPS, so anything less than that would be a fail. This test will be measured by a simple counter that measures the elapsed time between system updates.

Testing for performance using gaze tracking technology in the game will be measured using benchmark levels or mini games. These games will have multiple runs with both gaze tracking activated and without to measure the difference in performance. This difference will be measured as the time taken to pass a particular obstacle or goal. An acceptable variance in performance is if the Gazepoint performs with a lower time than without gaze tracking, or if it performs within 5% if greater than the time taken without using gaze tracking.

6. Methodology

The framework used for this game will be Unreal Engine 4.25. It is the latest available version of the engine to the public. Due to this, the programming language used for the project will be C++17. The plugin to use the Gazepoint data in Unreal will also be programmed in C++. The development platform for the project will be Microsoft Windows 10, due to it being the most popular PC gaming platform. The technology being used for gaze tracking will be the Gazepoint eye-tracking system. The development process used will be Agile and the development schedule and milestones will be planned using a work breakdown structure. The first milestone will focus on getting a game without gaze tracking implemented, while the second milestone will focus entirely on gaze tracking integration. The final milestone will involve extensive playtesting and revisions to the controls based on tester feedback.

For the assets in the game, all assets will inherit from the Object class. As an example, the Person class is inherited from the Object class, and the Player and NPCs are considered types of people. The player can use any choice of weapons, while the two different types of NPCs will be separated into Close Range and Long Range NPCs, each using either shorter range weapons or longer range weapons, and preferring to either engage closer to the player or prefer to stay at a distance.



The above figure shows the design of the game systems. Interfacing of the Gazepoint eye tracker will be done through an API with Unreal that will communicate with the Gazepoint API to translate eye tracking data. This API will send the data to the game input system, which will configure player control along with keyboard and mouse inputs. This feedback will be sent to the game manager, which also interfaces with the level generation system, audio system, and UI system. The level generator will deal with random level generation, which will randomize locations of NPCs and obstacles, which will be generated by the asset manager. The UI manage will provide visual feedback, including testing features such as heatmap visualization of player gaze focus, and score tracking for benchmarking purposes.

7. System/Software Architecture Diagram

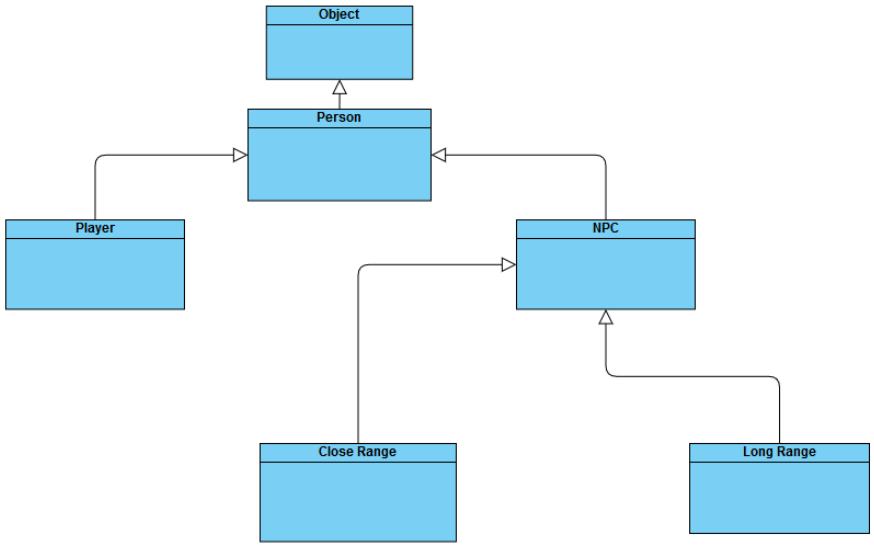


Figure 1: Class Inheritance for NPC and Player classes

Figure 1 shows the class inheritance for the player and NPC classes. Each class will have the same components such as health, movement speed, and the ability to carry weapons. The NPC classes will be separated into NPCs that prefer close range or long rage engagement, and their selection of weapons will reflect this.

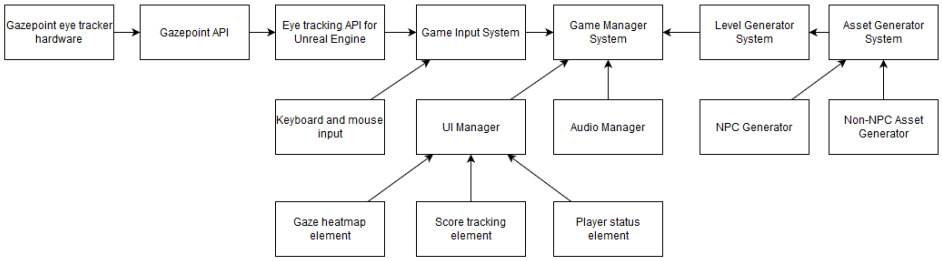


Figure 2: A system diagram of all the different systems in the game

Figure 2 shows the different systems that this game will have. Most of the management functionality will be split between the game input system, level generator system, and UI manager system, with the game manager system being responsible for cross communication between the different systems.

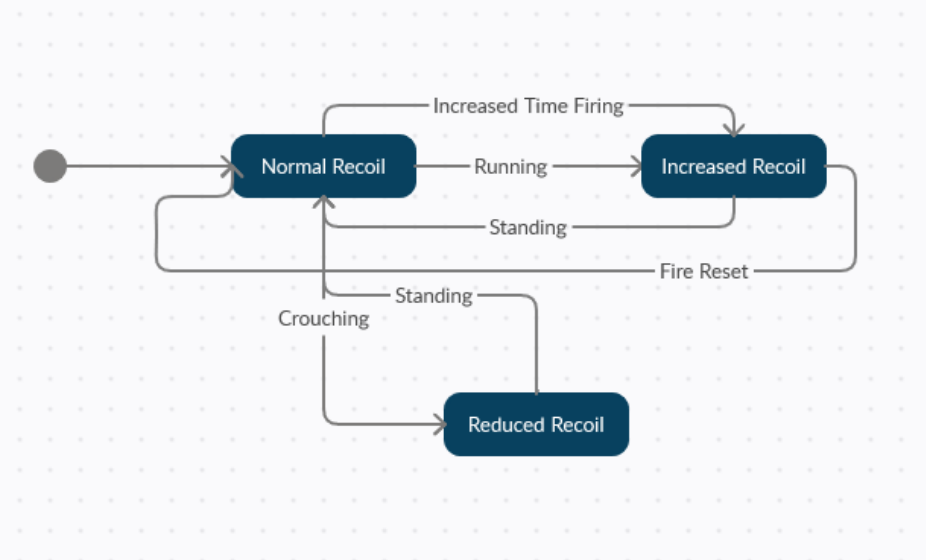


Figure 3: Player/NPC State Machine

Figure 3 shows the change in recoil physics for the player and NPCs in the game. There will be a starting recoil point that decreases the degree of recoil if the player is crouching and increases if the player is running, taking damage, and as the player continues to fire their weapon without pausing to reset the recoil. This recoil is reset using a decay by standing, slowing down the rate of fire, and by not taking damage.

8. Innovation

The innovate part of this project is exploring how to use gaze tracking input to allow for an increase in performance in first-person shooters. Gaze trackers have been used in games before, but not without some software assistance to make them more accurate or to be used for general camera movement and menu selection rather than aiming.

Some game developers have added gaze tracker input into their games using the Tobii eye tracker, but none of the games are competitive first-person shooters and they don't allow for fine-tuned aiming using the eye tracker.

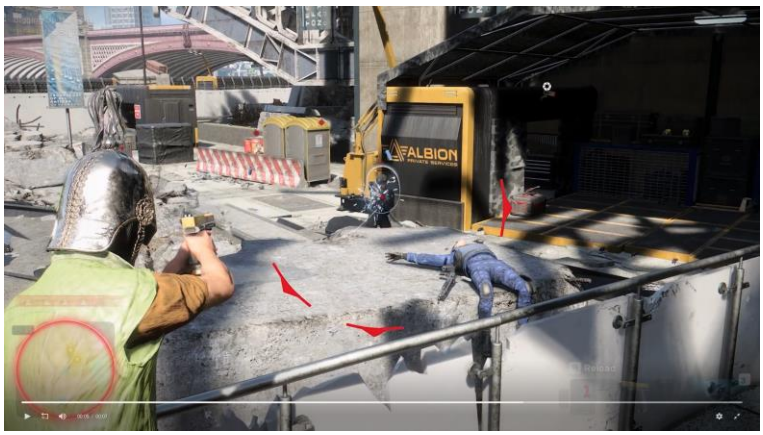


Figure 4: [Tobii eye tracker used in Watchdogs Legion]. (n.d.). Retrieved November 23, 2020, from https://gaming.tobii.com/games/watch-dogs-legion/

In the figure above, the pale white circle represents the player's focus on the screen. Due to the inaccuracy of the eye tracker, the player needs to aim for the body because they are under heavy fire and it would take too long to carefully move the red crosshair up to the head. This makes Tobii unusable in competitive first-person shooters, many of which have a time-to-kill of a couple seconds or less.

Figure 5: Tobii Eye Tracking Demonstration in The Division 2 - The Gooboberti Way(2019)

The figure above shows the use of the Tobii eye tracker in the game The Division 2. Again, the player needs to aim at the body because it would take too long to aim at the head. This game mainly uses the Tobii eye tracker to scan the surroundings, to access UI elements, and to place projectiles such as grenades on the ground. Both games are not competitive first-person shooters. The lack of accuracy with the Tobii eye tracker makes it impossible to be used in a competitive first-person shooter over a traditional mouse and keyboard only approach because the reaction time would not be sufficient.

The goal is to use the data from a gaze tracker to transform the gameplay into a better experience by using gaze tracking to track where the player is looking on the screen and eye tracking to provide additional features, such as sensing when a person has relaxed or tensed to increase or decrease cursor movement speed. Also, to build the platform to not require development support from individual game titles would be innovative.

9. Complexity

The complexity in this project lies in translating the data from the gaze tracker into meaningful gameplay improvements inside a competitive first-person shooter. Very fine accuracy is required in order to succeed in these games, so if the gaze tracking device isn't able to detect that level of accuracy, other data from the eye needs to be used to change the rate of mouse movement based on whether the player wants to quickly scan the screen or to focus in on a target. All this needs to be integrated seamlessly to result in a measurably better performance than simply relying on a mouse and keyboard.

To make the integration seamless is a very complicated task. There will be difficulty in allowing a player to scan the screen without automatically moving the crosshair as well. This functionality is required because a player needs to know if another enemy is about to engage as he is fighting someone else or because he is preparing to locate the next target as the first one is about to die. This will require the gaze tracker to be able to differentiate between a player focusing on a particularly fine location on the screen and doing a quick scan on the rest of the field of view. This involves tracking more than just the direction of a player's gaze, since other reactions in the eye are going to be needed to indicate how the player is focusing.

Another complicated task is to implement weapon recoil physics for each weapon in the game. I would need to consider the rate of fire, the damage output, the type of weapon, and whether the player is stationary, crouching, or running when implementing the recoil inaccuracy. All these points need to be considered in the implementation process to ensure that the guns feel balanced and somewhat realistic.

This will be a challenge for someone with a programming background because a lot of research into the reactions and signals of the human eye while playing games is going to be needed. It will be difficult for a person with a diploma to attempt this project due to the requirement of being able to implement complex pathing algorithms and decision trees for non-player character artificial intelligence. It will also require enough programming experience to understand how to use the TCP/IP protocols to communicate data and how to integrate that into a game engine. It will also require experience with using game engines to be able to complete this project in the given time frame.

The major reason this idea hasn't been done before is due to the inaccuracy and lack of responsiveness with existing products in the gaming space. The Tobii eye tracker is inaccurate due to the lack of sensors on the device, a decision that was made to do two things: lower the cost of entry and make it portable enough to apply as a sticker on the bezel of a laptop or monitor. However, I believe that with the growing impact of toxicity in this genre of games, more people would be willing to invest in more expensive equipment to have a much more satisfying experience.

10. Technical Challenges

Eye Tracking Data Translation

It'll be my first time using gaze tracking and eye tracking data, so using the Gazepoint API could be challenging. I would need to figure out how to allow my game to communicate with the Gazepoint device through its API, which uses the TCP/IP protocol. First, I would need to figure out how to be able to get the game engine to communicate with a non-traditional USB device. Then, I would have to create an API in Unreal Engine in order to interpret the data given by the Gazepoint device into meaningful directions in my game.

Unreal Engine Development and General Development on Windows

It will be my first time developing using Unreal Engine, which is considerably different than Unity. It is C++ based instead of C#, and it allows more freedom in how to implement the components of your game than Unity. However, this creates a challenge because more freedom also means more opportunity for mistakes and less of the engine is already set up for you. Also, integrating addon packages in Unreal Engine is more difficult than Unity as there is more setup required by the user in order to integrate the libraries.

11. Development Schedule and Milestones

Task	Time Required(h)/(Total)	Start Period
Phase 1: Basic Rendering/Level Generation		
Simple Rendered Map (basic shapes)	12 (12)	Week 1
Random Generation of Map Walls/Obstructions	15 (27)	Week 1
Random Generation of Spawn Points Based on Layout	12 (39)	Week 2
Loading of Assets From File	3 (42)	Week 2
Phase 2: Asset Generator System		
NPC Generator System Skeleton	3 (45)	Week 2
Player Class, NPC Class and Subclasses	3 (48)	Week 2
Weapons Classes	3 (52)	Week 2
Integrate Player/NPC/Weapons Classes into NPC Generator	3 (55)	Week 3

Non-NPC Generator (Walls/Obstructions) and Classes	6 (61)	Week 3
Phase 3: Basic Game With Single Random Generated Level		
Keyboard and Mouse Input Implemented	2 (63)	Week 2
Player Movement and Projectiles Firing	6 (69)	Week 2
Enemy Movement AI	6 (75)	Week 3
Enemy Targeting AI and Projectiles Firing	6 (81)	Week 3
Damage and Health System for Players and NPCs	2 (83)	Week 3
Collision Detection of Projectiles	2 (85)	Week 3
Player Testing and Feedback Adjusting	21 (106)	Week 4
Phase 4: UI Manager		
Implement Menus	15 (121)	Week 4
Implement Game UI with Health Bars	9 (130)	Week 4
Implement Directional Damage UI/Crosshair	6 (136)	Week 5
Implement Placeholder for Gaze Tracker Zone	6 (142)	Week 5
Phase 5: Audio Manager		
Implement Player/Weapon Sound Effects	10 (152)	Week 5
Implement Directional Sound	5 (157)	Week 6
Implement Ambient Sounds/Music	6 (163)	Week 6
Phase 6: Weapon Types and Recoil		
Implement Weapon Types with Fire Rates and Damage	5 (168)	Week 6
Implement Recoil Physics	16 (184)	Week 6
Phase 7: Game Manager Integration		
Implement Game Manager Integration with All Subsystems	15 (199)	Week 7
Milestone 1: Basic Game with Single Level		**Week 7**
Phase 8: Gaze Tracking Implementation		
Study Gazepoint API and TCP/IP Protocols	15 (214)	Week 7

Create Parser for Gazepoint Data (API for Unreal Engine data conversion)	30 (244)	Week 8
Create C++ API for Gazepoint and Unreal Engine Communication	21 (265)	Week 9
Add Unreal to Gazepoint API Call Functionality	27 (292)	Week 10
Add Gazepoint input to Game with Screen Cursor	9 (301)	Week 11
Gaze Tracking Player Testing and Control Fine Tuning	21 (322)	Week 12
Milestone 2: Gazepoint Integration		**Week 12**
Phase 9: Benchmarking Levels		
Single Static Benchmarking Level	18 (340)	Week 12
Simple Benchmarking Mini-Game	3 (343)	Week 13
Final Player Testing and Adjustments	10 (353)	Week 13
Phase 10: Wrap Up and Report		
Gather Data and Write Report and Analysis	12 (365)	Week 14
Milestone 3: Final Deliverables		**Week 14**
Extra Time For Report Changes	TBD	Week 15 (If Necessary)

12. Deliverables

Deliverables:

- Fully working game with one level that has a randomly generated layout, random NPCs, and random assets
- One benchmarking level that has a fixed layout, fixed NPCs, and fixed assets and asset locations. This benchmark will be run once with gaze tracking enabled and once without and the time difference will be displayed
- One simple benchmarking mini-game that involves clicking randomly generated shapes on the screen. This benchmark will be run once with gaze tracking enabled and once without and the time difference will be displayed

- One user manual detailing the controls and adjustment options for the game, the weapon statistics (including fall off range, damage, fire rate, recoil strength), as well as the different game modes
- A report detailing the project development process, and research results and analysis

13. Conclusion and Expertise Development

This project will further my experience in my specialization in two main ways: experience using Unreal Engine and experience using non-traditional hardware in games.

The experience gained from being able to develop using Unreal Engine will be of huge value because Unreal is the most popular game engine used for console and PC development. It also takes a more free and less handholding approach than Unity, which will allow me to face new challenges as I adjust to figuring out a new engine that is more prone to programmer error. Unity handholds developers by creating an ecosystem in which mistakes are easily caught and libraries and addons are easier to integrate. However, they do this by limiting access and functionality that Unreal does not in order to make sure that nothing game breaking happens that isn't easily traceable.

Being able to develop with an external, non-traditional input device like the Gazepoint gaze tracker will be extremely valuable because of the shift to virtual reality and augmented reality game development requires experience using eye-tracking and sensory tracking hardware. The experience developing an API to communicate between the game engine and the hardware API is also valuable.

14. References

- OPEN GAZE API BY GAZEPOINT. (n.d.). Retrieved October 16, 2020, from https://www.gazept.com/dl/Gazepoint_API_v2.0.pdf
- R. Netzel and D. Weiskopf, "Hilbert attention maps for visualizing spatiotemporal gaze data," *2016 IEEE Second Workshop on Eye Tracking and Visualization (ETVIS)*, Baltimore, MD, 2016, pp. 21-25, doi: 10.1109/ETVIS.2016.7851160.
- R. Mallick, D. Slayback, J. Touryan, A. J. Ries and B. J. Lance, "The use of eye metrics to index cognitive workload in video games," *2016 IEEE Second Workshop on Eye Tracking and Visualization (ETVIS)*, Baltimore, MD, 2016, pp. 60-64, doi: 10.1109/ETVIS.2016.7851168
- Eye Trackers for PC Games. (2020, November 20). Retrieved November 23, 2020, from https://gaming.tobii.com/products/
- G. (2019, March 23). Tobii Eye Tracking Demonstration in The Division 2 - The Gooboberti Way [Digital image]. Retrieved November 23, 2020, from https://www.youtube.com/watch?v=Nn9wzUNsKHc

15. Change Log

- November 23, 2020: Version 1.0