UNIVERSITY OF HERTFORDSHIRE

Department of Computer Science

Modular BSc Honours in Computer Science

6COM2021 - Software Engineering Project

Final Report

March 2024

Development of a prototype FPS game for the comparison of aim assistance algorithms.

McGrath-Johnston, C.

Supervised by: Zheng, Y.

Abstract

The use of aim assistance in video games is a constant area of debate in gaming circles and media.

This report documents the development of an FPS video game for PC, with the implementation of several different aim assist algorithms.

Analysis of existing research, a developing knowledge of games creation, and analysis of different algorithms are used to support the best overall aim assistance technology for games development.

This research shows a clear benefit from aim assistance, with differing results and ideal algorithms for different gameplay scenarios. Bullet Spread, Area Cursor, and Bullet Magnetism were all tested at different strengths and analysed from an expert's/designer's perspective, and a player's perspective.

Overall, it was found that the best aim assistance algorithm is Area Cursor. It is the most versatile and effective at enhancing player experience. However, there is still a need for different algorithms such as Bullet Spread and Bullet Magnetism for more specific situations.

Acknowledgements

I would like to thank my supervisor, Dr. Yongjun Zheng for providing support and guidance throughout the project, and for engaging in regular meetings to keep the project on track.

I would also like to extend my appreciation to Dr. Christoph Salge for giving initial support and guidance with generating project ideas.

Contents

Figures

Glossary

Accessibility requirements – *needs of users, both in general and of those who may be handicapped.*

Aim assist[ance] – *a method implemented with the goal of making targeting easier in software.*

Algorithm – *a program implemented for a specific purpose.*

Area Cursor – *a standard aim assistance technique by which the player reticle is expanded larger than is visible.*

Artefact – *a product developed: in this project an artefact is a software algorithm.*

Bloom – *a standard aim adjustment algorithm that punishes firing too quickly by randomizing bullet direction slightly.*

Bullet Magnetism – *a standard aim assistance technique by which bullets are attracted slightly towards nearby players.*

Bullet Spread – *a standard aim adjustment technique by which bullet direction is randomized, used for both aim assistance and punishment.*

Client-client data transfer – *the process by which data is sent between two nodes without an intermediary server.*

Client-server data transfer – *the process by which data is sent to/from an external server.*

Cone – *an imagined shape produced by bullet hitboxes changing size at a constant rate while travelling.*

Connection-based matchmaking (CBMM) – *a process by which players are selected to join a PvP game based on the quality of their individual data transfer across the network.*

Delay – *time taken to perform an action required.*

Development environment – *software or location used for programming.*

First Person Shooter (FPS) – *a video game genre where players have ranged weapons and see the game world through a character's eyes.*

Fitts' Law – *a rule that describes the relationship between time taken to aim, and the target size and distance.*

Flow state – *when a player becomes immersed in a game to the point where their actions feel fluid and almost subconscious.*

Game engine – *a software application used to create, compile and run a video game.*

Game Experience Questionnaire (GEQ) – *a form used to quantify subjective opinions on gameplay from a player.*

Ghost bullet – *an aim adjustment technique by which shots at random will not fire when the trigger is pulled.*

Hitbox – *an invisible interactable area attached to a visible object used for physics collisions.*

Hitscan – *a method for instant detection of bullet registration.*

Hit registration – *a method used to detect whether a bullet makes contact with an object.*

Input device latency – *the time taken between a signal sent by external hardware and the corresponding action on screen.*

Leading – *a process by which players aim away from a target which is moving or far away to account for travel time and bullets dropping due to gravity.*

Local latency – *the time taken between a player doing something in the real world and the corresponding in-game action due to their own device.*

Magic bullet – *a situation where a bullet has increased accuracy or damage than normal, usually due to bugs in a game or its network.*

[Programming] Method – *a block of code that performs a specific task: a function or procedure.*

Netcode – *a method by which network connections in a game are manipulated by the game to create fairness.*

Object-oriented programming (OOP) – *a coding paradigm by which related data and methods are encapsulated in object structures.*

Open-source – *software libraries available to the public for easier software development*

Output device latency – *the time taken to display something on screen that has happened in the game.*

Pan-based aim controls – *a process by which a cursor or crosshair is moved by rotating the player's camera.*

Pickup – *an in-game object that can be collected and added to the player's inventory.*

Reticle – *the dot or crosshair on the centre of the screen indicating where bullets should be expected to go.*

Reticle slowing – *a standard aim assistance technique by which the camera speed is reduced when the player is aiming at a target's hitbox.*

Rotational – *a standard aim assistance technique by which the camera is turned to keep a target on screen.*

Rubber banding – *when a player's connection is unstable and causes a bug where the player teleports on screen as their movement on their game and the server are out of sync.*

Scripting languages – *a programming language that is interpreted at runtime rather than compiled.*

Skill-based matchmaking (SBMM) – *a process by which players are selected to join a PvP game based on the players' individual measured skill level.*

Skill curve – *the difficulty of learning a new process or game strategy.*

Sticky targets – *a standard aim assistance technique by which the camera stops/slows moving when over a target.*

Target acquisition – *the process by which an enemy is aimed at.*

Target gravity – *an aim assistance technique by which the reticle gets dragged towards enemies when they are near the crosshair.*

V-Sync – *a graphics technology used to synchronise a monitor's refresh rate and the graphics card.*

Visual scripting – *creating code with a graphical environment rather than a written programming language.*

1 Introduction

1.1 Project Idea and Motivation

Problem/Motivation

Video game production is a multibillion-dollar industry. Of the many video game genres, one of the largest and most notable is the FPS genre. First person shooters (FPS) are a genre of games where the player takes on the role of a character in the game, and sees through their eyes; with a combat system based around the player's use of firearms and other ranged weapons (e.g. bows, magic).

Aim assist technologies in games are a constant subject of contentious debate across all First Person Shooter (FPS) games in the video games industry. The most popular aim assistance/aim altering mechanisms are 'bloom', 'Bullet Magnetism', 'rotational', and 'reticle slowing'. In games such as Destiny 2, the discussion even goes deep enough that there is a Twitch streamer (Drewsky) who became well known for analysing the aim assistance behaviours of different weapons by testing them in game. There are a lot of arguments online both for and against the use of aim assist in games.

Project Idea

This project seeks to explore and identify commonly used aim assistance technologies and will proceed to reproduce and critically evaluate selected methods. Once existing solutions have been researched and the artefacts have been created, I will proceed to apply this knowledge to analyse the different approaches in a comparative summary. If possible, there is also the potential to propose/develop my own improved solution(s) if I have identified limitations in the evaluated aim assistance technologies. This knowledge could be used by game developers for gameplay design choices which in turn could benefit gamers by reducing the controversy caused by aim assistance in games.

1.2 Project Aims, Objectives and Targets

Aim

The aim of this project is to create an FPS video game; and to critically evaluate, via a detailed comparative study, commonly used aim assistance technologies in FPS video games.

Objectives

I have identified a number of important objectives crucial in achieving my primary aim. These are:

O1. Conduct a comprehensive review of aim assistance techniques.

O2. Identify suitable evaluation parameters for comparison.

O3. Develop artefacts based on the selected aim assistance technologies.

O4. Analyse the performance of the artefacts in O3 (using parameters identified in O2).

O5. Comparatively summarise the developed artefacts.

O6. Propose solutions. (extension)

O7. Create an FPS game.

Targets

I have also identified key indicators which will demonstrate achieved an aim or objective:

T1. Identification and detailing of aim assistance technologies.

T2. Demonstration of working artefacts.

T3. Described proposed new/improved solutions. (extension)

T4. Demonstration of functional artefacts of proposed solutions. (extension)

T5. Demonstration of working FPS.

1.3 Report format

Introduction

- Brief intro and overview of the project and background – sets out the goals of the project and explains the need for this research.

Literature Review

- Analysis of existing research into the field and background research – used to generate and inspire solutions based on findings.

Project Work & Methodology

- Programming and implementation of systems – setting up the environment and creating algorithms.

Testing, Results, Discussion and Evaluation

- In-depth detailing of processes taken to check quality and efficacy of artefacts – strengths and weaknesses of each algorithm and measured success.

Conclusion, Recommendations & Future Research

- Learnings and suggestions for further study – information gathered and ideas that could be useful to industry and research.

Project Management Review

- Consideration of project in terms of planning and time management – strengths, weaknesses, and learnings.

References

- References to resources used by the project – literature and webpages used for research purposes.

Bibliography

- References to suggested research – not used by the project but in the surrounding field of study.

Appendices

- Additional supporting material – tables, screenshots, a required copy of the project management review.

2 Literature Review

2.1 Algorithm

Aim Assist

Existing research has shown that once taken out of a simplistic testing environment and brought into a more realistic environment with distractions, sticky targets and target gravity techniques can start to fall apart and have reduced benefits. Whereas Bullet Magnetism and Area Cursor work well, presumably due to changing the hit registration after firing the weapon, as opposed to altering the player's input (*Vicencio-Moreira et al. 2014*). It might also be expected that such techniques should be less likely to interfere with a player's 'flow state', as the player still has full control over the reticle; however, it is also worth considering that affecting the players reticle more may help people with accessibility requirements, who may struggle to move the reticle towards an enemy without extra assistance.

This research has been used as a basis for additional findings, the same group also suggest that aim assist helps people to perform better, without affecting players' perceived experience; and aiming and overall FPS skill development was not negatively affected by the presence, or lack of, aim assist (*Gutwin et al. 2016*). Interestingly, there was an observed improvement in aim skill development when adding aim assist to the game. This is very important information, as it shows that aim assist will not remove the need for skill development in the players. There are concerns that too much aim assist may cause players to become 'lazy'. While this research does not answer this concern directly (as it does not determine the effect which different aim assist strengths have on these findings), it supports the idea that (at least at certain strengths) the effect of aim assist does not remove the need for effort from the player. It is also important to note that the only aim assist technique implemented in this test was Bullet Magnetism (the same Bullet Magnetism algorithm used in their previous research).

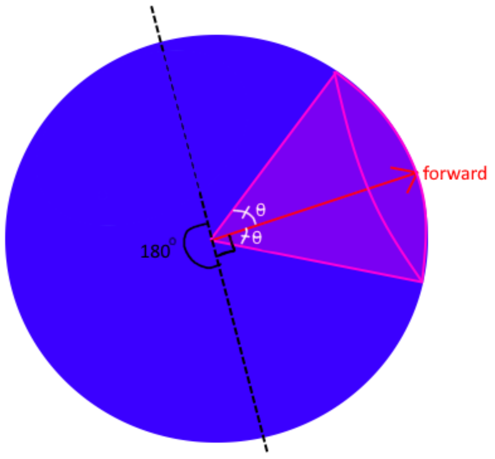


Figure 1. Schematic representation of Bullet Spread.

Bullet Spread is the term used to describe the range of trajectories of the bullet. Figure 1 shows that the bullet can land anywhere within the pink highlighted area. The gun barrel is located at the centre of the sphere shown, pointing forwards as represented by the red arrow. The bullet randomly travels along any one line (vector) in the pink area. This is calculated within a unit sphere to decide the vector at random, and then this unit vector is used to extrapolate the line where the bullet travels along.

Area Cursor can be done with a similar logic, except instead of finding one line for the bullet to travel down, it uses the whole area as if the bullet were expanding with the edges of the cone as it moves forward, and the cone widens. Rather than doing this by increasing the size of the hitbox as the bullet travels, it is more common to just place a 2D shape on screen and see if it overlaps the target (*Vicencio-Moreira et al. 2015*). This is much simpler when coding from the ground up and still has the same effect as if the bullet were widening further as it travels.

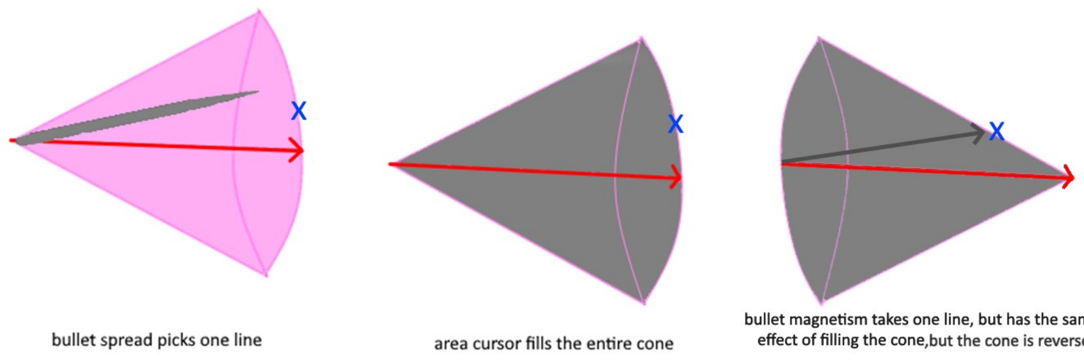

X
bullet spread picks one line
X
area cursor fills the entire cone
X
bullet magnetism takes one line, but has the same effect of filling the cone,but the cone is reversed

Figure 2. Representation of bullet cones for different algorithms.

Bullet Magnetism is supposed to take one line for the bullet to travel, the name "Bullet Magnetism" likely comes from the effect it causes on the line the bullet takes, as the bullet will often take a path bringing them nearer the enemy, even if the line does not actually touch the enemy (*Ryan 2014*). This gives the impression to the player that the bullets are pulled towards the enemy by an invisible force as if by a magnet. For testing purposes, it is much simpler to allow the bullet hitbox to expand and have it fill in this imaginary cone's radius as it travels. Rather than program the bullet's hitbox to travel and check this cone before firing the actual bullet. This means that in game the bullet does not have the visible effect that is characteristic of Bullet Magnetism, but the effect on aim should remain the same. The need for the hitbox to travel before the bullet will also create a tiny delay (likely negligible but still present) the hitbox would be used to scan before shooting the physical bullet, and therefore would need to move faster than the bullet does to reduce this delay. Otherwise, the bullet will take 2x the time to hit the enemy and at far distances or with slow moving bullets the enemy may have time to move and then the bullet will still miss the enemy on screen but register a hit. This would render finding this line completely pointless as the line still won't show that you hit the enemy, which is the whole point of sending out the hitbox to scan first.

"Hitscan" is similar to this idea of sending out the hitbox to determine a hit before sending the physical bullet, except it never sends the physical bullet, The scan is only done for hit detection – it does this with a line of infinite speed to check immediately. This removes travel time from bullets and therefore the game becomes less realistic. At far distances in some games, you have to "lead" your bullets by anticipating where a moving enemy will be and firing there instead of at the enemy themself, as the bullet would not hit them if they had moved (*Fully Loaded 2022*).

2.2 Related algorithms

<u>Input devices</u>

Games can be played with a variety of input devices, the most common being dual analogue stick controllers, and mouse and keyboard. It is worth considering that these will influence the accuracy and balance of a game. However, this does not necessarily mean that there will be an effect on player experience. While the number of deaths could potentially be linked to the player's choice of input (due to skill curves and hardware limitations), player experience seems to be more affected by the comfort of the player than the hardware capabilities (*Gerling et al. 2011*).

Gamers tend to have incredibly varied sensitivities and mouse dpi's. These factors influence aim accuracy by directly changing the speed the player can rotate the camera. The importance of this is that the player's time taken to drag to an opponent will be affected by these factors. If they are too low, it will greatly slow down their turn speed and interrupt gameplay; but if they are too high, the player will likely move the crosshair too far and have to readjust afterwards. Nvidia has found that a safe range for these factors is one that gives a degrees per mm of 0.45-1.8 (*Boudaoud et al. 2023*).

Measures

There are existing methods of predicting target acquisition. One notable method which came up in numerous research studies was "Fitts' Law". It suggests that the time taken to aim at a target is affected by the distance to the target and the target size. Fitts' Law has been shown to be a suitable model for 3D games using pan-based aim controls (where the players input rotates a camera) (*Looser et al. 2012*).

There are also more generalised measures existing for player experience in games. One such measure is the GEQ (Game Experience Questionnaire). After playing, the player will fill out the questionnaire, indicating on a scale of 0-4 how much they agree with each statement. The GEQ measures many different aspects of the player's gaming experience. The results are taken, and each measured aspect has a list of questions that affect its score. The overall score for each aspect of the experience is the average value of all its related questions (*IJsselsteijn et al. 2013*).

Delay

Delay can affect gameplay performance and flow of any game, but especially fast paced games such as RTS (real time strategy) and FPS. This delay can stem from a great many sources: client-server data transfer, client-client data transfer, input device latency, output device latency, player reaction time. The local latency is considered to be the latency caused by the delay from the player's closed system (i.e. the players input and output devices).

There are existing tools to minimize the effects of local latency. For monitors/displays, latency reduction technology often comes standard included in gaming models. However, their real-world performance is debatable. Many gamers turn on some form of V-Sync for almost every game they play, whereas others believe it to make no difference or even make their performance in-game worse. Research has shown that for a properly suited PC setup there is no reported benefit of V-Sync, although this may be different for higher performance setups – where V-Sync is supposed to do its best work (*Lee & Chang 2015*). It is worth pointing out that V-Sync is not actually a latency fix, but rather a solution to screen tearing which makes it easier to interpret objects on screen for the player. V-Sync itself can actually increase latency as the monitor has to handle the output differently. Aim assistance technologies have been shown to offput some effects of local latency for target acquisition, and almost all effects for target tracking (*Ivkovic et al. 2015*).

High latency across networks has been shown to have a noticeable effect on gaming performance. The 'netcode' of a game (the algorithm for handling network communications) can often affect performance and player experience; and are manipulated to reduce any delay (*Ahmed et al. 2022*). These manipulations can often improve experience for the player encountering network issues (*Lee & Chang 2015*), however they can worsen the player experience for players with faster connections. Issues can present in the form of 'rubber banding', shooting around corners, network abuse/cheating, 'ghost' and 'magic' bullets. For this reason, many games developers struggle with deciding whether to implement such systems. Games such as those in the Destiny franchise have implemented different systems across different parts of the same game and often changed the ways they work due to player feedback (*Willet & Hogan 2019*).

2.3 Development Environment

Game engine

Games are developed in specialised development environments known as 'game engines'. These game engines can facilitate the programming, graphics, physics, animation, and many other aspects of game creation all in one environment, reducing the need for multiple software packages. The most well-known game engines are 'Unity' and 'Unreal Engine'; although many other engines exist such as 'GameMaker' and 'CryEngine'.

For this project, there is a requirement to create working artefacts. These artefacts should be software-based but can be created in any choice of suitable programming language and development environment. For this project, an attempt will be made to solve a problem in the gaming field. This means that my language and environment choices should be standard to the industry.

Unity and Unreal Engine both have resources for beginners, but CryEngine is considered much more complex to learn (*Šmíd 2017*). This would not be a suitable choice of game engine for this project due to the time restraints of the project and CryEngine's steep learning curve.

Many games studios will develop their own proprietary game engines to better suit their own needs, and will have a dedicated team just for working on and updating this engine. Again, due to the complexity and time restraints, this will not be suited to my project. Creating my own engine would likely be beyond the scope of the project on its own, even without the need to create a game or write up the process.

Both Unity and Unreal Engine have a very large community of users, made up of both professional games studios and individual hobbyists. They are both considered incredibly dependable, and offer lots of support, in the form of documentation, tutorials, and assets. Unity offers assets for game creation such as 3D models and software development libraries via the Unity Asset Store, and Unreal Engine offers theirs via the Unreal Engine Marketplace. Both asset libraries contain paid, and free assets for people to use. Unity offers frequent version updates; all available for download. One problem caused by this is the incompatibilities between versions, as many projects and libraries will not work across different versions. This means that for specific libraries and use cases you are required to download an old, outdated version of Unity. For example, for VRChat avatar creation, you must use the specific Unity version “2022.3.6f1” (*VRChat 2024*), which is over 40 updates old, or 1 full update version. Unreal Engine, on the other hand, has much less frequent updates. The current version being Unreal Engine 5.3 (*Epic Games 2024*), although it is relatively new there is support to transfer Unreal Engine 4 projects across to Unreal Engine 5. This makes development much simpler, as there are less likely to be incompatibilities with libraries and past projects.

<u>Programming language</u>
The programming language available to programmers for Unity is C# - a scripting language which is interpreted line by line as the code runs, rather than needing to be compiled first. Unreal Engine uses C++ which is a non-scripting language. C++ is compiled and therefore does not need to be interpreted at runtime. Scripting languages are often more heavily abstracted, and contain many more existing keywords/pre-defined functions for use by the programmer. This means that the same program can be made much quicker and in fewer lines of code than in a non-scripting language. However, the memory consumption of a scripting language is considerably higher. Scripting languages also take longer to complete tasks, C and C++ are considerably faster in comparison to scripting languages (*Prechelt 2000*).

C++ in Unreal Engine and C# in Unity both follow object-oriented programming practices. This means that you create abstracted objects with their own methods and variables to be instantiated. These object instances can store different values to present different behaviours. While they focus on object-oriented programming, the separate program files are stored within scripts, and these scripts can be programmed using many different programming paradigms. A common choice in C++ and C# is functional programming. Functional programming allows the programmer to define methods to be called and alter the data and control flow to give the desired outcome (*Ambler et al. 1992*).

Both Unity and Unreal Engine offer an alternative to a classical programming language: visual scripting. Visual scripting offers a way to program without the need for knowledge of specific programming syntax. It is often block based, with the program flow controlled by flowcharts or nodes. This is often a great alternative to written programming languages for inexperienced programmers. It can also be faster to program with. However, it is not without its downfalls. In Unreal Engine, visual scripting is slower at runtime than C++, because it is not compiled directly into machine code before running (*Knutsen 2021*). It also has a learning curve, which, while easier to learn than written programming languages, takes time. Due to personal knowledge of many different written programming languages, the time taken to learn visual scripting could be better used.

Due to personal knowledge and the scope of this project, the algorithms will be created in Unity with C#. It is an engine used in industry on a professional level, and also usable for beginners. Unity also has a vast number of resources available to developers both from Unity themselves and third parties (*Vohera et al. 2021*).

2.4 Open-Source

In order to produce the artefacts, existing libraries will be used. Unity has offerings from first-party and third-party developers. An existing FPS game will make a good basis for my artefacts. Two potential open-source FPS game projects have been identified that can be developed upon and be adapted to suit the project's needs. These are both created by Unity themselves and are: "FPS Sample" and "FPS Microgame". Both are intended as the groundwork for an FPS game to be developed in Unity, designed for people to make their own games. Upon first glance, FPS Sample is much more visually appealing and has graphics on par with many current AAA FPS games. It seems well suited to a basis for a fully-fledged FPS game a professional studio might make. FPS Microgame on the other hand is very cartoony, and a less complete game experience. However, its intended use is for hobbyists and learners. It is supposed to be simple and easy to understand. In line with this, FPS Microgame also has a lot of available resources for development. Unity offers instructions on how to add assets, change game mechanics, and provides step-by-step tutorials to teach programmers how to work with it (Unity 2023). FPS Sample also has video tutorials, although much less in-depth. The resources available for FPS Sample are of a much higher quality, however the project uses experimental technologies and warns that the project is not supported, making it unsuitable for actual game development (*Unity 2018*).

Learnings

From reviewing this literature, the importance of research into games development becomes very apparent. Many intricacies exist even in what may seem to be the most insignificant algorithms. There is much debate and controversy between gamers; and there are many reasons for this. The most dangerous trap, that most games developers seem to be unable to avoid, is that many systems implemented to help one group's experience, detract from another group's experience. The ideal solution to this would be to find a new approach to these controversial game systems which has an equal effect on all groups, and considers every possible gaming setup, personal reaction time, and connection type.

The game will be developed in Unity due to its large provision of resources and my existing knowledge of the software. It must be ensured that the game is fair for all players, and meets the criteria laid out in existing measures of player experience.

FPS Microgame will be used for the artefact's development because of its simplicity and the ready availability of learning and development resources.

3 Project Work & Methodology

3.1 Software Engineering Principles

The project follows a waterfall software development methodology. Sequential stages are completed one-by-one in a designated order. These stages are: project selection, identifying aims and project specifications, existing solutions analysis / research, solution development, testing; and results analysis. This report documents each stage in detail, showing what each stage entails and how they are completed. The creation of a Gantt chart at the start of the project detailed the ideal timeline for each of these stages. While this chart was not strictly always adhered to, it offered a useful guide for the stage ordering, to ensure that current work was completed before moving onto the next stage. This was also made possible by the clear specifications set out in the beginning of the project.

While the overall project follows a waterfall methodology, the algorithm development, testing, and solution analysis took a more agile approach. Rather than being done sequentially, like the rest of the project, they took place cyclically, as would be done in a model such as scrum. During each solution's development, the algorithm was developed, tested, analysed for any downfalls, and then loop around back to development; starting the cycle again. This process repeated until one solution was completed, the second solution, then the third, thus the project reverted back to its previous waterfall methodology.

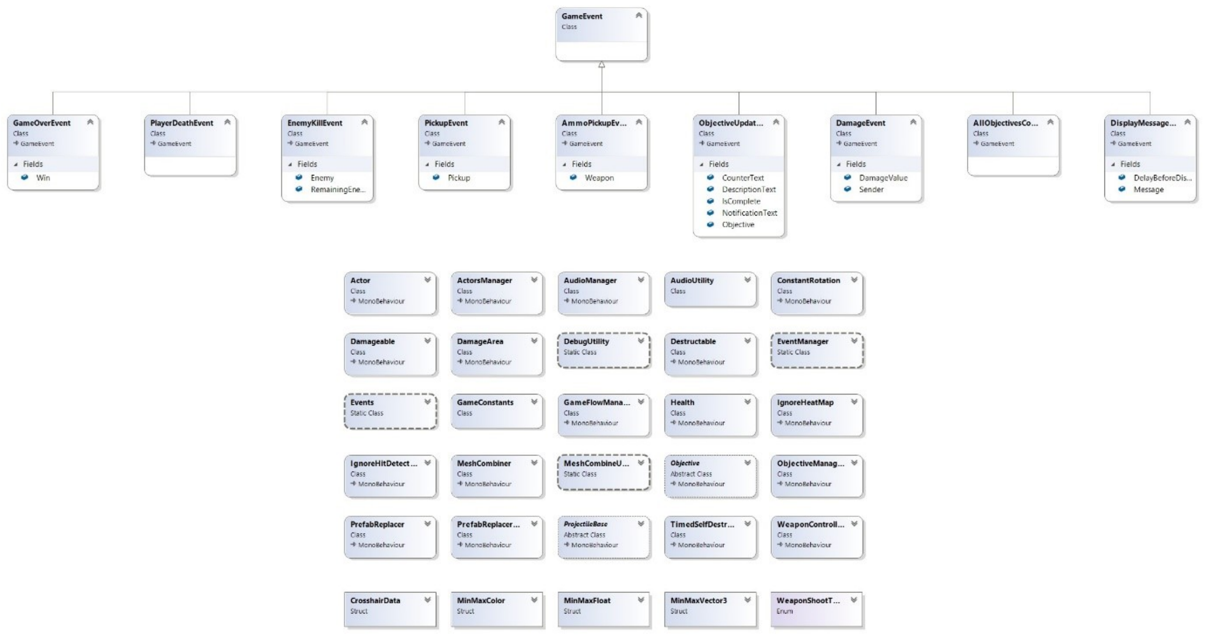


Figure 3. UML Class Diagram for the game's event system.

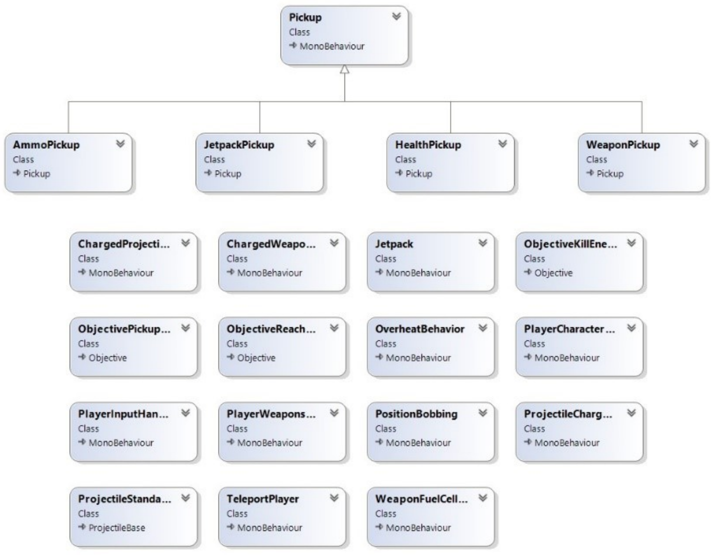


Figure 4. UML Class Diagram for weapons.

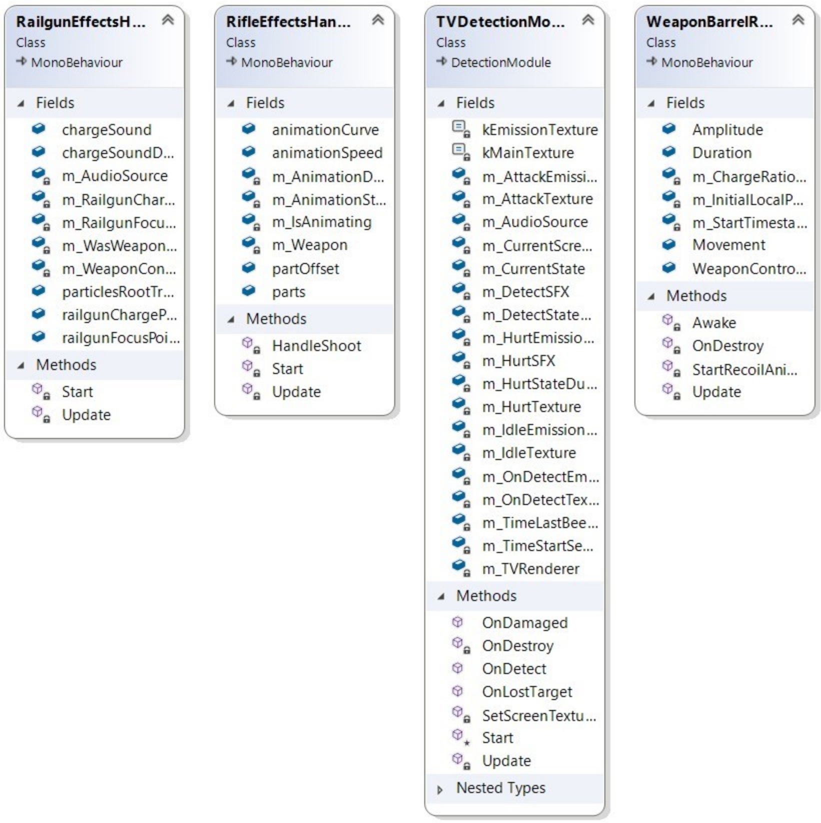


Figure 5. UML Class Diagram for weapon SFX.

Object-oriented programming paradigms are well suited to graphical representations due to their well-structured nature. UML (Unified Modelling Language) is used to create Use Case Diagrams, Object Diagrams, and Class Diagrams. These abstract systems into separate components and functionalities to break down the system for easy understanding to humans. As shown in figures 3-5, a Class Diagram can present the different classes in a system, and their use of inheritance. They can list all variables and methods required by a class, as well as showing the connections to other classes. Visual Studio offers automated Class Diagram generation for code. Getting this to work was not straightforward, as support online often guided to simply viewing the Class Diagram, but did not offer troubleshooting solutions in the event of these steps not working. The library enabling the use of Class Diagrams did not install with Visual Studio, and so needed adding via the Visual Studio installer. Once this was done, Visual Studio created the Class Diagrams for each system's C# code. Without the use of the Object-Oriented paradigm and its clearly defined class structure, these diagrams would not be as easy to automate. This structure is a great advantage to Object-Oriented Programming compared to other programming paradigms which are less often straightforward to present visually.

3.2 Open-Source Packages

Unity and Unity Hub were already installed on the PC and laptop intended to be used for development. However, different versions of Unity are required for certain open-source projects. Using the wrong Unity version would likely cause errors due to incompatibilities and dependencies. This made it incredibly important that a supported Unity version for FPS Microgame was installed via Unity Hub (Unity 2019.4.31f1).

Within Unity Hub, a new project was created using the FPS Microgame template. This template is marked as "learning", and had plentiful resources and support.

It was then important to play the game to ensure that it would be a suitable basis for the project artefact. The game did not need compiling or exporting, only to open the game window in the Unity editor, and then hit play. After playtesting, it could be seen that the game had two different types of enemies, and multiple empty rooms that could be utilised. It was already setup for basic gameplay, even with a menu screen that opens after completing the level, although this is not needed for the artefact. The artefact is not intended to be released, only used for demonstrative and testing purposes, so it does not need full functionality.

The open-source game seemed to be perfectly suited to the needs of the project, however it still needed to be checked for adaptability. Looking through the existing objects in the scene hierarchy, it was apparent that they should be easily customisable and adding new assets should not be difficult either. The assets in the project were plentiful, however not all assets came pre-downloaded. And the weapon included was fully automatic and fast firing, which was not ideal for testing.

A more suitable weapon for testing accuracy and reliability of an aim algorithm would be a weapon where good aim is imperative and the weapon itself is considered highly accurate. Fortunately, the online tutorials for the FPS Microgame included instructions for installing a sniper rifle asset; possibly the most accurate and aim-focused weapon of the FPS genre. Following the instructions led to the Unity asset store to download the project asset bundle. The asset store used to be available within the Unity editor, which made adding new assets to a project incredibly easy, however this function is no longer supported. Although there is a button on the webpage, which imports the asset folder directly into the Unity editor.

After importing the assets, a warning appeared explaining that the Unity version used by the project was not compatible with the assets intended for it. This meant that the Unity project first had to be converted to a newer Unity version. Converting Unity projects to different versions can cause many errors, fortunately there were no errors or noticeable bugs when checking the error log and playtesting the game in the newer version of Unity.

The assets were then imported by clicking the button again, this time importing the assets folder into a compatible Unity version (2020.3.17f1) with the FPS Microgame open. After checking the error log and playtesting once again, the game was still fully working with no errors or bugs. The sniper rifle appeared on the floor as a pickup and worked as intended; it seemed like it would be more suitable for testing than the existing weapon.

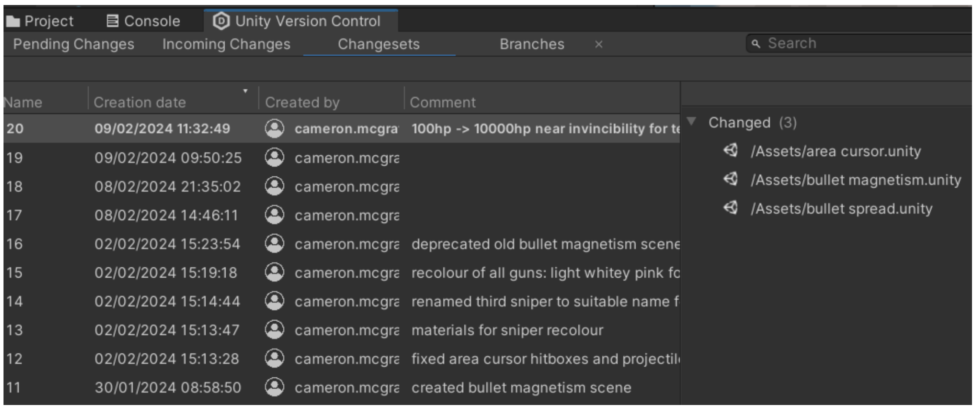
3.3 Version Control

Creation of a shared repository was necessary to enable collaboration between the laptop and PC during development. The laptop is a lower specification, and through playtesting, can be seen to run the game at a lower quality. The resolution needed to be turned down, and the input from its connected Bluetooth mouse was noticeably delayed. This is bad for testing the algorithms as the mouse input translates badly onto the screen and therefore the accuracy is harder to observe, making the algorithms incomparable. Being able to access and work on the project from the higher specification PC was important to solve this problem.

GitHub is one of the most popular services offering online repository storage for collaboration and version control. GitHub was already installed on both devices and so using this should have been easy and effective. However, when adding the Unity project folder to a new repository, there were error warnings from GitHub stating that the end-of-line formatting on the Unity project files was incompatible with GitHub. The error said that next time GitHub touches the files it would change the end-of-line from "LF" to "CRLF". In order to prevent GitHub from changing the data, there are many different solutions recommended online. Attempting to configure from the command prompt, changing the file ".gitignore", and changing ".gitattributes" did not solve the error. It should be possible to use GitHub for Unity, but due to the challenges faced, it seemed easier to find an alternative software package.

After researching alternative version control offerings, it turned out that Unity offers their very own version control service; which would mean that it should be fully compatible, as its sole intention is to be used with Unity projects. The version control offered by Unity is called "Unity Cloud" and offers cloud storage, version control, and many other services for software development. After attempting to enable version control by creating a new Unity Cloud workspace, it was easy to connect the project to the cloud services offered. However, there was seemingly no way to setup the version control feature itself. This was very confusing, after further reading, it was found that while Unity Cloud offers many services for free, version control is unavailable to the public and is currently in a closed beta. This meant that it would be impossible to use this as it was inaccessible.

There were other solutions for version control also offered by Unity, which should hopefully be compatible. Many resources for "PlasticSCM" exist in the Unity editor and online. After trying to set this up, it was again found to be impossible. Unity has deprecated support for PlasticSCM. Since the editor version required by the open-source game had to be an older version, the editor still had the functionality to connect to PlasticSCM, but PlasticSCM itself was no longer compatible.



Name	Creation date	Created by	Comment
20	09/02/2024 11:32:49	cameron.mcgra	100hp -> 10000hp near invincibility for te
19	09/02/2024 09:50:25	cameron.mcgra	
18	08/02/2024 21:35:02	cameron.mcgra	
17	08/02/2024 14:46:11	cameron.mcgra	
16	02/02/2024 15:23:54	cameron.mcgra	deprecated old bullet magnetism scene
15	02/02/2024 15:19:18	cameron.mcgra	recolour of all guns: light whitey pink fo
14	02/02/2024 15:14:44	cameron.mcgra	renamed third sniper to suitable name f
13	02/02/2024 15:13:47	cameron.mcgra	materials for sniper recolour
12	02/02/2024 15:13:28	cameron.mcgra	fixed area cursor hitboxes and projectil
11	30/01/2024 08:58:50	cameron.mcgra	created bullet magnetism scene

Changed (3)
- /Assets/area cursor.unity
- /Assets/bullet magnetism.unity
- /Assets/bullet spread.unity

Figure 6. Screenshot of Unity VCS within the Unity editor.

PlasticSCM was transferred to a new version control package: “Unity VCS”. After accessing online, Unity VCS was connected via the Unity Hub rather than the editor. But once activated, it is still managed in the editor. This package seemed to connect and once in the editor it had buttons to push updates and request incoming changes. After connecting the PC to the same project, it seemed to have shared the project via the cloud. And after testing the push/pull button functionality, everything was proven to work. This meant that the project could be worked on from any connected device, and could be reverted to previous versions much more easily.

3.4 Algorithm 1 – Bullet Spread

The first algorithm to be recreated was Bullet Spread. This sends out a bullet randomly within a certain angle in 3D space. The sniper rifle asset already included a public variable to select a Bullet Spread angle, but was set to 0 and therefore unused. This made enabling this very easy, but determining the correct value difficult, as the value of this variable will greatly affect gameplay experience.

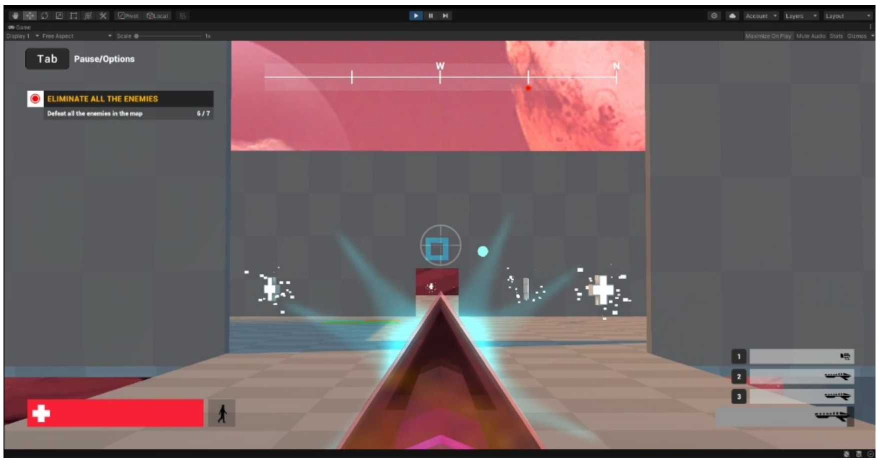


Figure 7. Bullet fired with high Bullet Spread. The bullet doesn’t align with the crosshair.

It was decided to test three different strengths for each algorithm, one with the aim algorithm off, one with low strength, and one with high strength. This allows easy comparison while gameplay testing. In order to create three weapons with different aim assist strengths, the sniper rifle was duplicated twice and each was placed next to the original sniper rifle on either side. The sniper on the right was set to a value of 0, the middle to 1, and the right to 5 degrees of Bullet Spread.

This worked perfectly for the weapons, through playtesting it could be very easily seen that the weapons exhibited the desired different behaviours. This made it possible to test on the enemy in the first room of the game, however there is only one enemy in the room and three different weapons that need testing. This enemy then dies instantly to the first bullet hit and then disappears. There is also an enemy in another room however they are much larger which will affect the testing greatly and corrupt the results. The enemy is also immobile unlike the smaller enemy.

The enemy could have been given extra health, but this would make it less noticeable when the bullets hit and when they don’t, as only a small chunk of the health bar would disappear rather than the whole health bar and the enemy. Instead, duplicates of the mobile enemy were created by copying the existing enemy in the project hierarchy and then moving them to be next to the existing enemy, as was done when replicating the sniper rifle. Playtesting showed that they could take damage when hit, and that this was much better for comparison as a single enemy could be targeted by the player for each different weapon. The enemies, however, did not move.

Looking into the scripts and reading the code in them for the enemies, it could be seen that the enemies’ movement is restricted to an instance of a separate object “patrol” which was not duplicated with the enemies. After duplicating this twice and placing the patrol perimeter objects on the ground below each enemy, they were still immobile.

After looking at the patrol objects themselves, a public variable was minimised (closed in the editor) and when opened it could be seen that the patrol object class takes an enemy instance as an input. All three patrol instances were still setup to connect to the first enemy however the patrol perimeters did not overlap. Therefore, the original enemy was still only controlled by the original patrol instance, while the duplicate ones were unaffected by any of the patrol instances and did not move. Upon changing these input values to the enemy instances above each patrol instance, the enemies all became mobile and would move within the confines of their given patrol perimeter.

It was noticed that while the playtests could show some strengths and weaknesses, it would not be fair or properly representative to only test aim assist methods on enemies at close range. Therefore, the large outdoor section of the map was utilised to place 3 more enemies further away for testing these algorithms at long range as well as the short-medium range enabled by the previous room.

The scene was then duplicated and all values for Bullet Spread were set to 0 to be used as a template. Any new algorithms created could be implemented by duplicating this template scene and then adapting the code. This eliminated any need to start from scratch and redo any work setting up the scene with more enemies before working on each algorithm. Upon reflection, a better way to do this would be to use "prefabs" to save a scene as a class and then abstract that scene within child scenes; further making use of Unity's object-oriented technologies.

3.5 Algorithm 2 – Area Cursor

Area Cursor creates an imaginary cone in the 3D space to make shooting targets further away easier. The common way of implementing this, is not to attach a hitbox to a travelling bullet. But rather to create an invisible 2D rectangle or other 2D shape on the GUI. If the shape overlaps with the target on the screen, then the bullet is considered to hit the enemy. The current implementation of the game uses physical bullets with travelling hitboxes. The effect of the 3D cone can be replicated with physical bullets by expanding their size by a constant as they travel. Changing the size of the bullet itself makes no difference, however would look very strange in game, so instead only the hitbox size was changed.

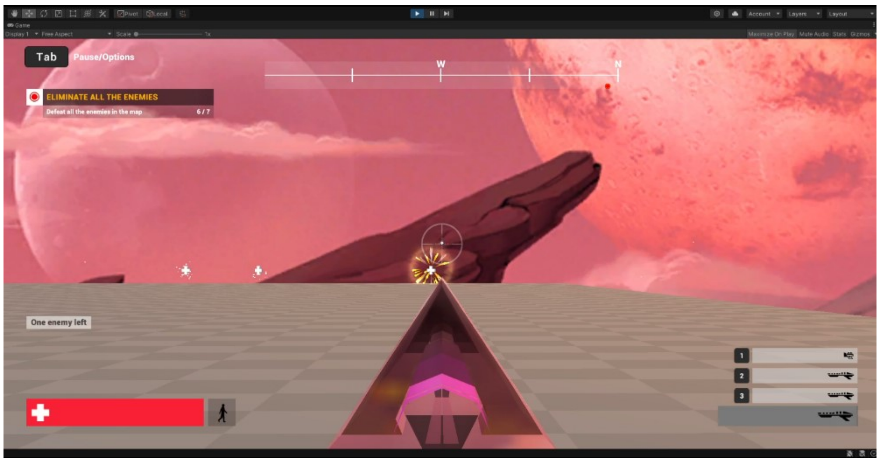


Figure 8. Enemy hit with high Area Cursor. The crosshair is not over the enemy.

Changing the size of the hitbox could not be done within the sniper rifle object, as the sniper objects take a projectile (bullet) as an input. The hitbox of the bullet is a property within the projectile object. The projectile class is not instantiated and so different strengths cannot be tested. To solve this, the projectile object files were found in the file explorer and duplicated to allow a different strength for each.

The new projectile objects were added to different sniper rifles as inputs, and could then allow different aim assist strengths for each sniper rifle. The algorithm could then be programmed within the projectile script. The script for the projectiles was already programmed functionally, allowing easy adaptations to be made by creating new functions or editing existing ones. A public variable was created at the start of the scripts, following the existing script structure. This variable (“radiusIncreaseFactor”) is public so that it can be seen and have its value changed within the editor, and is used to control the radius of the hitbox for the projectile. The size of the radius was incremented by this value every frame update, inside the Update() function. However, this is bad programming practice in game development, as everything needs to be smooth and flow well. This is why the increment value used was changed to be not just the value of the variable, but rather the value of the variable multiplied by the time taken by the processor between this frame and the last (Time.deltaTime). This is standard practice in games as it means that when a frame update happens more or less frequently, the value will be adjusted accordingly and the rate of change will appear as constant to the player, giving a smooth experience. Without this adjustment, the imaginary cone would not be smooth, meaning that projectiles would become less reliable, as the cone would be wider or narrower at points for every bullet fired; missing or hitting enemies that other cones produced by the same projectile may not.

Through playtesting, this could be seen to work perfectly. While not a classical implementation of Area Cursor, it had the exact desired effect. One weapon was set to a value of 0, the next to 0.05, and the last one to 0.2. All three worked via playtesting and showed promising results.

A new public variable was created for debugging purposes to store the current radius of the hitbox. This meant that while in the editor the size of the hitbox for a projectile could be seen, to check that the variables are behaving as expected. The projectiles travel so fast that they disappear quickly, and while playing it is near impossible to find them manually in the hierarchy. By searching for the term “projectile” in the hierarchy, these projectile instances would be the only assets shown in the hierarchy and could be instantly found when instantiated. The value of the current radius could be seen to increase as the projectile travelled, as desired.

While the use of Time.deltaTime worked perfectly, it was difficult to find the code for increasing the radius in the Update() function. A simple solution to this would be to move the code into its own function called something like “increaseRadius()”. However, there is a key function existing in C# that was not utilised in the projectile script and so could be used. This is the FixedUpdate() function. FixedUpdate() is similar to Update() except it is not called every frame update, and rather is called at a set time interval, to ensure smoothness. It is also called by default at these intervals, so does not need to be explicitly called by another line of code. Due to these benefits, this function was used instead of defining a new “increaseRadius()” function. This meant that Time.deltaTime could be removed from the radius increase code.

3.6 Algorithm 3 – Bullet Magnetism

Bullet Magnetism is classically programmed differently to Area Cursor, the effect produced is the inverse. The desired effect can be created by adapting the Area Cursor script to a negative rate of change. Instead of increasing the radius of the hitbox, it should be decreased. While the classical way of creating this effect in games is to send out a scan with infinite speed, the physical effect can be simulated by adapting the Area Cursor algorithm. The only difference to the player will be that the actual bullet itself does not fire directly at the enemy (or as close to them as possible). This will not affect gameplay but will affect player experience, as visually it behaves differently to Bullet Magnetism in other games.

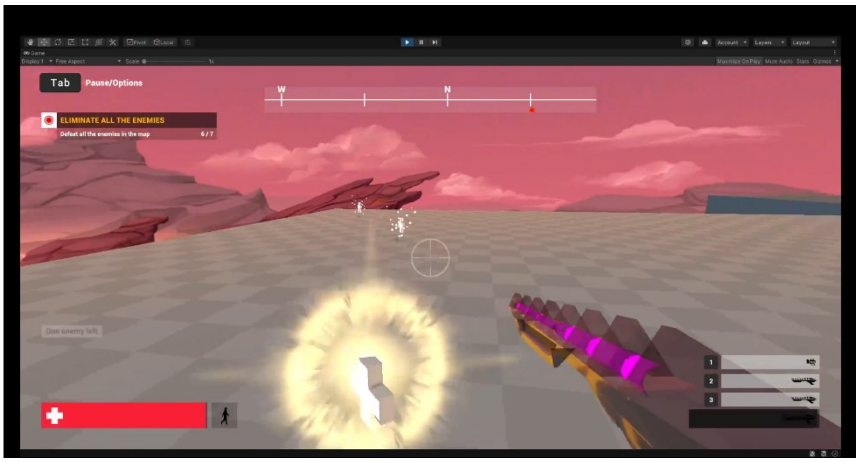


Figure 9. Enemy hit with high Bullet Magnetism. The crosshair is not over the enemy.

This was easy to implement, by simply creating a new scene based on the Area Cursor scene, and duplicating new projectile files; then inputting negative "radius increase" values. This, however, caused the bullet's hitbox size to not only reach 0, becoming non-existent, but to further decrease to negative values, meaning that the hitbox actually got bigger. Thus, a minimum hitbox radius was required to ensure that it would not go negative or reach 0 (set to 0.1, the same as the starting radius of Area Cursor). This value was checked in an if statement to control the script's flow; the value would only increase when the condition is met (when the radius is larger than the minimum). A flag variable was also created so that upon instantiation of the projectile, it could be seen that the bullet was shot, this was also included in the condition for the if statement, requiring both conditions to be met.

This then worked but ruined the script's functionality for the Area Cursor algorithm, which was not noticed until demonstrating to the project supervisor. The problem was that the minimum radius set was the same as the starting radius for Area Cursor. This meant that the bullet from Bullet Magnetism would stop changing size when it reaches the minimum, but the Area Cursor bullet would never change size, as the requirement to stop changing the radius would already be met from the start. To fix this, the minimum radius was changed from 0.1 to 0.999 so that it is near identical, but the Area Cursor and Bullet Spread algorithms will no longer branch when they are not supposed to. An else was added after the if for completeness, so that if the radius is somehow reduced further than to a size of 0.1, it will be readjusted back to 0.1 in the else clause.

Once all three algorithms were working fully, the different algorithms and different strengths could be seen to have a big impact on gameplay. It was confusing trying to remember which weapon was which, and so the decision was made to change the colours to make them distinguishable in the player's hands. In all scenes, the weapon with no aim assist was made a light, pale pink; the medium aim assist a bright red; and the high aim assist a dark, near-black red. This reduced the need for additional wasted time testing the guns to figure out which was which.

As the enemies fight back, the player would often die if trying to stand still while testing weapons. Player health was increased from 100 to 10,000 to prevent death, making the player near-invincible. Upon death the level restarts but returns to its original state with only one weapon, which is not suitable for testing; this health increase meant the likelihood of this being a barrier to testing was heavily reduced. An alternative solution would be to remove the enemies' ability to shoot back.

4 Testing, Results, Discussion and Evaluation

4.1 Software Testing

The developed software solutions focus on the topic of player experience. Since the output is observed by the player, measurements often become subjective. According to personal understanding of the systems, they should be good representations for the results seen in industry. However, they are aesthetically and physically different, this was done in order to better fit the open-source project used. The effect is a big factor for player experience, but so are the aesthetics. The physics behind the algorithm are considerably less important for experience than the actual effect on hit registration, but they will have an impact on the technical performance of the game.

Debugging the software is made incredibly easy through the Unity editor. If a script has errors compiling or anything that could cause potential issues, it will be flagged in the editor immediately with a warning in the debug console. The game becomes unplayable until the error is resolved properly, but this ensures that errors do not go untreated. For problems with duplication of asset files, this warned that files were missing, and gave detail on what file was needed; these missing files were then easily duplicated. C# offers public variables which Unity makes use of by displaying them in the editor. This way, changes to these values were monitored while playing the game to see what how the data is affected and check for desired behaviours or unintended actions. Public variables were incredibly important for debugging, as they allowed the radii of the bullets' hitboxes to be monitored. These hitboxes are not visible in game, and so the ability to see the size of the hitbox changing meant that problems could be found and addressed. These public variables also meant that flag variables could be easily created to watch when an event had occurred. Flag variables could also be created to signal an event occurrence via printing to the console with "Debug.Log()"; this enabled testing the scripts' existing functionality and control sequence to better understand if a section of code was unclear, for example for an understanding of what happens in the script when a bullet is instantiated.

4.2 Results

The effectiveness of the algorithms can be tested both qualitatively and quantitatively. A quantitative analysis can be done using the GEQ as detailed in the Literature Review, and by monitoring hardware usage during gameplay. A qualitative analysis of the algorithms can be done by playtesting in different imagined gameplay scenarios.

<u>Games Experience Questionnaire (GEQ)</u>

The GEQ is a measure of player experience created by researchers and used in industry. The questionnaire is split into sections; and since this project does not include a fully shippable game, it will only be tested against one section of the GEQ: in-game. This measures player experience through themes already explored such as flow and difficulty of the gameplay. The first question has been removed from the questionnaire, as it relates to a story for the game; and no such story exists in the artefacts created. Due to project scope and ethical concerns, the results were only collected from one individual; adding to the already high subjectivity of the questionnaire's results.

	Algorithm		
	Bullet Spread	**Area Cursor**	**Bullet Magnetism**
Competence	1	4	3
Sensory and Imaginative Immersion	0	2	2
Flow	0	3	3
Tension	3	0	0.5
Challenge	3	2	2.5
Negative effect	3	0	0
Positive effect	0.5	4	3

Figure 10. Tabulated GEQ values for each algorithm.

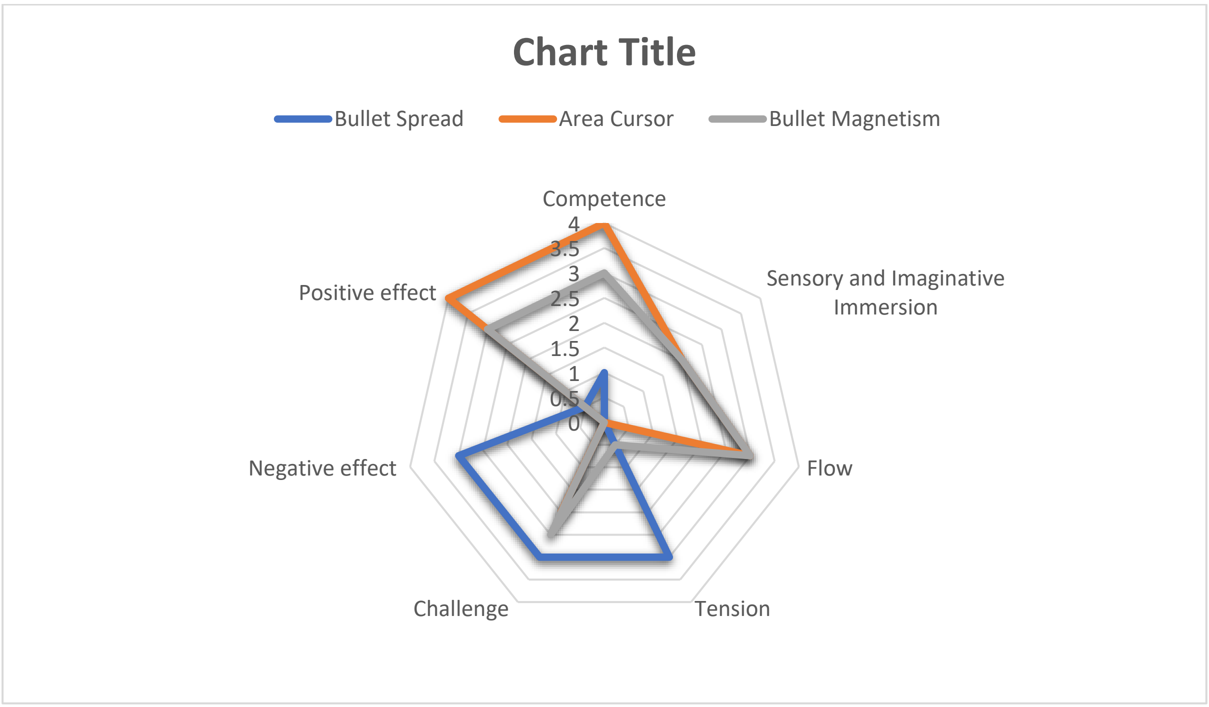


Figure 11. Radar chart overlaying algorithm results from GEQ.

The results from the GEQ show strong similarities in player experience between Area Cursor and Bullet Magnetism, while Bullet Spread is vastly different. Both Area Cursor and Bullet Magnetism created feelings in the player, of competence, positive effect, flow, challenge and immersion. These are good for gameplay and contribute to a good player experience. They also showed no feelings of negative effect in the player, indicating that there was no annoyance or intrusion from the systems which ruined the overall core gameplay. Bullet Spread showed low values for: flow, immersion, competence, and positive effect. Meanwhile the values for challenge, tension, and negative effect were all high. These indicate a negative experience, except for the feelings of challenge which are more subjective, but can be considered positive. There were very low values for tension in all of the algorithms, likely due to the lack of story in the game providing any motivation or feelings of risk.

These scores indicate overall that Bullet Spread is not a suitable aim adjustment algorithm to be used for aim assistance; while Area Cursor and Bullet Magnetism are much better suited for this purpose. This aligns with industry standards. While bullet spread has potential for aim assistance purposes in very small amounts, in industry it has been known to be used for making aiming more difficult, for instance when running. Area Cursor and Bullet Magnetism are both used for aim assistance purposes in industry, and have shown to give good results for player experience. Area Cursor received higher scores than Bullet Magnetism suggesting that it is more beneficial to player experience.

Playthrough

The following playthrough describes what the player did when testing the game, before the GEQ was measured. This process was the same for each algorithm. The player showed a clear preference for the most forgiving weapons, and would test them briefly to learn how to use them and push them to their limits.

> *Upon starting the level, three different sniper rifles were presented in front of the player to be picked up, each with a different colour. Walking over these snipers picked them up and added them to the player's inventory. The snipers were added to the GUI to show what weapons were available and which was active. Through the doorway could be seen a small room with three enemy robots, moving around on the spot.*
>
> *After entering the second room, the enemies were shot at, with one sniper being used on each enemy. These snipers displayed noticeably different behaviours, with one being the easiest to hit their respective enemy with, and another being the most difficult.*
>
> *Once the three enemies were defeated, the player returned to the first room to test the different sniper rifles' behaviours. The snipers were each shot into the sky, round corners, and through the doorway to determine how their bullets behaved both on their own, and when interacting with other game objects.*
>
> *The player then entered the second room, and went to the outside section to the left. This area had three more of the same enemies, split further away from each other, at a far distance from the player. Shooting at these enemies with each weapon showed that the behaviours of each weapon became exaggerated over a longer distance.*
>
> *Next the player walked back through the second room, and into the boss room. The boss was a much larger robot, although was immobile. The player quickly defeated the boss, choosing to use the most forgiving sniper rifle, while running around them for entertainment and to push the forgiving nature of the sniper to its limits.*
>
> *The game state changed and showed a menu screen to restart the level.*

Hardware

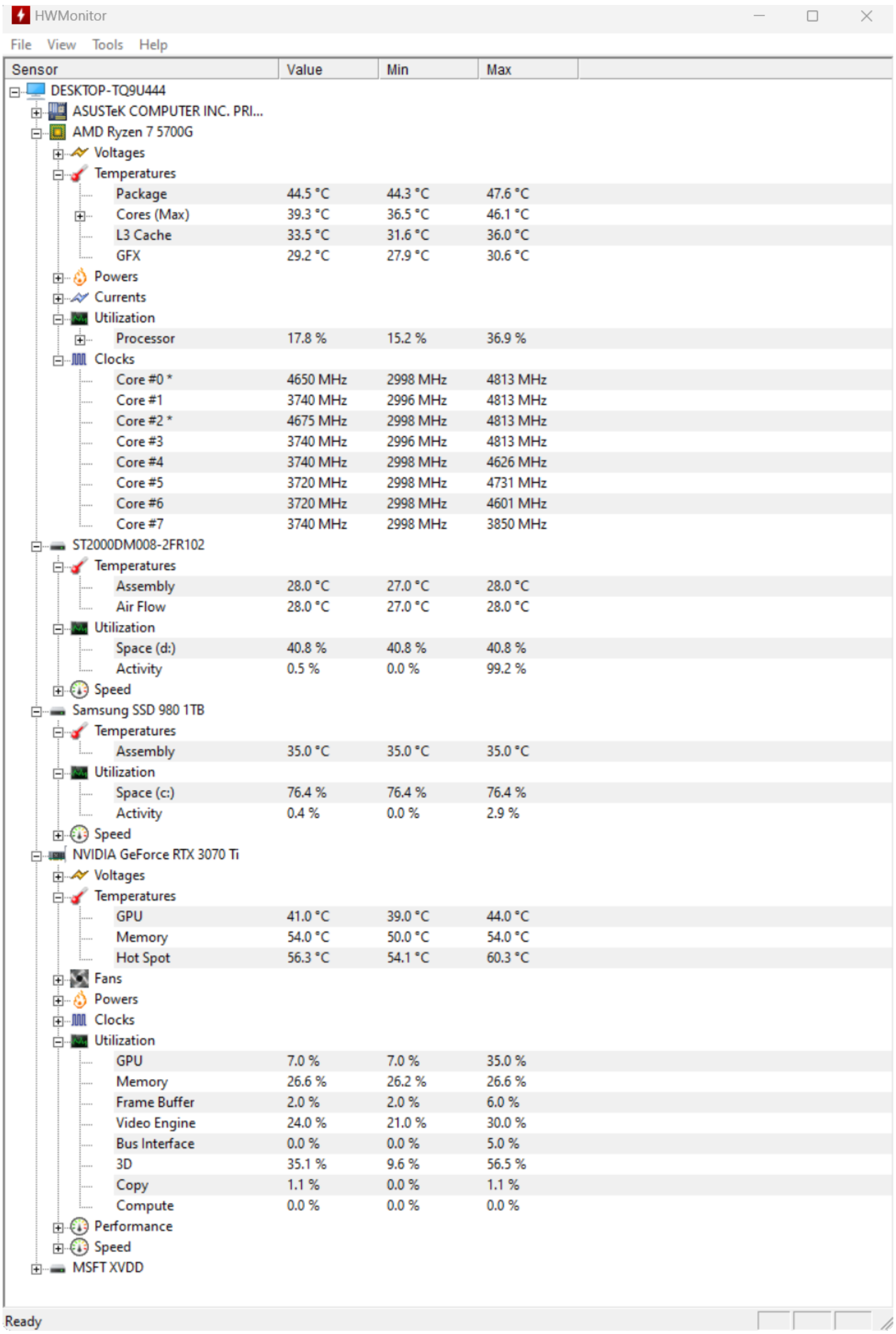
Games are enjoyed by people around the world: in different areas, with different backgrounds, and different financial situations. This means it is common for people who play games to be unable to obtain high performance PC components due to their personal circumstances. For this reason, it is imperative that games be made as accessible as possible to different hardware specifications. Code optimisation and testing is key to ensure this accessibility, and means that a game will be better received by the community as there are fewer limitations preventing people from being unable to play the game or have a smooth, quality experience.

The impact on hardware for each algorithm was tested by using the software "CPUID HWMonitor". This program monitors many metrics for each hardware component in a PC. While running, it displays the lowest value reached, current value, and highest value reached for each metric. It is important to note that the current value would be from when the results were recorded rather than during testing, so are not a fair representation of the algorithms. The tests were completed by resetting the values on HWMonitor, playing with the chosen algorithm for 5 minutes, then recording the values obtained. For these tests, the game was preloaded and ready to play – meaning that loading and compilation did not have any effect on the results. The only other program open in the foreground was "OBS Studio" which was used to record these tests being done.

The results of the tests showed that there were a few differences in performance, but none of the algorithms required the full capabilities of the hardware available. This suggests that the algorithms can be considered lightweight, and to not require high specification hardware to run. However, it is also worth noting that when implemented in a game with many systems, the impact of each system will add up. And so, any chance to optimise performance should be taken, regardless of the amount of performance saved. No values from Bullet Spread stuck out as being noticeably different: nothing high or low; suggesting that using bullet spread has no impact compared to other aim adjustment algorithms. The discernible differences seen with area cursor were: higher CPU clock frequency, and a higher maximum value on 3D utilization of the GPU; suggesting that the calculations done by the CPU were likely completed slightly faster on average, and increasing performance; but the spent more time rendering 3D models. Bullet magnetism reached a 1-2 degrees Celsius higher maximum temperature for the GPU values, and also saw a lot more SSD activity; this suggests that the GPU may have been under greater stress (although this was the final test so it could be due to the sustained stress from all three algorithms), and the SSD was definitely under a greater load.

From these results, it can be seen that none of the tested algorithms have any significant effect on hardware performance. The largest effect was from Bullet Magnetism; and the smallest from Area Cursor. All three should be suitable as lightweight aim adjustment algorithms, but if hardware performance is the priority, Area Cursor may be the best suited. Due to the absence of a control variable in this test, the effect on these algorithms compared to no aim adjustment cannot be determined; however, the very low utilization of hardware components was consistent across algorithms and therefore it can be inferred that they will not have much effect on performance.

These performance differences would likely be more discernible on a lower-end PC, as this testing was completed on a moderately high-end PC (RTX 3070 Ti, Ryzen 7 5700G, 32GB 3600MHz C16 RAM, 1TB Samsung 980, 2TB Seagate Barracuda).



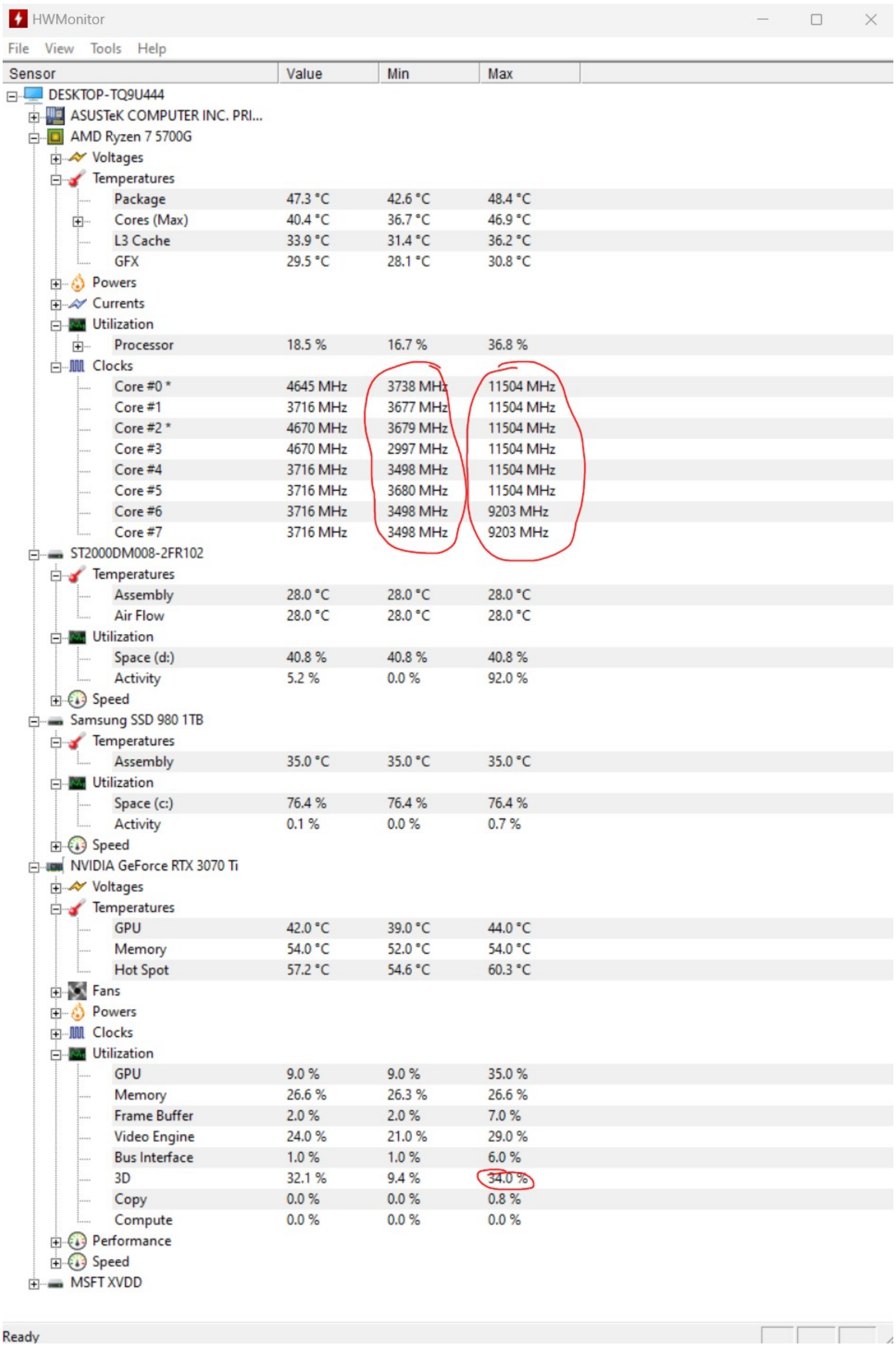
HWMonitor

File View Tools Help

Sensor	Value	Min	Max
DESKTOP-TQ9U444			
ASUSTeK COMPUTER INC. PRI...			
AMD Ryzen 7 5700G			
Voltages			
Temperatures			
Package	44.5 °C	44.3 °C	47.6 °C
Cores (Max)	39.3 °C	36.5 °C	46.1 °C
L3 Cache	33.5 °C	31.6 °C	36.0 °C
GFX	29.2 °C	27.9 °C	30.6 °C
Powers			
Currents			
Utilization			
Processor	17.8 %	15.2 %	36.9 %
Clocks			
Core #0 *	4650 MHz	2998 MHz	4813 MHz
Core #1	3740 MHz	2996 MHz	4813 MHz
Core #2 *	4675 MHz	2998 MHz	4813 MHz
Core #3	3740 MHz	2996 MHz	4813 MHz
Core #4	3740 MHz	2998 MHz	4626 MHz
Core #5	3720 MHz	2998 MHz	4731 MHz
Core #6	3720 MHz	2998 MHz	4601 MHz
Core #7	3740 MHz	2998 MHz	3850 MHz
ST2000DM008-2FR102			
Temperatures			
Assembly	28.0 °C	27.0 °C	28.0 °C
Air Flow	28.0 °C	27.0 °C	28.0 °C
Utilization			
Space (d:)	40.8 %	40.8 %	40.8 %
Activity	0.5 %	0.0 %	99.2 %
Speed			
Samsung SSD 980 1TB			
Temperatures			
Assembly	35.0 °C	35.0 °C	35.0 °C
Utilization			
Space (c:)	76.4 %	76.4 %	76.4 %
Activity	0.4 %	0.0 %	2.9 %
Speed			
NVIDIA GeForce RTX 3070 Ti			
Voltages			
Temperatures			
GPU	41.0 °C	39.0 °C	44.0 °C
Memory	54.0 °C	50.0 °C	54.0 °C
Hot Spot	56.3 °C	54.1 °C	60.3 °C
Fans			
Powers			
Clocks			
Utilization			
GPU	7.0 %	7.0 %	35.0 %
Memory	26.6 %	26.2 %	26.6 %
Frame Buffer	2.0 %	2.0 %	6.0 %
Video Engine	24.0 %	21.0 %	30.0 %
Bus Interface	0.0 %	0.0 %	5.0 %
3D	35.1 %	9.6 %	56.5 %
Copy	1.1 %	0.0 %	1.1 %
Compute	0.0 %	0.0 %	0.0 %
Performance			
Speed			
MSFT XVDD			

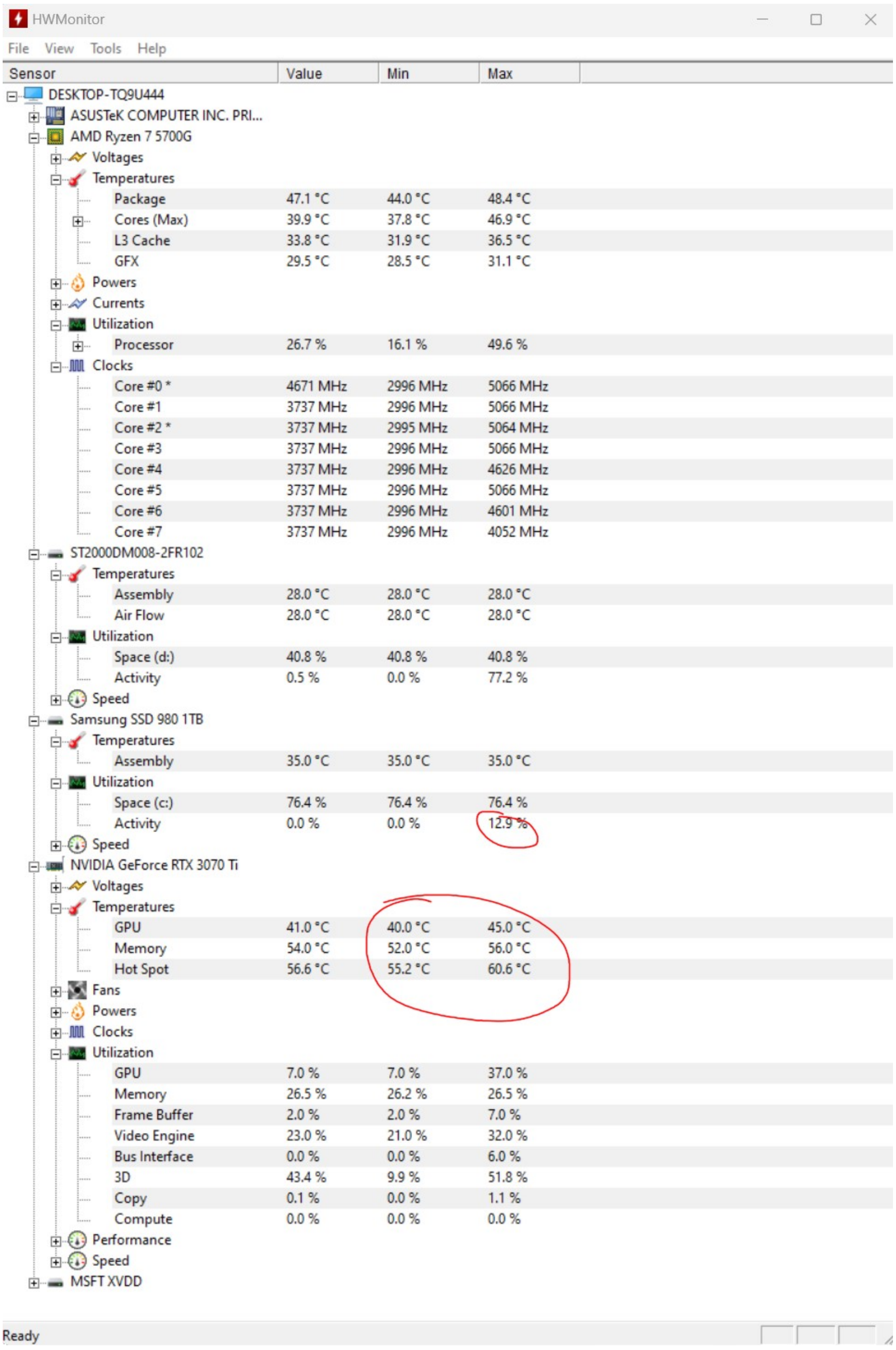
Ready

Figure 12. Bullet Spread Hardware Usage.



Sensor	Value	Min	Max
DESKTOP-TQ9U444			
ASUSTeK COMPUTER INC. PRI...			
AMD Ryzen 7 5700G			
Voltages			
Temperatures			
Package	47.3 °C	42.6 °C	48.4 °C
Cores (Max)	40.4 °C	36.7 °C	46.9 °C
L3 Cache	33.9 °C	31.4 °C	36.2 °C
GFX	29.5 °C	28.1 °C	30.8 °C
Powers			
Currents			
Utilization			
Processor	18.5 %	16.7 %	36.8 %
Clocks			
Core #0 *	4645 MHz	3738 MHz	11504 MHz
Core #1	3716 MHz	3677 MHz	11504 MHz
Core #2 *	4670 MHz	3679 MHz	11504 MHz
Core #3	4670 MHz	2997 MHz	11504 MHz
Core #4	3716 MHz	3498 MHz	11504 MHz
Core #5	3716 MHz	3680 MHz	11504 MHz
Core #6	3716 MHz	3498 MHz	9203 MHz
Core #7	3716 MHz	3498 MHz	9203 MHz
ST2000DM008-2FR102			
Temperatures			
Assembly	28.0 °C	28.0 °C	28.0 °C
Air Flow	28.0 °C	28.0 °C	28.0 °C
Utilization			
Space (d:)	40.8 %	40.8 %	40.8 %
Activity	5.2 %	0.0 %	92.0 %
Speed			
Samsung SSD 980 1TB			
Temperatures			
Assembly	35.0 °C	35.0 °C	35.0 °C
Utilization			
Space (c:)	76.4 %	76.4 %	76.4 %
Activity	0.1 %	0.0 %	0.7 %
Speed			
NVIDIA GeForce RTX 3070 Ti			
Voltages			
Temperatures			
GPU	42.0 °C	39.0 °C	44.0 °C
Memory	54.0 °C	52.0 °C	54.0 °C
Hot Spot	57.2 °C	54.6 °C	60.3 °C
Fans			
Powers			
Clocks			
Utilization			
GPU	9.0 %	9.0 %	35.0 %
Memory	26.6 %	26.3 %	26.6 %
Frame Buffer	2.0 %	2.0 %	7.0 %
Video Engine	24.0 %	21.0 %	29.0 %
Bus Interface	1.0 %	1.0 %	6.0 %
3D	32.1 %	9.4 %	34.0 %
Copy	0.0 %	0.0 %	0.8 %
Compute	0.0 %	0.0 %	0.0 %
Performance			
Speed			
MSFT XVDD			

Figure 13. Area Cursor Hardware Usage.



Sensor	Value	Min	Max
DESKTOP-TQ9U444			
ASUSTeK COMPUTER INC. PRI...			
AMD Ryzen 7 5700G			
Voltages			
Temperatures			
Package	47.1 °C	44.0 °C	48.4 °C
Cores (Max)	39.9 °C	37.8 °C	46.9 °C
L3 Cache	33.8 °C	31.9 °C	36.5 °C
GFX	29.5 °C	28.5 °C	31.1 °C
Powers			
Currents			
Utilization			
Processor	26.7 %	16.1 %	49.6 %
Clocks			
Core #0 *	4671 MHz	2996 MHz	5066 MHz
Core #1	3737 MHz	2996 MHz	5066 MHz
Core #2 *	3737 MHz	2995 MHz	5064 MHz
Core #3	3737 MHz	2996 MHz	5066 MHz
Core #4	3737 MHz	2996 MHz	4626 MHz
Core #5	3737 MHz	2996 MHz	5066 MHz
Core #6	3737 MHz	2996 MHz	4601 MHz
Core #7	3737 MHz	2996 MHz	4052 MHz
ST2000DM008-2FR102			
Temperatures			
Assembly	28.0 °C	28.0 °C	28.0 °C
Air Flow	28.0 °C	28.0 °C	28.0 °C
Utilization			
Space (d:)	40.8 %	40.8 %	40.8 %
Activity	0.5 %	0.0 %	77.2 %
Speed			
Samsung SSD 980 1TB			
Temperatures			
Assembly	35.0 °C	35.0 °C	35.0 °C
Utilization			
Space (c:)	76.4 %	76.4 %	76.4 %
Activity	0.0 %	0.0 %	12.9 %
Speed			
NVIDIA GeForce RTX 3070 Ti			
Voltages			
Temperatures			
GPU	41.0 °C	40.0 °C	45.0 °C
Memory	54.0 °C	52.0 °C	56.0 °C
Hot Spot	56.6 °C	55.2 °C	60.6 °C
Fans			
Powers			
Clocks			
Utilization			
GPU	7.0 %	7.0 %	37.0 %
Memory	26.5 %	26.2 %	26.5 %
Frame Buffer	2.0 %	2.0 %	7.0 %
Video Engine	23.0 %	21.0 %	32.0 %
Bus Interface	0.0 %	0.0 %	6.0 %
3D	43.4 %	9.9 %	51.8 %
Copy	0.1 %	0.0 %	1.1 %
Compute	0.0 %	0.0 %	0.0 %
Performance			
Speed			
MSFT XVDD			

Figure 14. Bullet Magnetism Hardware Usage.

Black Box Playtesting

Black Box testing is a standard software testing method by which a system is tested only from the perspective of a user; without any code analysis. This provides the developer with the knowledge of what the software will actually be like when a user is interacting with it. A user would almost never have any knowledge of the software's code, making this form of testing the best representation for how successful a piece of software is for its desired application.

Playtests are the most important form of testing a game, as it tests the real scenarios that a player will experience, and the functions that they will interact with. The scenarios tested were: short range combat, medium range combat, long range combat, shooting in an open space, shooting indoors, shooting around corners, and shooting through doorways.

Short range combat is a strongpoint for Bullet Magnetism and to some extent Bullet Spread too, it becomes far easier to hit an opponent very near you when the bullet hitbox is larger; and since small movements by the enemy become more difficult to track, bullet spread can randomise your shot, which can sometimes make up for the difficulty tracking the opponent.

In medium range tests, bullet magnetism and area cursor both gave a small degree of assistance to hit enemies without being as accurate, whereas bullet Spread started to become more difficult to hit shots with.

Long ranges were best for area cursor, where the algorithm made up for large inaccuracies when aiming, however bullet magnetism became little use and bullet spread became near impossible to hit enemies with; it is possible that a bullet magnetism algorithm could perform better at medium-long ranges with a slower decrease in hitbox radius.

Shooting around corners was very interesting, it showed that all three algorithms struggled in certain scenarios: bullet spread struggled further away from the corner, area cursor also struggled further away from the corner, and bullet magnetism struggled closer to the corner; this is because bullet spread is largely inaccurate and shooting around a corner requires increased accuracy to not hit the wall, and area cursor and bullet magnetism both have increased hitbox sizes, meaning that the hitbox will hit the wall.

The same effect from shooting around corners could be seen for shooting through doorways, except that these issues became even more noticeable; as the bullet had to avoid a wall from multiple different sides at once.

This testing showed that Bullet Spread is unsuitable for aim assistance at these strengths. A much smaller degree of spread may be more suitable but has not been tested. At the tested strengths, it was shown to make aiming more difficult. The exception to this rule, where Bullet Spread can assist with aiming, is for incredibly close-range combat. These findings are in line with industrial standards, which use Bullet Spread at medium-long ranges to make aiming more difficult, and at use it at short ranges with shotguns to increase ease of use.

4.3 Discussion

	Bullet Spread	Area Cursor	Bullet Magnetism
Pros	• Helps at close range • Simple to implement • Lightweight	• Easier to aim • Helps at mid-long range • Simple to implement • Best player experience • Very lightweight Easy to test	• Easier to aim • Helps at short-mid range • Simple to implement • Good player experience • Lightweight • Easy to test • Not too easy, still some challenge
Cons	• More difficult to aim • Bad at anything other than short range • Bad player experience • Difficult to test (due to randomness) • Can interfere with corners and walls • Inconsistent	• Interferes with corners and walls • Can become too easy if not configured carefully	• Interferes with corners and walls

Figure 15. Tabulated GEQ values for each algorithm.

Overall Bullet Spread is unsuitable for a general aim assistance algorithm. In the case of a close-range weapon, such as a shotgun, Bullet Spread could be a good solution. But for any other situations, Bullet Spread is unfit for this purpose. Adding randomness to affect what the player has done seems to give a negative experience.

Area Cursor shows the most promise as an aim assistance algorithm, it works well at all ranges except for very close-quarters fights, and has very few downsides. It provides a safety net to improve in game performance of a player, and overall improve experience.

Bullet Magnetism shows the most promise at close-mid ranges. It is more consistent than bullet spread at close range (due to bullet spread's randomness), however could be too powerful at close ranges if not carefully adjusted to suit the gameplay. It also has an increased level of challenge over Area Cursor, meaning that more skill is required to hit enemies at longer ranges, in line with what players often believe long range combat should be like.

5 Conclusion, Recommendations & Future Research

5.1 Conclusion

From this project, a game has been created which implements three of the top existing aim adjustment algorithms. These algorithms have been compared for their suitability to aim assistance purposes in FPS games. Overall, the project was successful, meeting the aim set out at the start of the project: an FPS game has been created, and has been studied to produce a critical evaluation and comparison of common aim assistance technologies.

Requirement	**Description**	**Met?**	**Reference**
O1	Conduct a comprehensive review of aim assistance techniques.	True	Section 4
O2	Identify suitable evaluation parameters for comparison.	True	Sections 2, 4
O3	Develop artefacts based on the selected aim assistance technologies.	True	Section 3
O4	Analyse the performance of the artefacts in O3 (using parameters identified in O2).	True	Section 4
O5	Comparatively summarize the developed artefacts.	True	Section 4
O6	Propose solutions. (extension)	True	Section 4 (below)
O7	Create FPS game.	True	Section 3
T1	Identification and detailing of aim assistance technologies.	True	Section 2
T2	Demonstration of working artefacts.	True	Sections 3, 4
T3	Described proposed new/improved solutions. (extension)	True	Section 4 (below)
T4	Demonstration of functional artefacts of proposed solutions. (extension)	False	
T5	Demonstration of working FPS.	True	Sections 3, 4

Figure 16. Tabulated project objectives and targets with a decision for whether each was met.

From Figure 16, it can be seen that all objectives and targets set out were completed successfully, with the exception of T4. T4 was an extension task, as it was beyond the scope of the project but would be beneficial to the project. Nonetheless, the other extension tasks have been completed along with the required specifications. This means that the project can be considered a success, as the specifications have been successfully met. But there is still room for improvement and further research to be done.

From existing research, it was identified that technologies that affect the user's control of their crosshair, such as "sticky targets" are unsuitable for aim assistance, as they are frustrating and detrimental to flow. Through testing, it has been identified that all three algorithms used (Bullet Spread, Area Cursor, Bullet Magnetism) are suitable for aim assistance in certain situations. Area Cursor is by far the most suitable for general gameplay, for close range combat Bullet Spread and Bullet Magnetism both show promise but start to fall apart at mid-long ranges (especially true of Bullet Spread). However, all three have a large problem when interacting with walls which must be carefully avoided. Some standard implementations of Area Cursor may be able to avoid this problem without adjustment, but Bullet Spread and Bullet Magnetism would likely need to be carefully designed to avoid this issue.

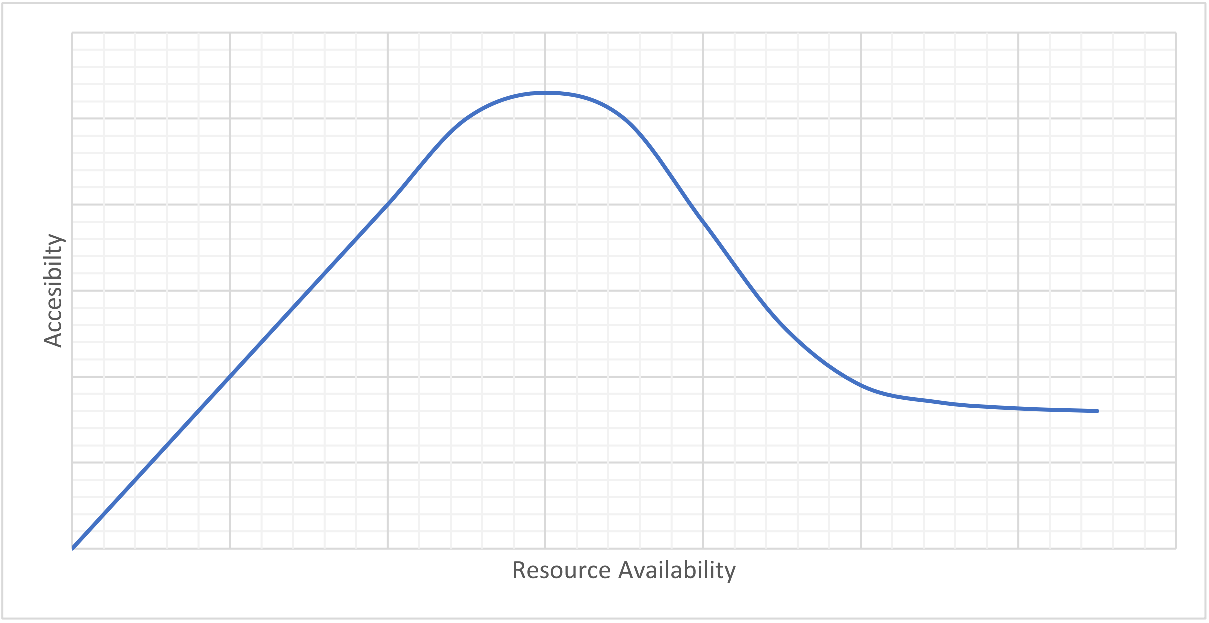


Figure 17. Graphical representation of how number of resources influences ease of use of a tool.

Due to the availability of resources, it was anticipated that finding solutions to problems and implementing standard technologies such as version control would be easily accessible. Through this project it was found that the abundance of Unity resources could be very overwhelming and lead to many dead-ends with much outdated solutions recommended online. This suggests that while high availability of resources is good in principle, in practice it can become detrimental if there are too many resources. As imagined in figure 17, as the availability of resources increases, the accessibility increases steadily, until it plateaus and then falls. It is depicted as falling with a decreasing rate at the end due to some solutions being suitable and therefore the accessibility should never reach 0.

5.2 Recommendations

Unfortunately, the interaction with walls could not be solved by programming the bullets' hitboxes to ignore walls, as this would cause bullets to always go through walls and therefore give the ability to shoot invisible enemies in other rooms. A potential solution for this would be to have two h itboxes: one that changes size, and one that remains constant, this would mean that if the bullet itself did not hit the wall but the changing hitbox did, this could be detected and the regular hitbox could continue travelling while the changing hitbox is destroyed. It would be important that the changing hitbox that hit the wall is destroyed otherwise the player may become able to shoot enemies through walls. While they were not identified and may be difficult to find, it is likely that some games in industry will already have solutions for this problem. However, this is just an assumption and may not be the case.

For games development, the recommended algorithm for general use aim assistance is area cursor. However, for use in short range weapons, such as shotguns, the recommended algorithm is bullet magnetism. Bullet Spread has been proven to be unsuitable for aim assist, and should instead be used only to make aiming more difficult, as is often already standard.

For future research, it should be tested to see if movement speed of a player affects the need for aim assistance. Different values for aim assist strength should also be tested to find the ideal strength for player experience. An interesting idea to test could be a system that adjusts each players aim assistance differently based on their skill level for balancing purposes. This could reduce the need for skill-based matchmaking (SBMM) and mean that connection-based matchmaking (CBMM) can be used, giving better flow and smoothness to the game.

It is important, with any software development, to consider the ethics of a system. When designing an aim assistance algorithm, the many ethical aspects of the system must be explored. A strong emphasis should be on the social aspect, as these systems will have a significant effect on players and, in PvP scenarios, also have a sizeable impact on the interactions between these individuals.

One learning which is relevant to many different fields, not just software or game development, is the negative effect that can result from too many resources to use for learning and problem solving. It would be beneficial for many industries and individuals to study the effect this can have. While it is important to support old software, it can be seen that companies should have a responsibility to make it clear when a resource is outdated, and warnings should be displayed when potentially outdated resources are shown online.

6 Project Management Review

6.1 Original Project Work Description

Literature Review
I will research existing literature to learn about the field my project is based on, and the themes surrounding it. This is vital so that I can properly understand the requirements of standardised solutions.

Programming
Artefacts 1-4:
I will produce several functional artefacts, each presenting a different existing algorithm. This will allow me to understand the solution in its entirety, and be more capable of analysing its benefits and shortcomings.

Proposed Solution:
I will develop one functional artefact of my own solution (if suitable) to the problem. This should demonstrate that better algorithms could be developed.

Artefact Analysis
I will analyse each artefact I program. This is key to breaking down the problem and seeing what is good and bad about each existing solution.

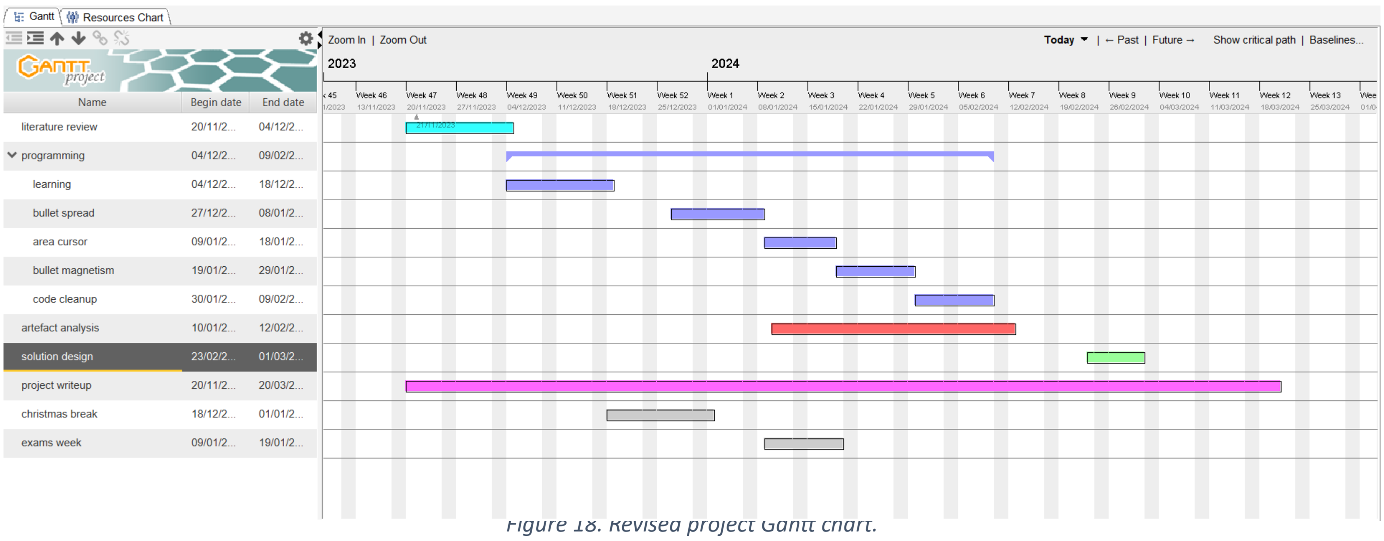
Solution Design
I will design my own proposition for a solution. This solution should learn from the existing solutions, and give a better/alternate solution to the problem.

Solution Analysis
I will analyse my proposed solution, highlighting its strengths and weaknesses (especially in comparison to existing solutions), and determine whether I think it is well suited to solving the problem.

Project Writeup
I will write an outline of what I have done, what I have found, and what I have concluded based on these findings.

I will also be having weekly meetings face to face with my supervisor to ensure that suitable progress is being made and the project is on track. After these meetings I will submit a logbook on canvas with notes of what was discussed during the meetings.

6.2 Revised Project Work Description



Name	Begin date	End date
literature review	20/11/2...	04/12/2...
programming	04/12/2...	09/02/2...
learning	04/12/2...	18/12/2...
bullet spread	27/12/2...	08/01/2...
area cursor	09/01/2...	18/01/2...
bullet magnetism	19/01/2...	29/01/2...
code cleanup	30/01/2...	09/02/2...
artefact analysis	10/01/2...	12/02/2...
solution design	23/02/2...	01/03/2...
project writeup	20/11/2...	20/03/2...
christmas break	18/12/2...	01/01/2...
exams week	09/01/2...	19/01/2...

Figure 18. Revised project Gantt chart.

It can be seen from figure 18 and appendix A that there were some clear changes made to the project pipeline through revisions. Rather than being split into 4 sub-sections, programming was split into 5 sub-sections representing each programming task. It was realized early on that the programming could not commence immediately, as it would take time getting used to the open-source project and learning how it works. By dedicating time to this, it made each individual programming task easier and more accomplishable, due to the prior knowledge of existing systems. However, the programming for each algorithm still took longer than originally planned, and so more time was allocated to each programming task. There was not sufficient time to produce an alternative solution, which was set as a challenge task; this was left out due to being a lower priority than other artefacts. This was removed from the programming and analysis, however its design was still considered and briefly discussed in this report. The time given for doing this was much shorter than previously planned, as there was no longer a need to design a solution in depth ready for programming; only a simple concept was provided. The artefact analysis was simpler to do when each artefact was fresh in my mind, and so after each artefact was completed, it was analysed simultaneously with the programming for the next artefact. Once all artefacts were completed, time was then allocated to do cleanup on the code, making it more readable, simpler to understand, and efficient.

As depicted in appendix A, there were many strengths and weaknesses to the project management. A suitable level of work with a suitable degree of challenge was successfully selected for this project. In future projects, I will make sure that I provide sufficient time for learning the environment and any libraries required. I will also attempt to check libraries for potential compatibility issues before implementation, to save time fixing errors.

	Strengths	Weaknesses
Time and Workload	• An appropriate workload was identified for the scope of the project • Project completed on time	• It was underestimated how long programming would take • The time taken for learning the environment and existing systems were not planned for • Challenge/extension tasks could not be completed
Quality of work	• A high-quality report was created • Artefacts are fully functional	• 2/3 artefacts were representations rather than authentic copies of existing algorithms
Schedules and Meetings	• Planned what to talk about in advance • Emailed with information ahead of time if necessary • Gantt chart helped to keep project work on track	• Due to performance issues, artefacts could not be demonstrated to a high quality in meetings • Gantt chart missed out a few stages • Gantt chart needed some revision (to be expected with any software development)
Communication	• Well explained and thought-out report • High understanding of existing literature • Timely communications with supervisor via email • Regular meetings with supervisor at assigned time each week	• Could have had longer meetings running supervisor through all work rather than a quick look-over
Resources	• Open-source libraries were found and used well • Online resources were used to find solutions to problems and bugs • Took suggestions and guidance from supervisor as needed	• Found too many online resources overwhelming • Trying to use libraries for software meant a higher learning curve and compatibility issues
Documentation	• Well documented project • Good analysis and understanding of existing literature • Frequent use of official software documentation	• Strict format for project means less useable for researchers or professionals – more to filter through

Figure 19. Tabular analysis of project management.

References

Ahmed, M. Reno, S. Rahman, Md R. and Rifat, H S. (2022) *Analysis of Netcode, Latency, and Packet-loss in Online Multiplayer Games*. Trichy, India: Institute of Electrical and Electronics Engineers.

Ambler, A L. Burnett, M M. Zimmerman, B A. (1992) *Operational Versus Definitional: A Perspective on Programming Paradigms.* Washington, USA: IEEE Computer Society Press.

Boudaoud, B. Spjut, J. and Kim, J. (2022) *Mouse Sensitivity In First-person Targeting Tasks.* USA: NVidia.

Drewsky. (2023) *drewskyschannel*. YouTube. Available at www.youtube.com/@drewskyschannel (Accessed: 11/2023)

Epic Games. (2024) *What's new in Unreal Engine 5.3.* North Carolina, USA: Epic Games.

Fully Loaded. (2022) *Lead your shot - Fortnite Sniper Headshot*. YouTube. Available at https://youtu.be/2iUAnCXTGGQ (Accessed: 02/2024)

Gerling, K M. Klauser, M. and Niesenhaus, J. (2011) *Measuring the impact of game controllers on player experience in FPS games*. New York, USA: Association for Computing Machinery.

Gutwin, C. Vicencio-Moreira, R. Mandryk, R L. (2016) *Does Helping Hurt?: Aiming Assistance and Skill Development in a First-Person Shooter Game*. New York, USA: Association for Computing Machinery.

IJsselsteijn, W A. de Kort, Y A W. and Poels, K. (2013) *The Game Experience Questionnaire*. Eindhoven, Netherlands: Technische Universiteit Eindhoven.

Ivkovic, Z. Stavness, I. Gutwin, C. and Sutcliffe S. (2015) *Quantifying and Mitigating the Negative Effects of Local Latencies on Aiming in 3D Shooter Games*. New York, USA: Association for Computing Machinery.

Knutsen, Í K. (2021) *Visual Scripting in Game Development*. Espoo, Finland: Metropolia University of Applied Sciences.

Lee, W K. and Chang, R K C. (2015) *Evaluation of lag-related configurations in first-person shooter games.* Zagreb, Croatia: NetGames.

Looser, J. Cockburn, A. and Savage, J. (2012) *On the Validity of Using First-Person Shooters for Fitts' Law Studies Human-Computer Interaction Lab Department of Computer Science and Software Engineering* Christchurch, New Zealand: University of Canterbury.

Prechelt, L. (2000) *An empirical comparison Of C, C++, Java, Perl, Python, Rexx, and Tcl for a search/string-processing program*. Karlsruhe, Germany: Fakultat fur Informatik Universitat Karlsruhe.

Ryan, C. (2014) *Bullet Magnetism in Halo 1*. YouTube. Available at https://youtu.be/fRg2DwZlJTo (Accessed: 02/2024)

Šmíd, A. (2017) *Comparison of Unity and Unreal Engine*. Prague, Czech Republic: Czech Technical University in Prague.

Unity. (2018) *FPS Sample*. Unity Technologies. Available at https://Unity.com/FPS-sample (Accessed: 02/2024)

Unity. (2023) *FPS Microgame*. Unity Technologies. Available at https://learn.Unity.com/project/FPS-template (Accessed: 02/2024)

Vicencio-Moreira, R. Gutwin, C. Mandryk, R L. (2015) *Now You Can Compete With Anyone: Balancing Players of Different Skill Levels in a First-Person Shooter Game*. Saskatoon, Canada: University of Saskatchewan.

Vicencio-Moreira, R. Mandryk, R L. Gutwin, C. and Bateman C S. (2014) *The effectiveness (or lack thereof) of aim-assist techniques in first-person shooter games*. New York, USA: Association for Computing Machinery.

Vohera, C. Chheda, H. Chouhan, D. Desai, A. and Jain, V. (2021) *Game Engine Architecture and Comparative Study of Different Game Engines.* Kharagpur, India: Institute of Electrical and Electronics Engineers.

VRChat. (2024) *The VRChat Documentation Hub.* San Francisco, USA: VRChat.

Willet, S. and Hogan, M. (2019) *Cheating the Network: How Gamers Play the Infrastructure*. Calgary, Canada: University of Calgary.

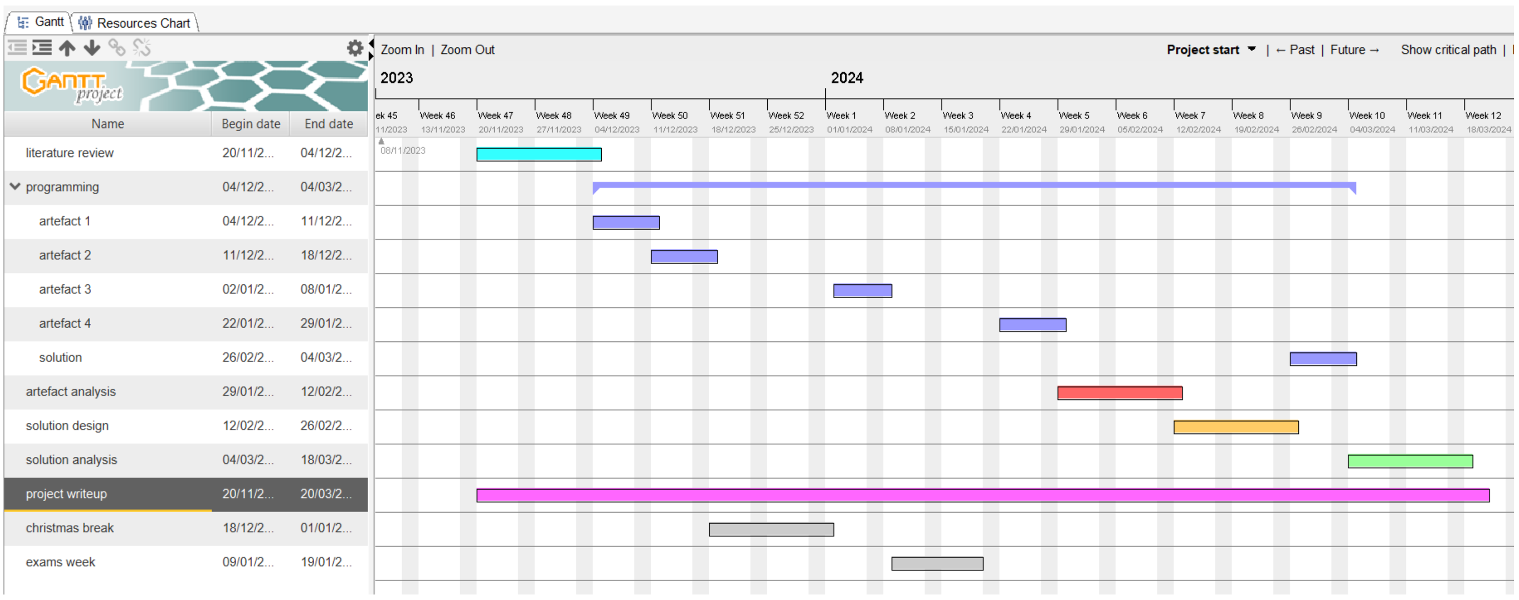
Bibliography

Davison, E. (2020) *Aim assist in the crosshairs*. Washington, USA: The Washington Post.

Fallout Plays. (2019) *FPS Games: What is BLOOM?* YouTube. Available at https://youtu.be/yQltm9nctNk (Accessed: 03/2024)

tripleWRECK. (2016) *GHOST BULLETS TESTED!* YouTube. Available at https://youtu.be/bZ24eDTg_s4 (Accessed: 03/2024)

Appendix A: Gantt Chart



Name	Begin date	End date
literature review	20/11/2...	04/12/2...
programming	04/12/2...	04/03/2...
artefact 1	04/12/2...	11/12/2...
artefact 2	11/12/2...	18/12/2...
artefact 3	02/01/2...	08/01/2...
artefact 4	22/01/2...	29/01/2...
solution	26/02/2...	04/03/2...
artefact analysis	29/01/2...	12/02/2...
solution design	12/02/2...	26/02/2...
solution analysis	04/03/2...	18/03/2...
project writeup	20/11/2...	20/03/2...
christmas break	18/12/2...	01/01/2...
exams week	09/01/2...	19/01/2...

Appendix B: Project Resources Checklist

Student's Name:	**CAMERON MCGRATH-JOHNSTON**
SRN:	**20035374**
Project Title:	**FPS GAME DEVELOPMENT; AIM ASSISTANCE ALGORITHMS.**
Supervisor's Name:	**YONGJUN ZHENG**
Date:	**13/11/2023**

1	Will a specialist laboratory PC be required for the project? If YES, please specify below:	~~YES~~	NO
2	If the answer to 1 is Yes, will you require any special hardware installed/attached? If YES, please specify below:	~~YES~~	NO
3	Do you require access to specialist systems and/or hardware (Database Servers, Network Hardware, Robotics, etc)? If YES, please specify below:	~~YES~~	NO
4	Will you require specific software? If YES, please specify below: I already have the software required (Unity IDE)	YES	~~NO~~
5	Are there other resources required, not covered above? If YES, please specify below:	~~YES~~	NO

Appendix C: Ethics Checklist

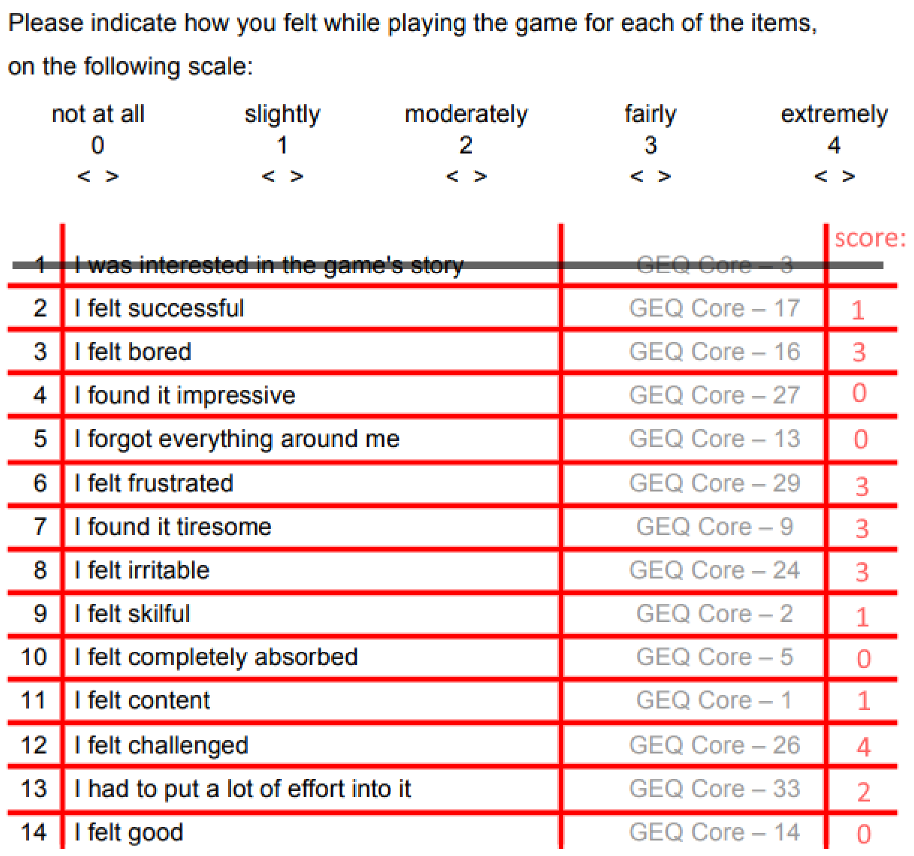
Ethical Issues			
1	Will your work involve human subjects, e.g., survey, interview, experiment, etc. This includes friends and family.	~~YES~~	[✓ NO]

Appendix D: Risk Register

Student's Name:	**CAMERON MCGRATH-JOHNSTON**
SRN:	**20035374**
Project Title:	**FPS GAME DEVELOPMENT; AIM ASSISTANCE ALGORITHMS.**
Supervisor's Name:	**YONGJUN ZHENG**
Date:	**13/11/2023**

Task	Associated Risk	Mitigation Process	Likelihood	Severity
Programming	Limited support available at the university	Selected development environment with the most support/resources available online	high	low
Programming	Game engine/base game is non-functional/ unsuited to project	Researching alternative game engines and game projects to base off; and limiting use of libraries to reduce dependencies	low	high
Comparison	No suitable parameters available for a reliable comparison	Research to find existing measures of similar algorithms	low	high
Demonstrating the working artefacts	Artefacts are non-functional	Multiple different artefacts proposed, libraries will be researched before use	low	low
Failure to identify potential improvements	Inability to propose new solution	Reproducing existing solutions will give a better understanding, and enable a comparison even if I can't design a new algorithm	medium	low

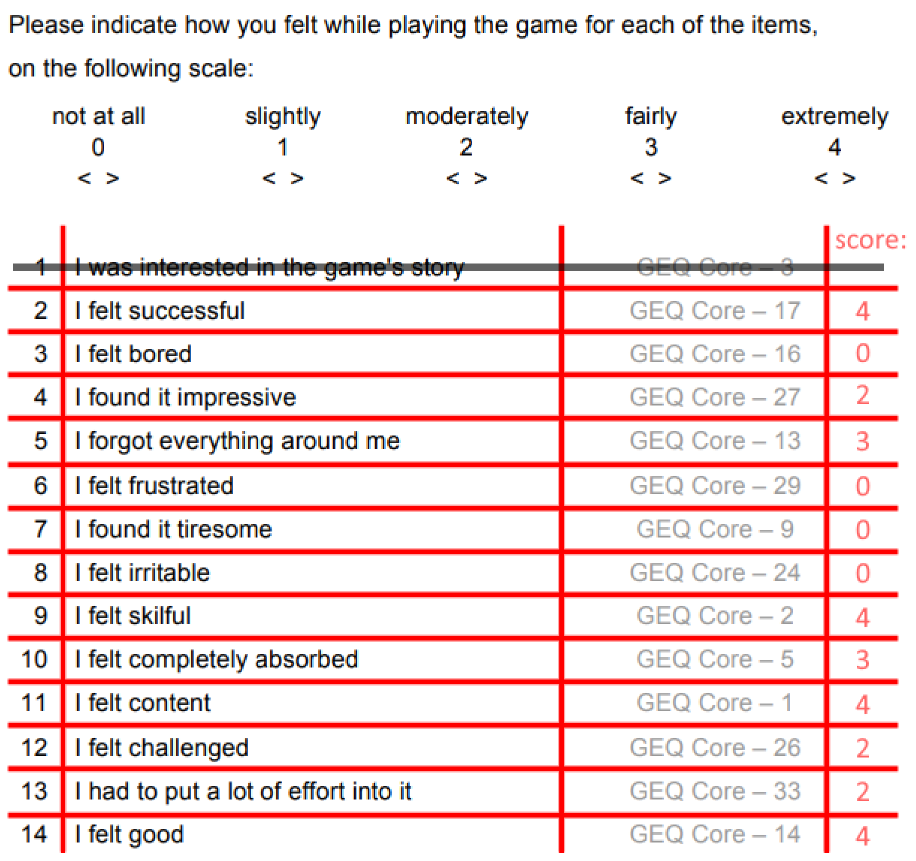
Appendix E: Bullet Spread in-game GEQ


Please indicate how you felt while playing the game for each of the items, on the following scale:

not at all	slightly	moderately	fairly	extremely
0	1	2	3	4
< >	< >	< >	< >	< >

			score:
~~1~~	~~I was interested in the game's story~~	~~GEQ Core – 3~~	
2	I felt successful	GEQ Core – 17	1
3	I felt bored	GEQ Core – 16	3
4	I found it impressive	GEQ Core – 27	0
5	I forgot everything around me	GEQ Core – 13	0
6	I felt frustrated	GEQ Core – 29	3
7	I found it tiresome	GEQ Core – 9	3
8	I felt irritable	GEQ Core – 24	3
9	I felt skilful	GEQ Core – 2	1
10	I felt completely absorbed	GEQ Core – 5	0
11	I felt content	GEQ Core – 1	1
12	I felt challenged	GEQ Core – 26	4
13	I had to put a lot of effort into it	GEQ Core – 33	2
14	I felt good	GEQ Core – 14	0

Appendix F: Area Cursor in-game GEQ

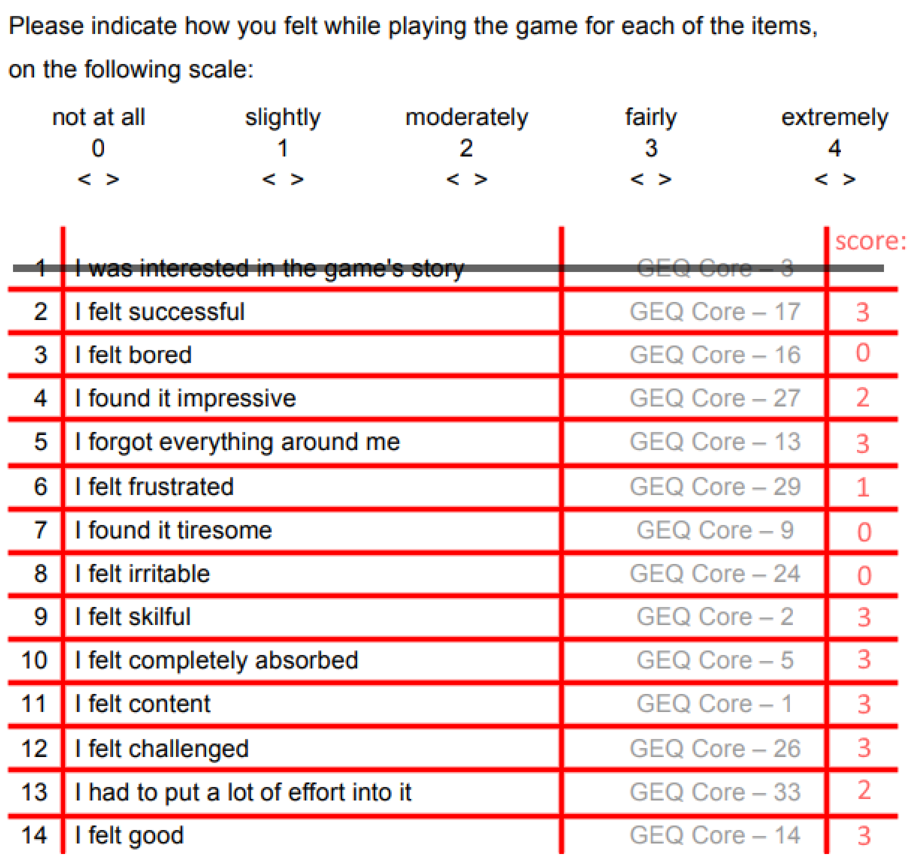


Please indicate how you felt while playing the game for each of the items, on the following scale:

not at all	slightly	moderately	fairly	extremely
0	1	2	3	4
< >	< >	< >	< >	< >

			score:
~~1~~	~~I was interested in the game's story~~	~~GEQ Core – 3~~	
2	I felt successful	GEQ Core – 17	4
3	I felt bored	GEQ Core – 16	0
4	I found it impressive	GEQ Core – 27	2
5	I forgot everything around me	GEQ Core – 13	3
6	I felt frustrated	GEQ Core – 29	0
7	I found it tiresome	GEQ Core – 9	0
8	I felt irritable	GEQ Core – 24	0
9	I felt skilful	GEQ Core – 2	4
10	I felt completely absorbed	GEQ Core – 5	3
11	I felt content	GEQ Core – 1	4
12	I felt challenged	GEQ Core – 26	2
13	I had to put a lot of effort into it	GEQ Core – 33	2
14	I felt good	GEQ Core – 14	4

Appendix G: Bullet Magnetism in-game GEQ

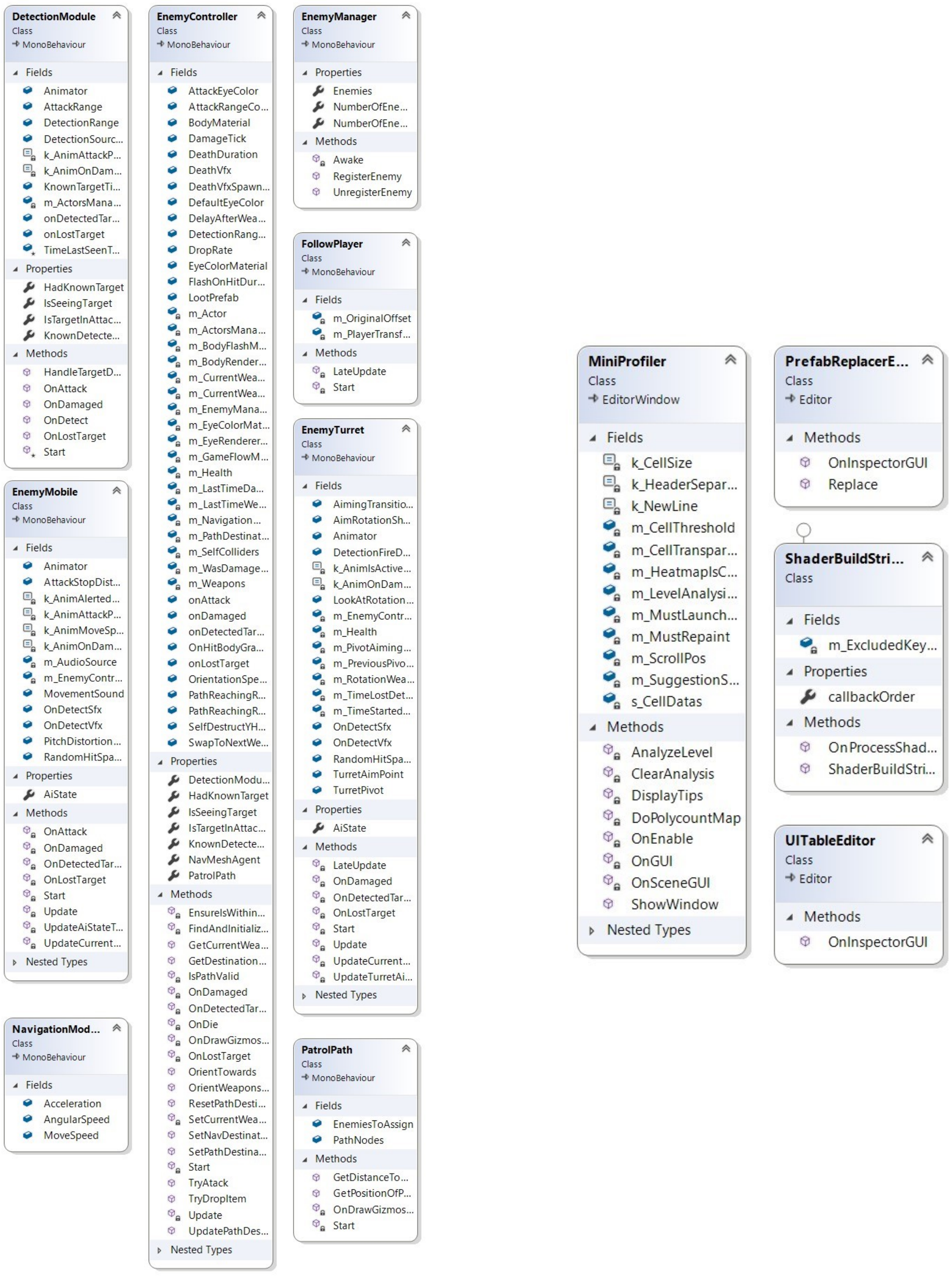


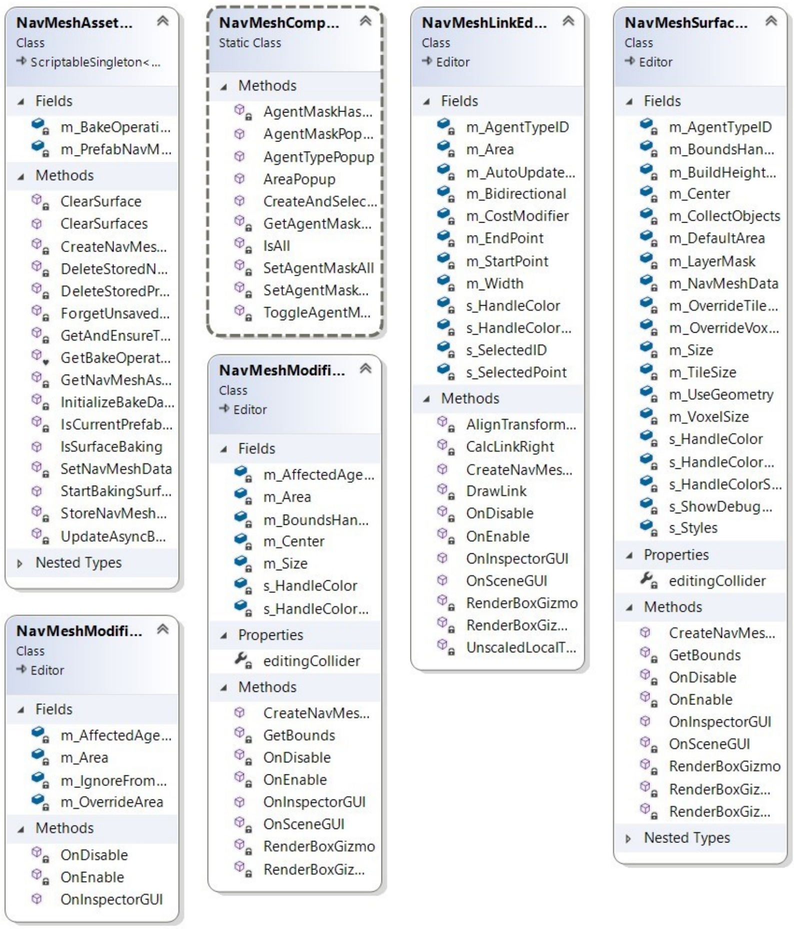
Please indicate how you felt while playing the game for each of the items,
on the following scale:

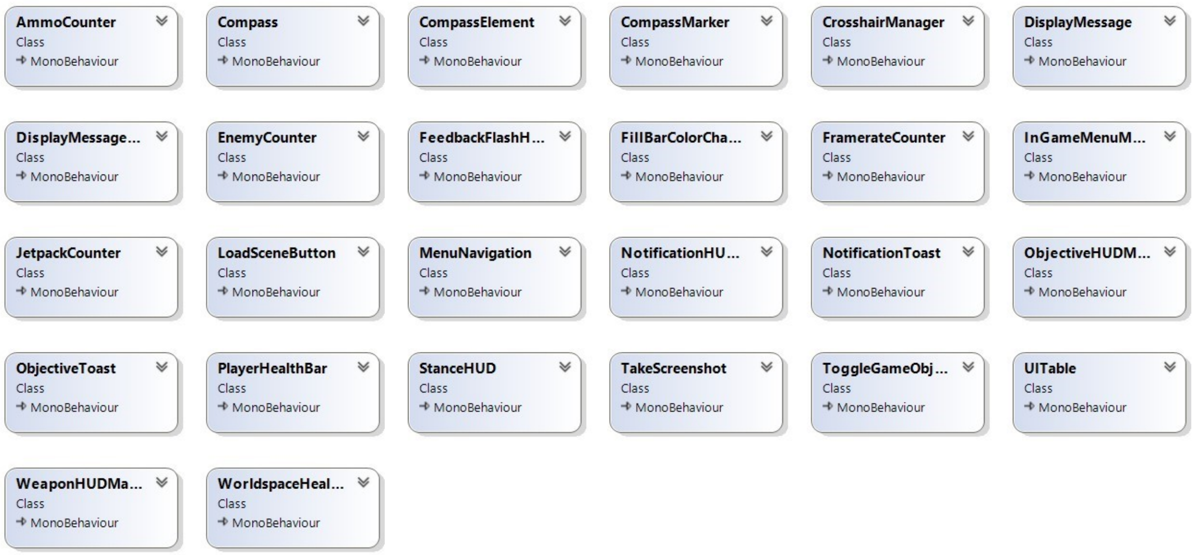
not at all	slightly	moderately	fairly	extremely
0	1	2	3	4
< >	< >	< >	< >	< >

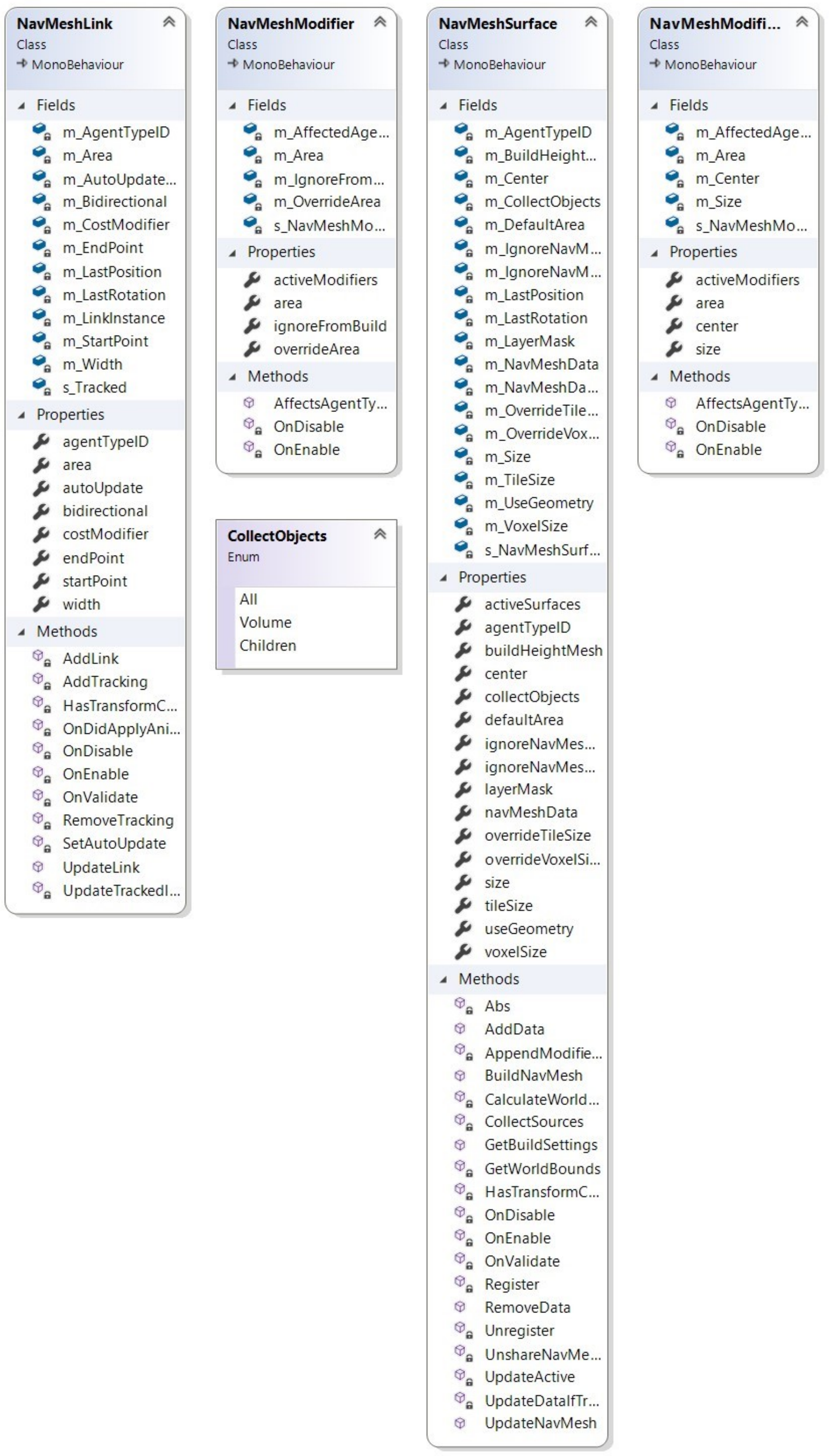
			score:
~~1~~	~~I was interested in the game's story~~	~~GEQ Core – 3~~	
2	I felt successful	GEQ Core – 17	3
3	I felt bored	GEQ Core – 16	0
4	I found it impressive	GEQ Core – 27	2
5	I forgot everything around me	GEQ Core – 13	3
6	I felt frustrated	GEQ Core – 29	1
7	I found it tiresome	GEQ Core – 9	0
8	I felt irritable	GEQ Core – 24	0
9	I felt skilful	GEQ Core – 2	3
10	I felt completely absorbed	GEQ Core – 5	3
11	I felt content	GEQ Core – 1	3
12	I felt challenged	GEQ Core – 26	3
13	I had to put a lot of effort into it	GEQ Core – 33	2
14	I felt good	GEQ Core – 14	3

Appendix H: Game UML Class Diagrams




NavMeshAsset...
Class
ScriptableSingleton<...
Fields
m_BakeOperati...
m_PrefabNavM...
Methods
ClearSurface
ClearSurfaces
CreateNavMes...
DeleteStoredN...
DeleteStoredPr...
ForgetUnsaved...
GetAndEnsureT...
GetBakeOperat...
GetNavMeshAs...
InitializeBakeDa...
IsCurrentPrefab...
IsSurfaceBaking
SetNavMeshData
StartBakingSurf...
StoreNavMesh...
UpdateAsyncB...
Nested Types
NavMeshModifi...
Class
Editor
Fields
m_AffectedAge...
m_Area
m_IgnoreFrom...
m_OverrideArea
Methods
OnDisable
OnEnable
OnInspectorGUI
NavMeshComp...
Static Class
Methods
AgentMaskHas...
AgentMaskPop...
AgentTypePopup
AreaPopup
CreateAndSelec...
GetAgentMask...
IsAll
SetAgentMaskAll
SetAgentMask...
ToggleAgentM...
NavMeshModifi...
Class
Editor
Fields
m_AffectedAge...
m_Area
m_BoundsHan...
m_Center
m_Size
s_HandleColor
s_HandleColor...
Properties
editingCollider
Methods
CreateNavMes...
GetBounds
OnDisable
OnEnable
OnInspectorGUI
OnSceneGUI
RenderBoxGizmo
RenderBoxGiz...
NavMeshLinkEd...
Class
Editor
Fields
m_AgentTypeID
m_Area
m_AutoUpdate...
m_Bidirectional
m_CostModifier
m_EndPoint
m_StartPoint
m_Width
s_HandleColor
s_HandleColor...
s_SelectedID
s_SelectedPoint
Methods
AlignTransform...
CalcLinkRight
CreateNavMes...
DrawLink
OnDisable
OnEnable
OnInspectorGUI
OnSceneGUI
RenderBoxGizmo
RenderBoxGiz...
UnscaledLocalT...
NavMeshSurfac...
Class
Editor
Fields
m_AgentTypeID
m_BoundsHan...
m_BuildHeight...
m_Center
m_CollectObjects
m_DefaultArea
m_LayerMask
m_NavMeshData
m_OverrideTile...
m_OverrideVox...
m_Size
m_TileSize
m_UseGeometry
m_VoxelSize
s_HandleColor
s_HandleColor...
s_HandleColorS...
s_ShowDebug...
s_Styles
Properties
editingCollider
Methods
CreateNavMes...
GetBounds
OnDisable
OnEnable
OnInspectorGUI
OnSceneGUI
RenderBoxGizmo
RenderBoxGiz...
RenderBoxGiz...
Nested Types


AmmoCounter
Class
MonoBehaviour
Compass
Class
MonoBehaviour
CompassElement
Class
MonoBehaviour
CompassMarker
Class
MonoBehaviour
CrosshairManager
Class
MonoBehaviour
DisplayMessage
Class
MonoBehaviour
DisplayMessage...
Class
MonoBehaviour
EnemyCounter
Class
MonoBehaviour
FeedbackFlashH...
Class
MonoBehaviour
FillBarColorCha...
Class
MonoBehaviour
FramerateCounter
Class
MonoBehaviour
InGameMenuM...
Class
MonoBehaviour
JetpackCounter
Class
MonoBehaviour
LoadSceneButton
Class
MonoBehaviour
MenuNavigation
Class
MonoBehaviour
NotificationHU...
Class
MonoBehaviour
NotificationToast
Class
MonoBehaviour
ObjectiveHUDM...
Class
MonoBehaviour
ObjectiveToast
Class
MonoBehaviour
PlayerHealthBar
Class
MonoBehaviour
StanceHUD
Class
MonoBehaviour
TakeScreenshot
Class
MonoBehaviour
ToggleGameObj...
Class
MonoBehaviour
UITable
Class
MonoBehaviour
WeaponHUDMa...
Class
MonoBehaviour
WorldspaceHeal...
Class
MonoBehaviour


NavMeshLink
Class
MonoBehaviour
Fields
m_AgentTypeID
m_Area
m_AutoUpdate...
m_Bidirectional
m_CostModifier
m_EndPoint
m_LastPosition
m_LastRotation
m_LinkInstance
m_StartPoint
m_Width
s_Tracked
Properties
agentTypeID
area
autoUpdate
bidirectional
costModifier
endPoint
startPoint
width
Methods
AddLink
AddTracking
HasTransformC...
OnDidApplyAni...
OnDisable
OnEnable
OnValidate
RemoveTracking
SetAutoUpdate
UpdateLink
UpdateTrackedI...
NavMeshModifier
Class
MonoBehaviour
Fields
m_AffectedAge...
m_Area
m_IgnoreFrom...
m_OverrideArea
s_NavMeshMo...
Properties
activeModifiers
area
ignoreFromBuild
overrideArea
Methods
AffectsAgentTy...
OnDisable
OnEnable
CollectObjects
Enum
All
Volume
Children
NavMeshSurface
Class
MonoBehaviour
Fields
m_AgentTypeID
m_BuildHeight...
m_Center
m_CollectObjects
m_DefaultArea
m_IgnoreNavM...
m_IgnoreNavM...
m_LastPosition
m_LastRotation
m_LayerMask
m_NavMeshData
m_NavMeshDa...
m_OverrideTile...
m_OverrideVox...
m_Size
m_TileSize
m_UseGeometry
m_VoxelSize
s_NavMeshSurf...
Properties
activeSurfaces
agentTypeID
buildHeightMesh
center
collectObjects
defaultArea
ignoreNavMes...
ignoreNavMes...
layerMask
navMeshData
overrideTileSize
overrideVoxelSi...
size
tileSize
useGeometry
voxelSize
Methods
Abs
AddData
AppendModifie...
BuildNavMesh
CalculateWorld...
CollectSources
GetBuildSettings
GetWorldBounds
HasTransformC...
OnDisable
OnEnable
OnValidate
Register
RemoveData
Unregister
UnshareNavMe...
UpdateActive
UpdateDataIfTr...
UpdateNavMesh
NavMeshModifi...
Class
MonoBehaviour
Fields
m_AffectedAge...
m_Area
m_Center
m_Size
s_NavMeshMo...
Properties
activeModifiers
area
center
size
Methods
AffectsAgentTy...
OnDisable
OnEnable

Appendix I: Project Management Review

It can be seen from figure 18 and appendix A that there were some clear changes made to the project pipeline through revisions. Rather than being split into 4 sub-sections, programming became split into 5 sub-sections representing each programming task. It was realized early on that the programming could not commence immediately, as it would take some time getting used to the open-source project and learning how it works. By dedicating time to this, it made each individual programming task easier and more accomplishable, due to the prior knowledge of existing systems. However, the programming for each algorithm still took longer than previously expected, and so more time was allocated to each programming task. This time originally designated to the programming tasks was clearly optimistic and overly ambitious. There was not sufficient time to produce an alternative solution, which was set as a challenge task; this was left out due to being a lesser priority than other artefacts. This was removed from the programming and analysis, however its design was still considered and briefly discussed in this report. The time given for doing this was much shorter than previously planned, as there was no longer a need to design a solution in depth ready for programming; only a simple concept was provided. The artefact analysis was simpler to do when each artefact was fresh in my mind, and so after each artefact was completed, it was analysed simultaneously with the programming for the next artefact. Once all artefacts were completed, time was then allocated to do cleanup on the code, making it more readable, simpler to understand, and efficient.

There were many strengths and weaknesses to the project management. A suitable level of work with a suitable degree of challenge was successfully selected for this project. In future projects, I will make sure that I provide sufficient time for learning the environment and any libraries required. I will also attempt to check libraries for potential compatibility issues before implementation, to save time fixing errors.

Small changes made to the project plan had a knock-on effect to other tasks, setting them back or requiring multiple tasks to be completed simultaneously, as can be seen for the artefact analysis which needed to be done at the same time as some of the programming. Doing these concurrently made it much easier to complete the artefact analysis quickly and effectively, and will be something to consider for future projects to streamline the overall process.

As a project manager, I believe I have shown good planning skills and the ability to break problems down well. However, my time management could be improved; as some tasks were set unrealistic time goals. An aspect of the project management I am satisfied with in particular is the adaptability shown when the needs of the project change. The main aspects of the project management that could have been improved are upon are the time goals, and lack of concurrency in the initial Gantt chart. If the project were to start again, I would make sure to designate plenty of time to learn the environment and libraries being worked with.

My approach to time management has changed in terms of priorities. I believe that more time should have been spent on the initial planning of how to manage the project. This time spent planning would hopefully save time on the project as a whole, and mean that more time could be spent on another task if it took longer than expected, making the workflow more flexible. I have learnt how to better decompose a problem into smaller tasks and the importance of preliminary research. Rather than just jumping right into a project, time should be initially spent planning and learning in order to make fewer mistakes during the implementation. This project has outlined that while I am able to manage my time to a good enough degree to complete work on time, I do find time management challenging; and there is always something that could be learnt and improved upon.