University of Plymouth

School of Engineering,

Computing, and Mathematics

COMP3000

Computing Project

2022/2023

Reverse Engineering Shenanigans

Vlad Burca

10705143

BSc (Hons) Computer Science

Acknowledgments

Firstly, I would like to express my gratitude to my supervisor, Swen Gaudl, for his recommendations and assistance throughout the development of this project. A genuinely exemplary individual whose expertise has provided me with the knowledge and guidance needed to overcome some technical difficulties.

Secondly, I would like to thank my classmates and friends for their invaluable feedback and suggestions and for their participation in usability testing. Their contributions have greatly aided in the project's refinement and improvement.

Finally, I greatly appreciate my family's emotional support and encouragement throughout my academic odyssey.

Abstract

This report documents the development process of a dynamic-link library cheating software for the game Counter-Strike: Global Offensive. The project's primary objective was to delve into the field of reverse engineering and produce a tangible output as a result. The developed software includes both gameplay-assisting features and neutral functionalities. The development process was facilitated by various reverse engineering tools such as Cheat Engine, The Interactive Disassembler, and ReClass.NET.

Given the controversial nature of both cheating software and reverse engineering, this report provides a detailed analysis of the positive and negative aspects of the project, as well as the potential legal, social, ethical, and professional implications.

The project adopted well-planned project management techniques by following the Agile methodology, utilizing Kanban boards and GitHub for version control. The development process was broken down into small iterative steps called sprints. Testing the software was given top priority, and various types of testing were conducted, including memory, functional, and usability testing.

Despite the challenges encountered due to the unfamiliarity with the subject matter and significant changes and deviations during the software development, such as the overhaul that led to switching the reversed game, the project successfully achieved its envisioned objectives and created functional software. The most significant outcome of the project is the invaluable knowledge and experience gained from undertaking a project of this scale and purpose.

Table of Contents

Word Count

Word count: 10876

Code Repository

GitHub: link

1. Introduction

1.1 The Topic

Video games have become an integral part of entertainment since their very commencement (Ritterfeld and Weber, 2012). In contemporary times, video games have flourished, leading to an increase in popularity. The technological advancements allow the games to have realistic, eye-catching graphics with engaging interactions and an immersion that puts reality into question. Moreover, the diversity of genres within the industry makes it challenging for individuals not to find at least one game that appeals to them. The positive impact of video games on mood, stress relief, and depression reduction has led to the widespread adoption of video games by individuals of all ages and backgrounds (Ajmal et al., 2022).

Unfortunately, every single game is doomed to have *bad actors*. These players will try to cheat and find exploits and vulnerabilities for malicious intents. It is an infamous problem that nobody likes to hear about, and alas, a never-ending game of cat and mouse between developers and cheaters. Naturally, this only affects online multiplayer games, which overall have a higher player base and receive more traction than single-player games. While there is no direct research backing that up, it is a reasonable assumption because multiplayer games are designed with replayability and higher longevity in mind.

Usually, cheats are associated with ill-mannerism, malevolent intentions, and a means for someone to gain an unfair advantage. Remarkably, there is a spectrum to it, and they can have positive use cases that are not harmful or interfere with someone else’s gameplay experience.

This vision and perspective are precisely what the project Reverse Engineering Shenanigans represents. It is a piece of software in the form of a dynamic-link library developed for a popular online game Counter-Strike: Global Offensive. It is able to inject into the game’s memory and run seamlessly as if it were a part of it. The game is a first-person shooter; therefore, the software offers features appropriate to that specific genre. The project aims to open up opportunities for fun experiences that would otherwise be unachievable by conventional means. Although it has grounds for potential misuse or negative consequences, it is meant and intended to be utilized in a positive and neutral setting.

1.2 Report Overview

The report commences with a comprehensive project background, contextualizing its rationale, motivation, and objectives. It subsequently delves into the approach adopted to address the project's objectives and requirements. It provides an overview of the methodology employed and the various technologies and tools utilized during software development. Next is a thorough discussion of the software's legal, social, ethical, and professional implications, given its controversial nature.

The report continues with an overview of the project's architecture and design aspects, along with the project management techniques and measures implemented. Following is a breakdown of the project's evolution process, structured into sprints. The report then proceeds to discuss the testing process, characterized by different types and stages of testing.

Finally, the report concludes with a summary of the project's objectives and achievements, reflecting deeply on the project's entire journey, including the mistakes and positive outcomes. It also includes the project's future avenues and prospects.

2. Background

2.1 Aims, Reasoning & Motivation

The project's purpose was twofold: facilitate access to novel and exciting experiences through game modifications and dive into reverse engineering to explore its intricacies and possibilities.

2.1.1 Reasoning – The good & fun side of cheats

The aforementioned statement regarding the utilization of cheats in an unharmful manner refers to the use cases and scenarios mentioned in this section.

2.1.1.1 Hack vs. Hack (HvH)

Although it sounds odd, CS: GO is so unique that it adopted and formed a niche subculture called HvH. It is a community of players that play with cheats to compete against each other.

The same game and rules, now the battle’s outcome is most likely determined by the team with better cheat functionalities and specific tactics. For example, a feature called *Spin Bot* spins a player’s in-game character model so fast that it becomes challenging for the opponent’s *Aim Bot* feature to calculate the correct position and angle at which it has to aim. The action occurs on dedicated servers to not interfere with regular players.

Overall, it is an unusual experience characterized by intense and fast-paced battles. The community has grown to love it so much that it has become a form of entertainment, with players streaming their matches and organizing tournaments.

2.1.1.2 Singleplayer

Undoubtedly, using cheats in single-player is entirely harmless unless *bots* and *NPCs* have feelings. Despite CS: GO being designed for multiplayer, it still features a single-player mode with matches against bots. Naturally, the enjoyment is way stronger in single-player games with a story or campaign by design. Cheats can provide a sense of power and opportunities to go beyond what is allowed and normal, which feels rewarding.

The games' developers understand this feeling, and some implement cheat codes, usually hidden, in the form of *Easter eggs*. One example of a cheat code that goes way back to 1986 is the *Konami Code*, which provides the player with 30 extra lives; it can be encountered in many Konami video games, though the most notable one is Contra (Edwards, 2021).

2.1.1.3 Neutral functionality

Along with gameplay-assisting features, the software offers neutral functionality, such as the Skin Changer. It is a feature that allows players to modify the appearance of their in-game weapon models. It is a fascinating way to personalize the game experience and make it more enjoyable.

The project's weapon changer is just the tip of the iceberg. In reality, every asset in the game, such as sound, weather, *skybox*, and player models, can be modified. Invested game fans often seek out these visual features, especially when bored or wanting to add spark and variety to their favorite game.

2.1.2 Motivation – Experience

Reverse engineering is an intricate and multiplex process involving dissecting a piece of software or system to analyze and determine its way of operation (Lutkevich, 2021).

Unlike other fields of study, reverse engineering is a rather guideless topic and can only be fully grasped with hands-on experience. While *practice makes perfect* can refer to any area of expertise, it is especially true for reverse engineering since its learning process is predominantly practice-oriented. Without experience dealing with *obfuscation*, working with various complex tools, and understanding technical jargon can be an ordeal that can stagnate the work for a lengthy period.

This overall challenge presented by this specialized subject area and the opportunities in different industries that can open up by having expertise in reverse engineering is precisely why this project piqued the author's attention and eventually materialized.

2.2 Objectives

A set of clear objectives have been defined for the project to be achieved by its completion. These comprise CS: GO functionalities that are gameplay-aiding nature and neutral features.

2.2.1 Assisting features

The following assisting features grant significant power, and their use against regular legit players is not tolerated.

- Aim Bot – automatically aims at the enemy's in-game character model with pixel-perfect precision.
- Trigger Bot – automatically fires the weapon as soon as the player aims at another enemy; it is a significantly weaker version of an Aim Bot.
- Extra Sensory Perception (ESP) – retrieves information about other players' positions, health, ammunition, and much more. It is an improved version of a long-forgotten *Wallhack* feature, which simply renders the walls transparent.
- Anti-Flash – makes the enemies' *Flashbang* grenades useless by voiding their blindness effect.

2.2.2 Neutral features

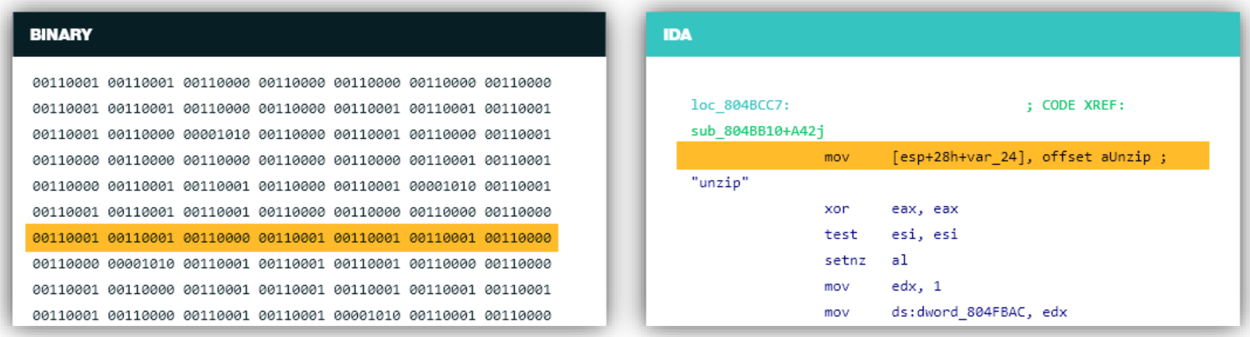
Neutral features do not threaten regular players, as they do not provide an advantage.

- Skin Changer – allows the customization of the player's in-game weapon models appearance.
- Static crosshair – renders a static crosshair in the middle of the screen; it proves useful for tricks such as *Quickscope*. Normally, the game's built-in crosshair disappears when wielding specific rifles. Some consider this feature a bit cheaty, though modern monitors already have it as an option.
- Bunny Hop – issues a jump command as soon as the player's character lands on the ground; consecutive well-timed jumps provide a movement speed boost. Likewise, it could be considered cheaty, though the same results can be achieved by modifying the provided in-game configuration file.

2.3 The Importance of RE in Cheat Development

Reverse engineering is the solitary way to develop high-level cheats. Upon compilation, every computer program is transformed into *machine code*. It is considered the lowest-level language because its instructions are represented using binary digits of 0 or 1. The rationale for the exceeding simplicity is that the Central Processing Unit (CPU) can understand and execute these instructions directly.

Fortunately, reverse engineering is not as cruel as digging through machine code and making sense of it. Some specialized tools offer a disassembling functionality that transforms binaries into Assembly, the second lowest-level language. An illustration of a disassembler can be seen below in Figure 2.3.1.



```
00110001 00110001 00110000 00110000 00110000 00110000 00110000
00110001 00110001 00110000 00110000 00110001 00110001 00110001
00110001 00110000 00001010 00110000 00110001 00110000 00110001
00110000 00110000 00110000 00110000 00110000 00110001 00110001
00110000 00110001 00110001 00110000 00110001 00001010 00110001
00110001 00110001 00110001 00110000 00110000 00110000 00110000
00110001 00110001 00110000 00110001 00110001 00110001 00110000
00110000 00001010 00110001 00110001 00110001 00110000 00110000
00110001 00110000 00110001 00110001 00110001 00110001 00110001
00110001 00110000 00110001 00110001 00001010 00110001 00110000
```

```
loc_804BCC7:                            ; CODE XREF:
sub_804BB10+A42j
                mov     [esp+28h+var_24], offset aUnzip ;
"unzip"
                xor     eax, eax
                test    esi, esi
                setnz   al
                mov     edx, 1
                mov     ds:dword_804FBAC, edx
```

Figure 2.3.1 – The disassembling process using The Interactive Disassembler (IDA)

While Assembly might still come off as off-putting, it is at least composed of human-readable syntax instead of raw bits.

Simply put, making sense of the disassembled binaries is all that's needed as they constitute the mocked-up source code of the reversed software represented in Assembly. Another vital aspect of reverse engineering is filtering out and searching only for the necessary parts; for example, the structure of the Player class. Nobody casually reverses whole applications as big as games. Usually, they contain hundreds of thousands of lines of code written in a high-level programming language, spanning up to ten times more lines of code when converted to Assembly.

2.4 CS: GO Overview

Counter-Strike: Global Offensive is a team-based first-person shooter that originated in 2012 and is currently the latest series of the *Counter-Strike* franchise. It features various captivating maps, characters, weapons, and game modes (Valve Corporation, 2012).

2.4.1 Selection reasoning

Besides the author's experience and involvement in the game, many other reasons with a higher weighting factor contributed to selecting CS: GO. The exact justifications, ranked in order of importance, starting from the most significant:

1. Architecture – the determining factor in the overall choice. An application compiled for the 32-bit architecture is easier to work with and reason about, especially regarding the reverse engineering process. It has a less convoluted code structure, lower memory addresses, less memory capacity, fewer *registers*, and the list goes on.

2. Genre – the genre of shooters is recognized for providing a broader spectrum of cheating possibilities compared to other gaming categories.

3. Experience – familiarity with the game and its distinct mechanics provides better insight into features that would be useful and cool to implement.

4. Skin Changer – for the longest time, until 2020, Valve was not taking any direct measures against these features. Ever since there were no public alternatives, and the author felt deprived of his beloved possession, which sparked the motivation to create something personal.

5. Popularity – the game's continued relevance over the years serves as an incentive to develop software that will remain relevant for years to come.

2.4.2 Anti-Cheat measures

Anti-cheat measures are essential to every online multiplayer game, and CS: GO is no exception. They protect the integrity of the game and help retain its relevance and player base. This section discusses CS: GO's anti-cheat measures and their effect on the project.

It should be noted that the utilization of the software presented by this project violates the game's Terms of Service, which is punishable with a VAC ban on the user account. Although the software provides novel experiences, the user should take full responsibility and liability for any repercussions, given the inherently compromising nature of cheating.

2.4.2.1 Networked Variables (Netvars)

Netvars represent variables stored and controlled by the game server. Undoubtedly, the best measure at preventing total anarchy because the client can *only* read their values from memory. Netvars are used for potential game-sensitive information like health, money, movement speed, ammunition, etc.

It does not affect the project in any way. Furthermore, modifying the netvars is only possible through vulnerabilities and compromising the game server, which is a legal violation.

2.4.2.2 Valve Anti-Cheat (VAC)

VAC is Valve's proprietary software that protects numerous games hosted on Steam. Besides other cheat-detection methods, the VAC system can reliably detect cheats using their binary signatures (Valve Corporation, 2023a). In other words, VAC scans for running applications and its memory integrity for any potentially injected DLLs against their database of cheat signatures; in case of a match, it issues a ban.

It does not affect the first two use cases mentioned in section 2.1.1 because they are meant for gameplay on *VAC-unsecured* servers. An attempt to use any gameplay-assisting feature on protected servers will result in a relatively fast account restriction.

However, using neutral functionality on VAC-secured servers is tricky and risky. Although the anti-cheat does not scan for memory modifications specific to these features, it is enough for one individual to provide the binaries of the software to Valve, making it easy to flag and detect.

2.4.2.3 Overwatch System

The Overwatch System is a clever solution that allows the community to self-regulate by reviewing in-game *replays* of their matches (Valve Corporation, 2023b). It is tailored to punish all kinds of misconduct, such as griefing and miscommunication, though it was predominantly crafted against cheaters.

The system is highly effective against individuals who use cheats in conjunction with a *VAC bypass*. Once they receive enough reports, their game replays are sent to be reviewed by real players that can usually make a well-educated verdict about someone cheating.

The project remains unaffected by the system as it is not designed for secure matches. Additionally, the Skin Changer is not detectable by Overwatch, given that the customized visual aspects are exclusively client-side.

2.5 Deliverables

Unequivocally, the project’s final high-level deliverables are tied to its objectives. Their successful completion should yield a dynamic-link library with the cheating functionality mentioned in the project’s objectives. Additionally, an Injector capable of loading the specified DLL inside the target process; in this case, the game Counter-Strike: Global Offensive.

3. Method of Approach

3.1 Agile

Agile is a flexible and iterative approach to software development and project management (Agile Alliance, n.d.a). It is widely practiced in the IT industry due to its emphasis and precedence on values mentioned in their Manifesto, such as customer satisfaction, cross-functional teams collaboration, working software, and adaptability to changes as opposed to more formal and rigid principles (Agile Alliance, 2001).

Although Agile is primarily used in team-based projects, its effectiveness is not hindered in individual projects like this one. Moreover, it is quite impressive how well Agile suits the current project. One of its key advantages is the ability to handle unpredictable and rapidly changing situations, which is exceptionally relevant in reverse engineering. The reverse engineering process is often spontaneous and unpredictable, especially when approaching it with little to no prior experience. It requires constant adaptation and improvisation to discover features, functions, and variables necessary for the required cheating functionality.

Agile’s iterative approach allows for flexible implementation of each functionality by dividing the whole process into small increments, called sprints. That is extremely suitable for the project because developing a single cheating feature involves many other small steps, and each step’s complexity is often variadic and unknown.

All in all, Agile methodology is extremely well-suited to the unique challenges of reverse engineering. Therefore, other options, such as Waterfall and Spiral, were hardly a consideration due to some of their limitations.

3.1.1 Minimum Viable Product (MVP)

The Minimum Viable Product is yet another critical element of the Agile methodology. Its concept is to define and deliver a product with the bare minimum functionality that still offers value and usability to the end user. The goal behind the said basic version of the product is to gain customer feedback as soon as possible by launching and testing it in the real-world (Agile Alliance, n.d.b). Hence, using that validated learning early in its development phase to stay on the right track and improve upon the product as it evolves. It significantly reduces the risk of developing an unusable product that does not satisfy the end-user which ultimately saves a ton of resources and avoids overhauls.

Reverse Engineering Shenanigans is no exception, and its MVP was drafted at the outset of its development, specifically during the first sprint, later to be achieved in the fifth one. In the context of this project, the identified MVP constituted the features that would classify the software as a ‘cheat’. In broad terms, these represent the following:

- The ability to attach to the target process’s memory.
- A memory manager class capable of reading from and writing to the game’s memory as well as interacting with Windows API.
- A hooking library that sets traps to hijack the game’s execution flow and redirect it toward the DLL’s code.
- Finally, with the prerequisite tools in place, reverse engineer the portions of the game’s binaries needed to achieve an ESP – the most basic and easily achievable cheat.

Overall, the MVP served as a great guide for prioritizing the development efforts. Having focus and targeting the essential features allowed the author to conduct the first usability test and utilize the feedback further down the road.

3.1.2 Risk assessment

Risk assessment is another crucial part of the Agile methodology. It involves identifying the potential risks and analyzing their likelihood of occurring and their impact severity on the project. Overall, it significantly minimizes the chances of setbacks and deviations in the project timeline.

With that being said, the potential risks identified in the context of this project and their mitigative actions are the following:

1. Legal issues: Developing cheats for video games is a contentious issue with many nuances and caveats to consider and weigh. Therefore, the potential risks must be carefully assessed and ensure that the software does not violate any legal standards and that its development is solely for educational purposes.

2. Account ban: As stated previously, using any third-party memory-modifying software in the game on VAC-secured servers could result in an account ban. Therefore, the prospective dangers must be clearly communicated and given guidance on their possible avoidance.

3. Game updates: Game updates represent a significant problem for every cheat developed for an online game. Once the game's source code changes, some of the portions in the game's binaries will also change upon a new compilation. Therefore, it leads to a mismatch in the memory addresses that the game has and what the cheat is trying to access. To prevent that, the software must implement a pattern scanning technique to update itself along the game.

4. Memory safety: Modifying game memory improperly can cause crashes and undefined behavior. The software must be thoroughly tested and have memory validation and security checks to avoid this.

3.1.3 High-level project plan

Although not part of Agile, establishing a high-level project plan is another step towards a safer and greater project. It is a plan that outlines the major milestones of a project and the steps, resources, and time required to achieve them (Indeed Editorial Team, 2023).

Although Agile emphasizes flexibility with project plans that are never set in stone, it was created to serve as a *rough* roadmap throughout the development process rather than a definitive plan of action. The initial project plan, drafted in a Gantt Chart, can be seen below.

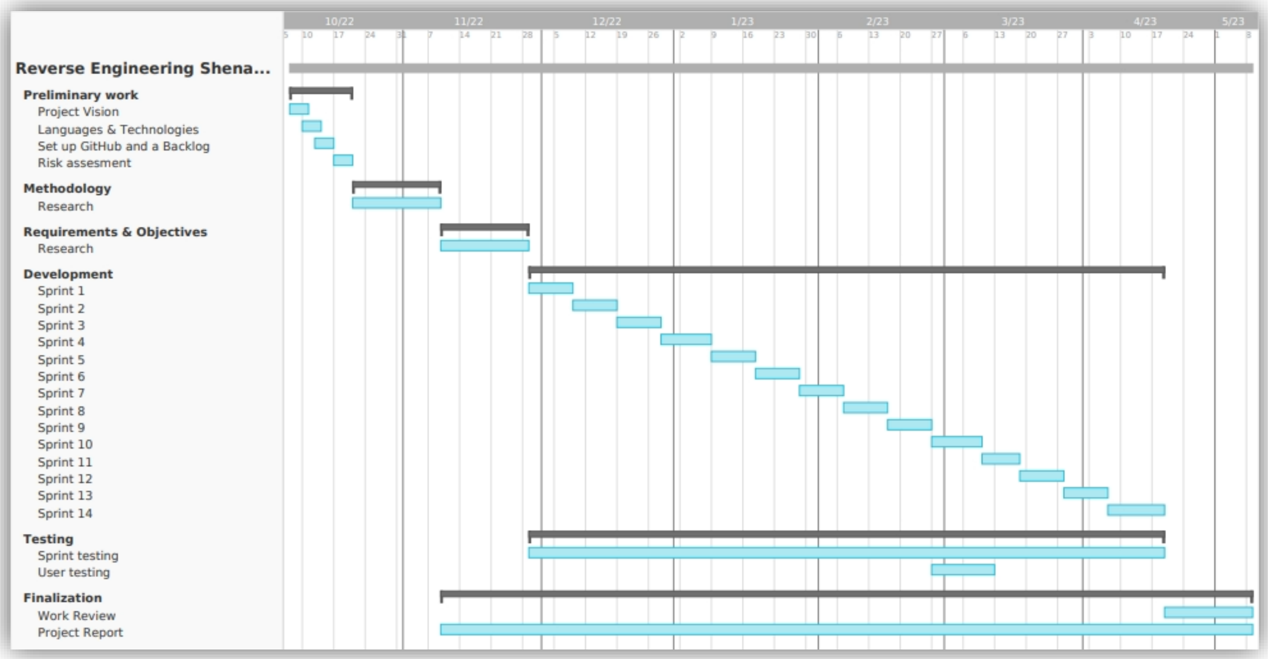


Figure 3.1.3.1 – Gantt Chart representing the project's high-level plan.

The sole version was designed to depict the bigger picture of the project and would not be updated during its progress, instead relying on Kanban for project management.

3.2 Kanban

Since Agile technically represents a project management principle rather than a methodology, it will be implemented by the Kanban framework to produce a holistic project management philosophy. Kanban is a popular workflow management method that has its roots in the 1940s in Japan and is often used in industries requiring management, such as software development and manufacturing.

The idea behind Kanban can be deducted from its Japanese translation, "visual board" or "sign," which conveys the idea that the workflow is visualized using signs in the form of cards that represent tasks ("What Is Kanban?", n.d.). However, that is just the gist because the tasks are sorted out in columns on a board, each representing various stages of completion (Martins, 2022).

Another important aspect of Kanban which dates back to its author's vision, is its pull system, referred to as "just in time" ("What Is Kanban?", n.d.). In essence, it adopts the *supply-on-demand* strategy, which comes with the benefit of minimizing the work in progress to improve efficiency and productivity.

Regarding this project, the Kanban practices were implemented using Trello, whose usage is discussed further in the report, section 6.2. Overall, using Kanban's project management principles ensured the avoidance of any possible bottlenecks, as well as the completion of the tasks in a timely and efficient manner due to them being prioritized and sorted out in appropriate categories.

3.4 Technologies & Tools

The development of the software was facilitated with the use of various technologies and tools. These comprise the target operating system, the accompanying reverse engineering tools, and programming languages.

3.4.1 Operating System

Generally, there is little choice in operating systems when it comes to gaming, let alone cheating. The industry is primarily based on platforms with the most extensive user base, such as Windows PC and video game consoles. However, some popular games, including CS: GO, might still offer native support for operating systems such as macOS or Linux.

Although the software is targeted for Windows, there were considerations regarding Linux as a possible cross-platform option. Obviously, it was a high hope to pursue due to time constraints and the need to familiarize oneself with a different memory layout and API.

3.4.1.1 Windows

Windows was the obvious number one option for several reasons, such as popularity, a more comprehensive range of reverse engineering tools, better documentation available through the rich resources of MSDN and Stackoverflow, and the author's overall convenience and experience with the platform.

An important notice is that this application was developed using Windows API, otherwise known as Win32 API, which provides a high-level abstraction for creating user-mode applications. It is a subset of the larger and more powerful Windows Native API.

The Native API can be used to develop kernel-mode applications with the highest privilege level, similar to the operating system. Unfortunately, developing kernel-mode applications is a *highly specialized* area requiring extensive knowledge of Windows's internals. Moreover, Microsoft does not provide official documentation for the Windows Native API; information about it can only be found through their past documentation or other third-party sources. Thus, transitioning to a kernel-mode application is somewhere in the far future works.

3.4.1.2 Linux

In contrast to Windows, cheating on Linux can be easier due to its open-source kernel and the overall customizable nature of the Linux ecosystem. More flexibility and control over the operating system is precisely what the cheaters seek and need.

Besides a smaller user base, cheating is another major factor in why games do not have support for Linux. It is challenging for game developers to craft a well-rounded anti-cheat that accounts for different Linux distributions and kernel modifications.

3.4.2 Languages

The project involves two programming languages, one chosen deliberately, while the second language, Assembly, was necessary.

3.4.2.1 C++

The programming language choice for the software involved little contemplation. C++ was the first and only thought as it is a powerful high-level programming language that offers fine-grained control over memory, making it ideal for game modifications and high-performance applications.

Other decisive factors include the author’s personal experience with the language and the game being written in C++, which could provide insights and advantages during the reversal process by knowing some of the internal key features of the language. Furthermore, the fact that much of the Windows operating system is written in C++ is a testament to the language’s power and versatility.

3.4.2.2 Assembly (Intel syntax)

Assembly is a low-level programming language usually used to develop applications that interact with the hardware directly due to its outright high efficiency (Preston, 2023).

In the context of this project, Assembly was needed to make sense of the disassembly output generated by reverse engineering tools like IDA and Cheat Engine. Understanding the assembly language's syntax is the first and most crucial step in reverse engineering.

3.4.3 Reverse Engineering Tools

Several tools have aided in the reverse engineering process. Ultimately, they can all disassemble the binaries, which is their inherent purpose. However, there are differences in their functionality that make them uniquely better in different scenarios.

3.4.3.3 The Interactive Disassembler (IDA)

The Interactive Disassembler is an excellent tool for disassembling and analyzing binary files. It is so powerful that it is often referred to as the “de-facto industry standard disassembler” (Prado & Erickson, 2018).

It has primarily been used for static analysis due to its sublime interface, allowing easy navigation through the disassembled code. Other helpful features include a function-relationship graph view and the ability to detect known imported functions, such as those from the Windows API. Additionally, it has the capability to develop scripts and plugins that can further personalize and streamline the reverse engineering process.

3.4.3.4 Cheat Engine

Cheat Engine is a popular tool for memory analysis and debugging purposes. Although it may initially seem feature-scarce due to its simplistic interface, it unleashes its true potential once users become familiar with it.

It has been used for dynamic analysis due to its ability to attach to a running process and provide a live view of the entire memory. That makes it easy to understand which values change based on interactions with the game, making it the primary technique for searching for the necessary in-game values. In some sense, it is the most intuitive and easy-to-use tool because it can scan for in-game values in different formats, such as integral numbers, regular text, and even raw bytes.

3.4.3.5 ReClass.NET

ReClass.NET is a unique memory scanner primarily used for reverse engineering classes and structs. Its feature set revolves around understanding and reversing complex data structures by providing ways to name addresses, assign data types, and allocate bytes in a comprehensible interface.

4. Legal, Social, Ethical, and Professional Issues

The development of any software considered for distribution involves a lot of responsibility and consideration due to the potential impact it can have on both its author and end-users. That intensifies even more when the software is, by convention, controversial in nature.

Developing cheating software for video games is a sensitive and intricate topic that could raise several legal, social, ethical, and professional concerns. It is essential to address them to ensure and justify that the project’s existence is based on judicious and reasonable manners.

4.1 Legal

The first and foremost topic that needs attention is reverse engineering, which is how most of the software’s functionality was attained.

4.1.1 Cheat development

In the United Kingdom and generally around the globe, the applicable laws regarding reverse engineering typically offer leeway, with due consideration to the potential caveats (Peil, 2019).

According to the Steam Subscriber Agreement, which is a legal agreement that governs the use of its software and services, it is prohibited to reverse engineer, modify, disassemble, and decompile any software accessible via Steam unless given consent or permitted under applicable law (Valve Corporation, 2023c, section 2. G).

To dispute the aforementioned prohibition, UK legislation offers statutory protection under the Copyright, Designs and Patents Act 1988. While it is intended to protect the creators' work against fraudulent activities such as unauthorized copying, distribution, or modification, it still has exceptions. One of which is a declaration stating that reverse engineering is considered legal as long as it does not involve copying or source code extraction in an attempt to steal ideas from it. Clearly, it does not affect this project, as it only reversed some parts of the game to understand it and make appropriate client-side modifications. Additionally, the Act promotes fair use and dealing, which permits limited use of copyrighted material for educational or research purposes (Copyright, Designs and Patents Act 1998).

Intellectual property and licenses outside the reverse engineering process were also considered, ensuring compliance with their regulations and requirements. For instance, the software uses some code from Valve's Source SDK. Therefore, it must comply with its license prohibiting further commercial software distribution. Any other third-party components or resources used in software development were also attributed and referenced on the GitHub page.

This project has no other legal implications, such as any common data protection acts, because the software does not collect personal user data. Moreover, it is fully open-source and licensed under MIT.

As a final remark regarding the legality of cheat development, or reverse engineering per se, it is worth noting that there are currently no documented court cases or legal consequences for *these actions alone*. However, the issue typically arises with large cheat distributors whose user base exceeds twenty percent of the game's user base. In such cases, it can be argued in court that their software sales present severe damages to the company. A well-known case is a ten-million-dollar lawsuit involving Tencent Games and Krafton against a hacking company that developed and sold cheats for their online game, PUBG Mobile (Moore, 2022).

4.2 Social & Ethical

Using cheating software in online games to gain an unfair advantage over other players is undoubtedly pushing the boundaries of social and ethical norms.

4.2.1 Cheat usage

As a developer, it is crucial to consider the impact of this project's creation on the socio-ethical dynamics within the gaming community. Indisputably, its usage is frowned upon because it negatively affects the gaming experience of others, which leads to an overall toxic gaming environment. Moreover, cheating can result in a loss of trust in the game's community and damage its reputation, ultimately leading to a sad decline in relevance and popularity. Cheating is also unethical towards game developers, as it undermines their invested efforts in creating an enjoyable experience.

The author acknowledges these implications and emphasizes that this project was undertaken chiefly for educational purposes and does not encourage or endorse using this software for unethical or immoral intents. However, it is important to recognize that cheating is *not always black and white*, as previously stated. Therefore, responsible usage, with limitations to single-player modes or neutral features, is always welcome.

Ultimately, it is up to the individual's ethical and moral values to determine the usage of the software. While the cheat has harmful potential due to its gameplay-assisting functionality, *it cannot be effectively fulfilled* as the software does not provide protection against the game's anti-cheat system. Consequently, users who cheat will face penalties under the Steam Subscriber Agreement, which is an account restriction to the game (Valve Corporation, 2023c, section 4.).

4.3 Professional

Maintaining a high professional standard is another crucial aspect to consider and uphold during software development. Although developing cheats contradicts professionalism, its existence and purpose have been thoroughly justified in the sections above.

The author has maintained a professional software development approach that is in accordance with a certified Computer Scientist's standards. That implies adhering to a strict methodology, following project management principles, researching, conducting user testing, and developing high-grade software. These characteristics highly reflect a developer's professionalism and separate casual development.

5. Architecture & Design

5.1 Architecture's Core Aspects

The architecture of the software was designed with the consideration of several vital aspects in mind.

The most crucial characteristic that highly affects cheats is performance. For optimal experience, cheats have to stay in sync with the game's execution. Usually, this means constantly reading from and writing to the game's memory to perceive its state at any point in time. Alas, this performance sacrifice has to be done in favor of responsiveness and a jitterless experience.

The first step towards performance improvement was the programming language choice. As discussed earlier, C++ represents a highly performant language with low-level capabilities that perfectly suit the project's needs. Another significant performance-enhancing remedy is the Internal cheating modality, whose impact is discussed further in the report. Additionally, the software follows good programming practices such as avoiding unnecessary operations, min-maxing the data types, and caching frequently used data. Finally, as a safeguard, it is compiled under an optimization profile.

Another crucial aspect of the software is its modularity. Each feature is encapsulated in a separate class. It allows for easy maintenance and scalability, as new functionality can be added without affecting the rest of the code. Additionally, it introduces the separation of concerns, as each class is mostly independent. Finally, the project is further maintained and managed through a version-controlled environment.

Last but not least is the software's robustness. It is a crucial aspect that cannot be overlooked due to the cheat's frequent interaction with raw memory addresses. These activities are susceptible to errors and crashes, making resilience a necessity rather than an option. As a result, the software has undergone different types and stages of testing to ensure flawless and uninterrupted operation.

5.2 Design Pattern

The application mostly follows the Utility design pattern to improve its structure and code organization. Although it is not a widely recognized design pattern, it refers to classes or functions that provide general-purpose functionality that can be easily used across different application parts.

While utility classes are often viewed skeptically in object-oriented environments, it makes perfect sense for this application. Every cheating functionality is better off being encapsulated in static classes because none have any state.

The Utility pattern was deemed appropriate for the project and has proven beneficial in promoting code reusability, simplicity, modularity, and maintainability. These aspects have been previously mentioned and discussed as crucial components of a successful project. Adopting this pattern has resulted in a more coherent and manageable codebase.

5.3 Cheating Modality

Cheating software have different cheating modalities, which considerably change their core architecture. Generally, there are two distinct types of cheats: Externals and Internals. The difference between them is substantial, varying in functionality possibilities, memory access, stealthiness, and performance. The sections below provide a general idea of their modus operandi and the rationale behind the choice.

5.3.1 External

External cheats have the file extension of an executable (*.exe*). Their name suggests that they operate as a *separate* external process. That means that communication with the target game and access to its memory is solely possible with the help of a process that has access to all system resources, which is the Windows kernel. Thus, the cheat can query information about the game's process via the Windows API, which will be facilitated by the Windows kernel operating in *Ring 0*, a *protection ring* with the highest privileges, as depicted in Figure 5.3.1.1. It is a normal phenomenon that has to happen due to Windows's isolation and protection between processes which separates them in their own virtual memory space.

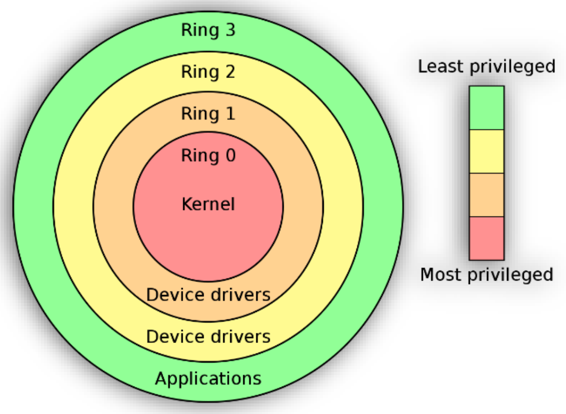


Figure 5.3.1.1 – The protection rings available on Windows.

There are several other technicalities, though this is generally the gist of its operating technique. External cheats come with several limitations due to the ordeal of having to operate indirectly via a third party. These include fewer cheating possibilities due to memory access limitations, significant degradation in performance, and worse stealthiness. For instance, the game's anti-cheat can simply call the function 'NtQuerySystemInformation' from the Windows API to list every open *handle* to the game, which are resources used by External cheats to communicate with the process (Microsoft Developer Network, 2021).

5.3.2 Internal

Internal cheats have the file extension of a dynamic-link library (*.dll*). As the name implies, they operate internally, meaning *they work from within a process*. As a result, their interaction with the game's memory is seamless and unobstructed since the cheat is already part of the game's memory.

5.3.3 Verdict

The advantages of Internal cheats over Externals are undeniable, as they provide more flexibility and possibilities when used appropriately. Prior to the overhaul discussed in section 7.2, the software functioned in External mode due to its simpler learning curve. Nonetheless, as the software's development progressed, its limitations became apparent and were hindering the attainment of certain project objectives and requirements.

5.4 Interface

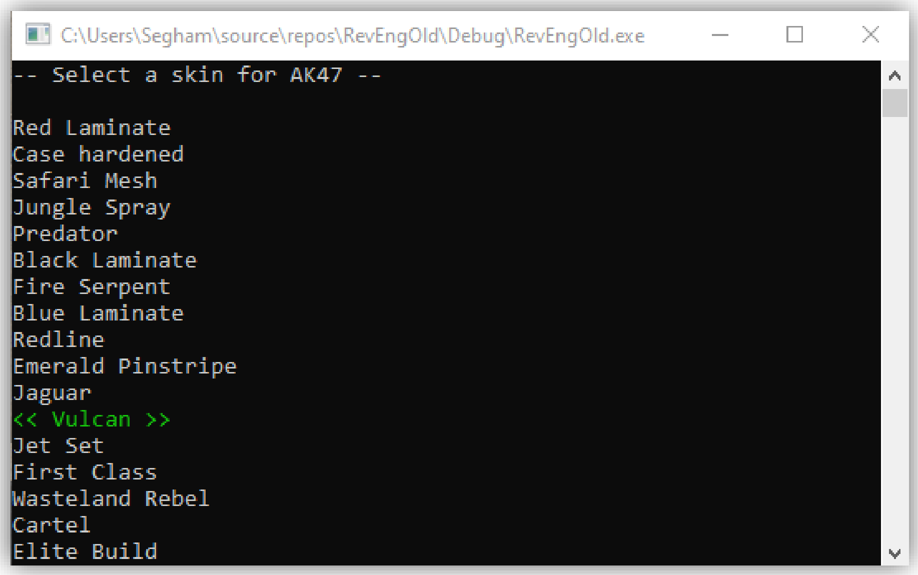
A sublime interface is often one of the key driving factors toward an application's success. Humans are naturally more likely to be attracted to a simplistic and intuitive interface than having to consult a manual to navigate an application. This project recognizes the importance of this aspect, which is why it underwent several phases of significant interface changes to ensure that it meets these standards.

5.4.1 Command Line Interface (CLI)

The command line interface (CLI) was initially used to implement the cheat's interface. It has existed since the dawn of computers, which is reflected in its simplistic nature.

Despite its straightforwardness and quick setup with a single line of code, the CLI presents several drawbacks. These include a tedious configuration process for proper menu navigation, the need to alt-tab out of the game to access the cheat's interface, performance degradation due to the need to spin in a while loop to listen for user input, and aesthetic reasons that make it unsuitable for a cheat.

Considering these complications, in conjunction with the feedback from the first usability test, prompted the decision to transition to a graphical user interface that was later achieved in the eighth sprint. The CLI was retained solely as an output for debugging purposes, though an early version of one of the menus can be observed below.



```
-- Select a skin for AK47 --

Red Laminate
Case hardened
Safari Mesh
Jungle Spray
Predator
Black Laminate
Fire Serpent
Blue Laminate
Redline
Emerald Pinstripe
Jaguar
<< Vulcan >>
Jet Set
First Class
Wasteland Rebel
Cartel
Elite Build
```

Figure 5.4.1.1 – The CLI menu for the Skin Changer functionality.

5.4.2 Graphical User Interface (GUI)

The software made a significant leap forward, transitioning from a CLI to a graphical user interface, eradicating the previous limitations. The GUI provides a more visually appealing interface with faster navigation and an intuitive way to access different features of the cheat. In addition, unlike the CLI, the GUI is integrated into the game and not displayed as a separate window.

The project went through two major phases of GUI development. Initially, GUI elements were manually drawn using the game's graphics rendering API, Direct3D, which was tedious. This burden was later eliminated by integrating the ImGui library, which made menu configuration a much more enjoyable experience. A snapshot of one of the ImGui menus is shown below.

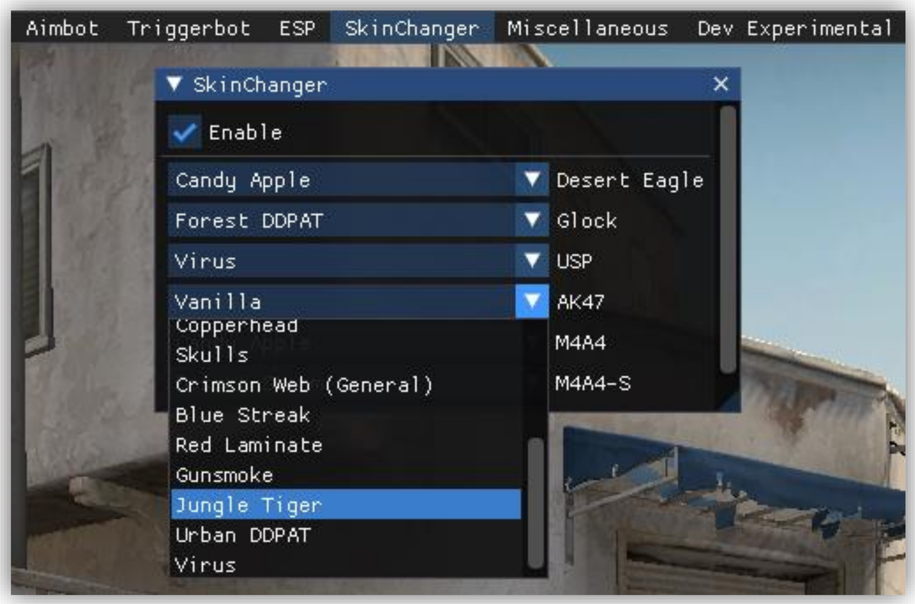


Figure 5.4.2.1 – The ImGui menu for the Skin Changer functionality.

6. Project Management

Enforcing well-thought project management techniques is crucial to completing the project’s goals within its specified timeframe. Generally, they involve planning, monitoring, gathering feedback, and evaluating.

6.1 Supervisor Meetings

Meetings with the project supervisor were held throughout the software development life cycle to obtain feedback or address concerns. These meetings were conducted fortnightly, aligning with the project sprints' timeframe, allowing the supervisor to review progress at the appropriate time. As a result, the set schedule served as a catalyst to ensure that progress was being made within the allotted time and to avoid procrastinating.

6.2 Trello

Trello is a well-known project management and collaboration tool that revolves around the Kanban principles (“What is Trello?”, 2022). As previously mentioned, the Kanban system is a simple yet effective method for organizing workflow using cards on a board.

Trello, as a web-based platform, is well-suited for the Kanban system because of the user-friendly interface that makes it easy to create boards, lists, and cards and manage them using various settings.

In summary, utilizing Trello and its functionalities helped the author steer the project toward successful completion. The initial Trello setup is shown in Figure 6.2.1, where the columns representing different stages were later filled with prioritized tasks assigned with specific timelines.

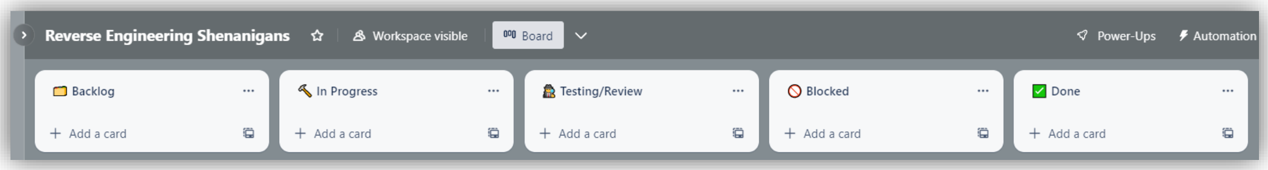


Figure 6.2.1 – Initial Trello board.

6.3 GitHub Version Control

GitHub is a popular online platform built on top of the Git version control system. With Git at its core, it allows developers to host their codebase remotely and efficiently manage it using various features, such as issue tracking, collaboration, test running, and much more (Brown, 2019).

The importance of setting up a version-controlled environment for a project has been widely acknowledged, which is why it was implemented at the outset. It can be compared to creating a safety net before embarking on a high-wire act. Just like a safety net protects a performer, Git allows developers to make changes and experiment without fearing losing their work or making irreversible mistakes.

7. Development

The project has had an interesting development journey that consisted of multiple sprints. Each sprint represents a significant leap toward the project’s completion. Although the project has faced a significant deviation in its objective, the milestones were mainly achieved on a constant two-week basis.

7.1 The Reversal Process

Before diving into the sprints that describe the development of specific cheat features, it is essential to grasp the *general concept* of their attainment through reverse engineering. This section will explore its details on a basic example, while consecutive sprints will be explained in broader terms.

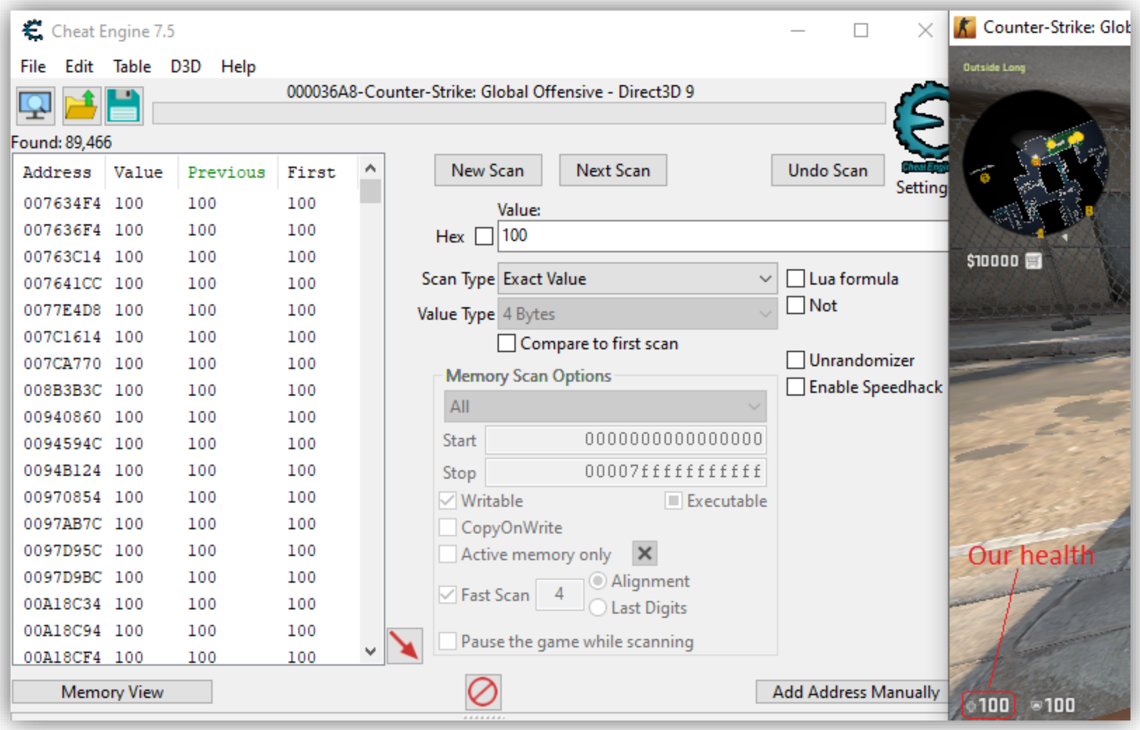
Generally, there is never an exact way of obtaining specific resources from the game, so it often involves much trial and error, hypothesis formation, and evaluation.

7.1.1 The basics

The most basic example that is also often the starting point of developing a cheat is getting the memory address of the *local player*.

7.1.1.1 Hypothesis formation

There could be thousands of ways to get the player's address, though it is often best to start from the most basic clues. In this case, it is the player's health. Logically and according to object-oriented programming (OOP) concepts, the health and other attributes must be bound to the player object. Therefore, the first step would be to scan for the health value using Cheat Engine, as shown below.



000036A8-Counter-Strike: Global Offensive - Direct3D 9

Found: 89,466

Address	Value	Previous	First
007634F4	100	100	100
007636F4	100	100	100
00763C14	100	100	100
007641CC	100	100	100
0077E4D8	100	100	100
007C1614	100	100	100
007CA770	100	100	100
008B3B3C	100	100	100
00940860	100	100	100
0094594C	100	100	100
0094B124	100	100	100
00970854	100	100	100
0097AB7C	100	100	100
0097D95C	100	100	100
0097D9BC	100	100	100
00A18C34	100	100	100
00A18C94	100	100	100
00A18CF4	100	100	100

Value: 100
Hex ☐
Scan Type: Exact Value
Value Type: 4 Bytes
☐ Compare to first scan
☐ Lua formula
☐ Not
☐ Unrandomizer
☐ Enable Speedhack

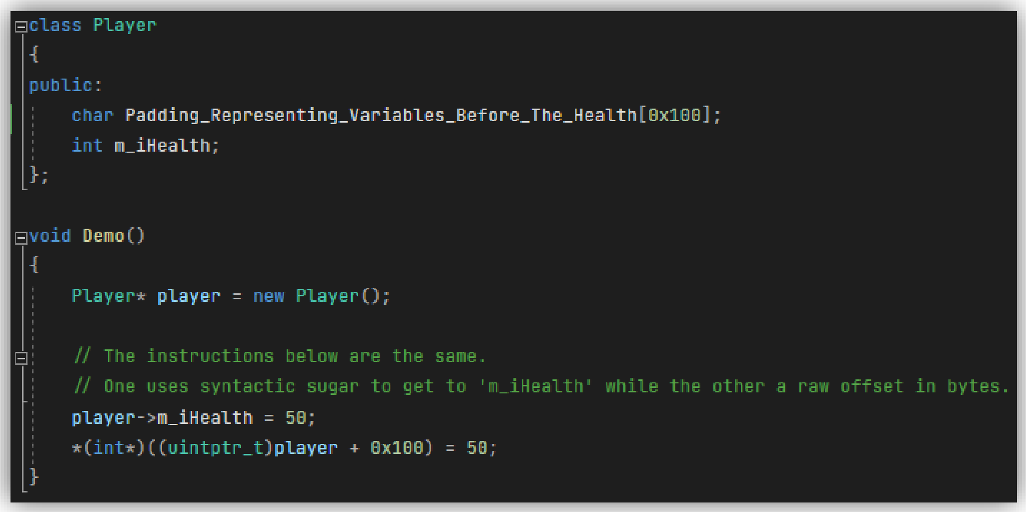
Memory Scan Options: All
Start: 0000000000000000
Stop: 00007fffffffffff
☑ Writable ☐ Executable
☐ CopyOnWrite
☐ Active memory only
☑ Fast Scan 4 ◉ Alignment ○ Last Digits
☐ Pause the game while scanning

Figure 7.1.1.1.1 – Scanning for the player's health value in Cheat Engine.

CE has found over eighty-nine thousand results that contain the value ‘100’ in the game. Ideally, the total results must be at least a two-digit value for proper inspection. The filtering can be easily done by repeatedly modifying the health value in-game, then performing consecutive scans.

7.1.1.2 Trial and error

After the filtering is done, there will only be values that actually represent the health. Getting down to one result is rarely possible because the game will naturally have multiple copies of the health variable. However, only the one that roots from the player object is of interest. To find it, it is crucial to understand how pointers and offsets work in C++ and Assembly as a mechanism to resolve the addresses of the member variables inside a class. Figure 7.1.1.2.1 illustrates this important concept.



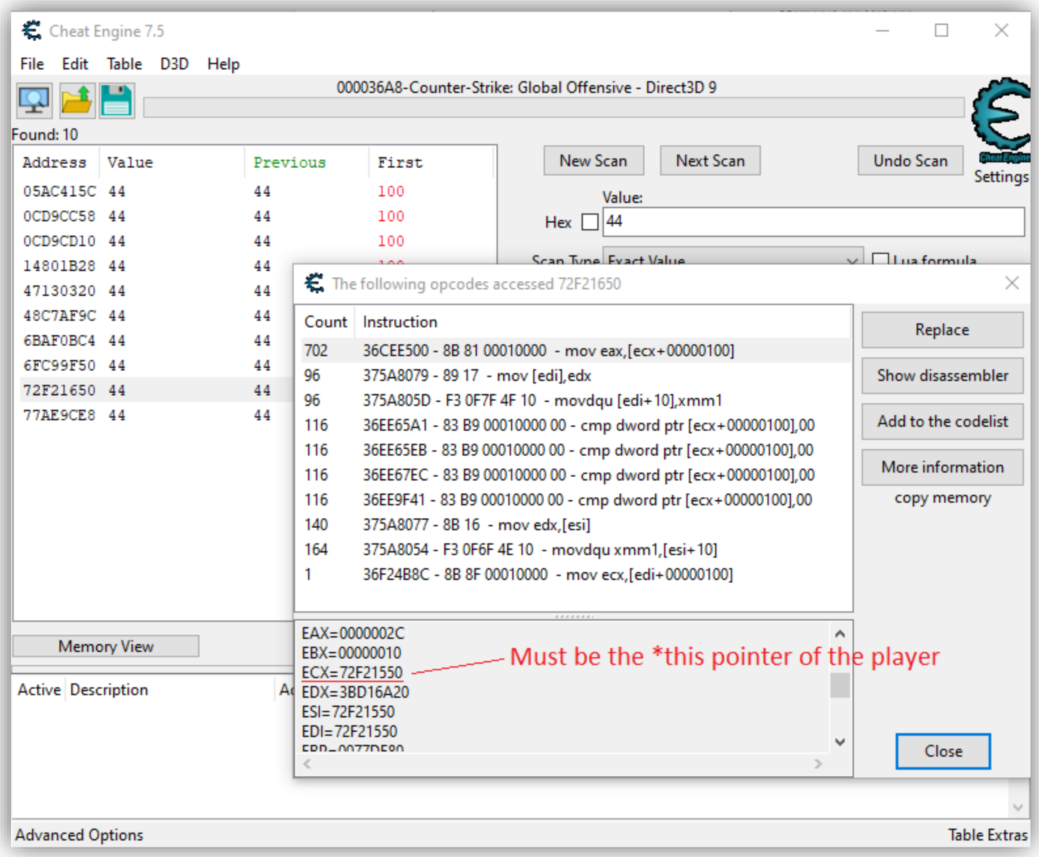
```cpp
class Player
{
public:
    char Padding_Representing_Variables_Before_The_Health[0x100];
    int m_iHealth;
};

void Demo()
{
    Player* player = new Player();

    // The instructions below are the same.
    // One uses syntactic sugar to get to 'm_iHealth' while the other a raw offset in bytes.
    player->m_iHealth = 50;
    *(int*)((uintptr_t)player + 0x100) = 50;
}
```

Figure 7.1.1.2.1 – Modifying a class variable in C++ without abstraction.

Consequently, every remaining filtered address must be scanned again for *in-code accesses* to understand whether the health address is accessed as a regular variable or from the actual player object. Figure 7.1.1.2.2 displays a bunch of ‘[ecx+00000100]’ instructions, which correlates to how an *object variable* would be accessed. The point of interest is the ECX register, which stores the player object’s address, while the value ‘0x100’ is the offset from the beginning of the class to the health.



Address	Value	Previous	First
05AC415C	44	44	100
0CD9CC58	44	44	100
0CD9CD10	44	44	100
14801B28	44	44	
47130320	44	44	
48C7AF9C	44	44	
6BAF0BC4	44	44	
6FC99F50	44	44	
72F21650	44	44	
77AE9CE8	44	44	

Value: 44

Count	Instruction
702	36CEE500 - 8B 81 00010000 - mov eax,[ecx+00000100]
96	375A8079 - 89 17 - mov [edi],edx
96	375A805D - F3 0F7F 4F 10 - movdqu [edi+10],xmm1
116	36EE65A1 - 83 B9 00010000 00 - cmp dword ptr [ecx+00000100],00
116	36EE65EB - 83 B9 00010000 00 - cmp dword ptr [ecx+00000100],00
116	36EE67EC - 83 B9 00010000 00 - cmp dword ptr [ecx+00000100],00
116	36EE9F41 - 83 B9 00010000 00 - cmp dword ptr [ecx+00000100],00
140	375A8077 - 8B 16 - mov edx,[esi]
164	375A8054 - F3 0F6F 4E 10 - movdqu xmm1,[esi+10]
1	36F24B8C - 8B 8F 00010000 - mov ecx,[edi+00000100]

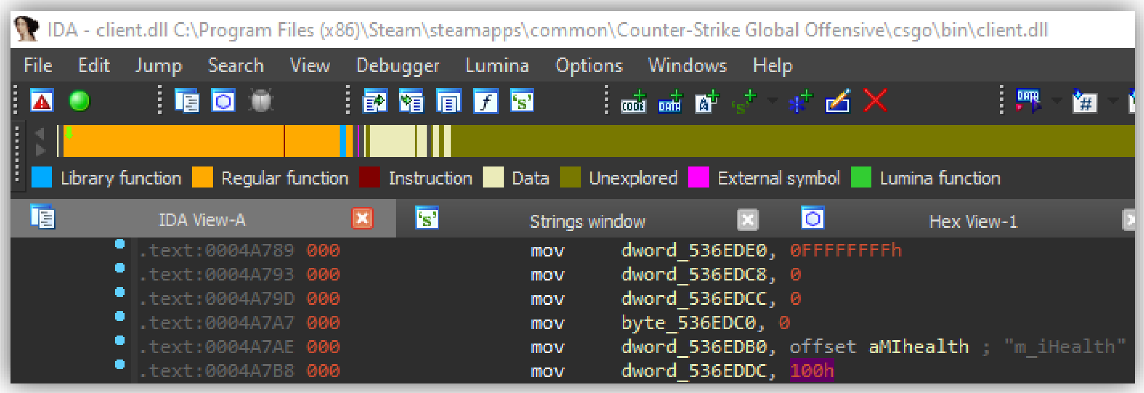
```
EAX=0000002C
EBX=00000010
ECX=72F21550
EDX=3BD16A20
ESI=72F21550
EDI=72F21550
```

Figure 7.1.1.2.2 – In-code accesses for the address ‘72F21650’.

As mentioned earlier, understanding Assembly is the first and most important requirement for successful reverse engineering.

7.1.1.3 Evaluation

Although the illustration above shows the correct result right away, eight prior addresses were also tested. Moreover, most of the trial and error was eliminated by knowing that by convention, in Assembly, the objects are almost always stored inside the ECX register. As per the offset of the health being ‘0x100’, it was found using IDA, as shown in Figure 7.1.1.3.1.


IDA - client.dll C:\Program Files (x86)\Steam\steamapps\common\Counter-Strike Global Offensive\csgo\bin\client.dll
File Edit Jump Search View Debugger Lumina Options Windows Help
Library function
Regular function
Instruction
Data
Unexplored
External symbol
Lumina function
IDA View-A
Strings window
Hex View-1

```
.text:0004A789 000                 mov     dword_536EDE0, 0FFFFFFFFh
.text:0004A793 000                 mov     dword_536EDC8, 0
.text:0004A79D 000                 mov     dword_536EDCC, 0
.text:0004A7A7 000                 mov     byte_536EDC0, 0
.text:0004A7AE 000                 mov     dword_536EDB0, offset aMIhealth ; "m_iHealth"
.text:0004A7B8 000                 mov     dword_536EDDC, 100h
```

Figure 7.1.1.3.1 – Locating the health offset using IDA.

Incidentally, 'm_iHealth' is a known netvar discovered as part of familiarizing with the CS: GO's game engine, Source. Studying the game engine's SDK before diving into reversing is a great tactic that can provide great insight.

7.1.1.4 Conclusion

Acquiring the address of the local player is a great start, as it can be used to reverse the structure that it belongs to using ReClass.Net; the offset of the health in the structure is already known. Because other players will have the same class object, it will only be enough to find their addresses and then apply offsets from the reversed structure, such as the health, to access the needed data.

7.2 The Overhaul

Initially, the project Reverse Engineering Shenanigans never had a fundamental objective besides engaging in reverse engineering activities on some application, preferably a game. It was due to the author's rather vague knowledge about the subject, and as a result, the initial choice fell on a game called Assault Cube. Breaking down an offline game that is relatively old, does not update, and weighs a couple of megabytes indubitably presents fewer challenges than a massive modern online game.

Consequently, after gaining some confidence, it was decided to take a well-calculated deviation toward a more challenging milestone and switch to CS: GO. Although the project started from ground zero, some sprints took less time, and it was easier to rework some of the features due to the pre-built foundation.

7.3 Sprint 0 – Preparatory Work

The initial sprint involved preparatory work of setting up the development environment and prerequisites needed to start the project. It included elaborating a well-thought project vision, setting up a GitHub version-controlled repository, configuring a Trello project backlog, and sketching a high-level plan using a Gantt Chart. The sprint also involved identifying a rough idea of the tools and technologies, though the definite choice was taken in the following sprint.

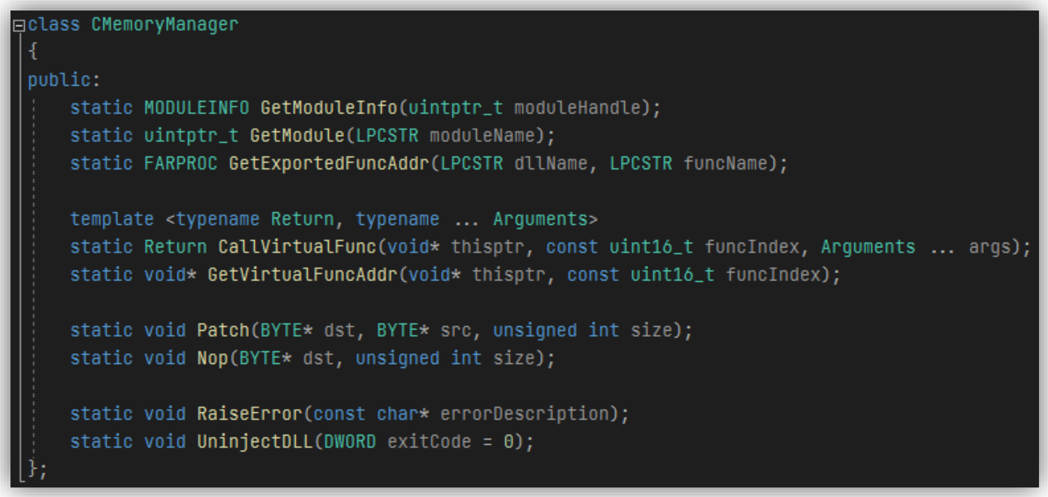
7.4 Sprint 1 – Planning

The first sprint focused on brainstorming the methodology, requirements, and objectives for the project, which were explained and justified earlier in the report. That involved drafting the MVP, creating the risk assessment, identifying the cheat's essential features, and conducting better research into the project's most suitable tools and technologies.

Backlog state: Appendix 13.2.1.1

7.5 Sprint 2 – Memory Manager

This sprint starts off the actual coding journey. It involved implementing a crucial memory manager class that stood as the foundation for implementing various cheat features. In broad terms, it includes the ability to read from and write to the game's memory, retrieve any information about the game modules, and find the memory addresses of the in-game functions. A snapshot of the class can be seen below.



```cpp
class CMemoryManager
{
public:
	static MODULEINFO GetModuleInfo(uintptr_t moduleHandle);
	static uintptr_t GetModule(LPCSTR moduleName);
	static FARPROC GetExportedFuncAddr(LPCSTR dllName, LPCSTR funcName);

	template <typename Return, typename ... Arguments>
	static Return CallVirtualFunc(void* thisptr, const uint16_t funcIndex, Arguments ... args);
	static void* GetVirtualFuncAddr(void* thisptr, const uint16_t funcIndex);

	static void Patch(BYTE* dst, BYTE* src, unsigned int size);
	static void Nop(BYTE* dst, unsigned int size);

	static void RaiseError(const char* errorDescription);
	static void UninjectDLL(DWORD exitCode = 0);
};
```

Figure 7.5.1 – The declaration of 'CmemoryMangaer' class.

Backlog state: Appendix 13.2.1.2

7.6 Sprint 3 – Hooking Library

The third sprint involved implementing a hooking mechanism to intercept the game's execution flow and redirect it toward the DLL's code. It is one of the most potent concepts available only in Internals. For example, a practical usage of it is hooking the function 'EndScene' from the game's graphics API, Direct3D, which marks the end of a rendering operation. Thus, putting the cheat's drawing functions right before it will allow it to stay synchronized with the game's rendering pipeline, significantly reducing jittering and unnecessary operations. There are several hooking mechanisms, though the most reliable is 'Trampoline', whose concept is depicted in the figure below.

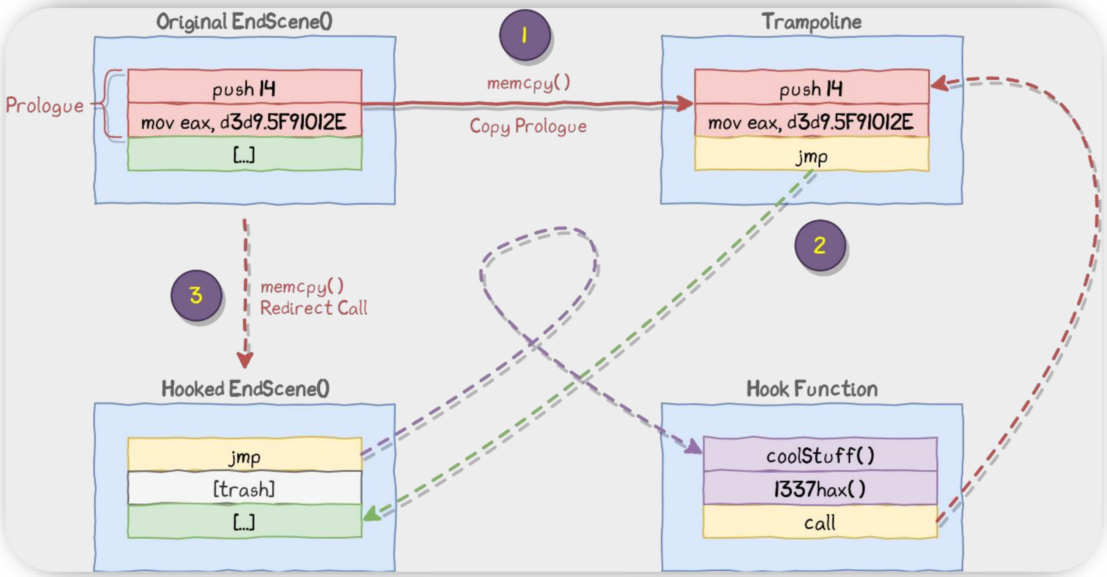


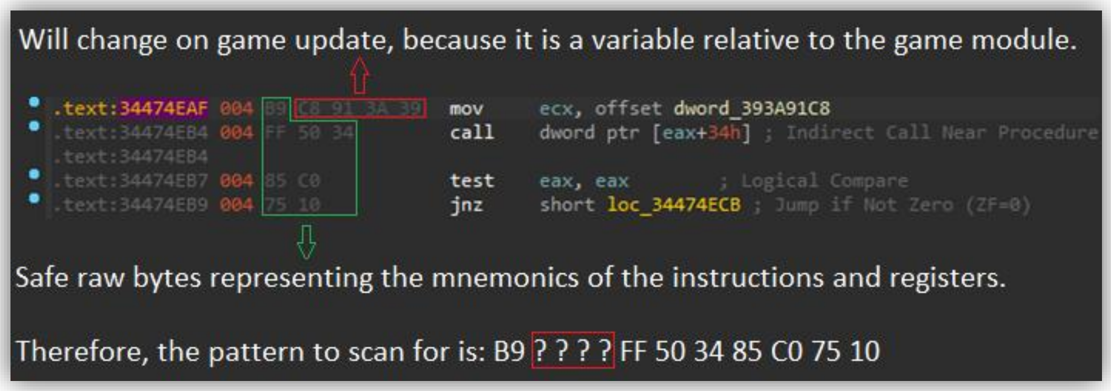
Figure 7.6.1 – The 'Trampoline' hooking concept.

Although the project's hooking class is functional, it does not account for every possible function structure and calling convention, as it would have taken too much time to implement. Therefore, it was decided to resort to a proper third-party hooking library called 'MinHook'.

Backlog state: Appendix 13.2.1.3

7.7 Sprint 4 – Pattern Scanner

The fourth sprint focused on implementing a pattern scanner that retrieves correct memory addresses even after a game update, effectively updating the cheat along with it. Essentially, it traverses the whole memory space and searches for a predetermined pattern containing the needed variable. However, the concept is never fail-proof, and there is always a tiny chance that a massive update might completely delete the instructions forming the pattern. An illustration of the concept can be seen below.



Will change on game update, because it is a variable relative to the game module.

```
.text:34474EAF 004 B9 C8 91 3A 39    mov     ecx, offset dword_393A91C8
.text:34474EB4 004 FF 50 34          call    dword ptr [eax+34h] ; Indirect Call Near Procedure
.text:34474EB4
.text:34474EB7 004 85 C0             test    eax, eax         ; Logical Compare
.text:34474EB9 004 75 10             jnz     short loc_34474ECB ; Jump if Not Zero (ZF=0)
```

Safe raw bytes representing the mnemonics of the instructions and registers.

Therefore, the pattern to scan for is: B9 ? ? ? ? FF 50 34 85 C0 75 10

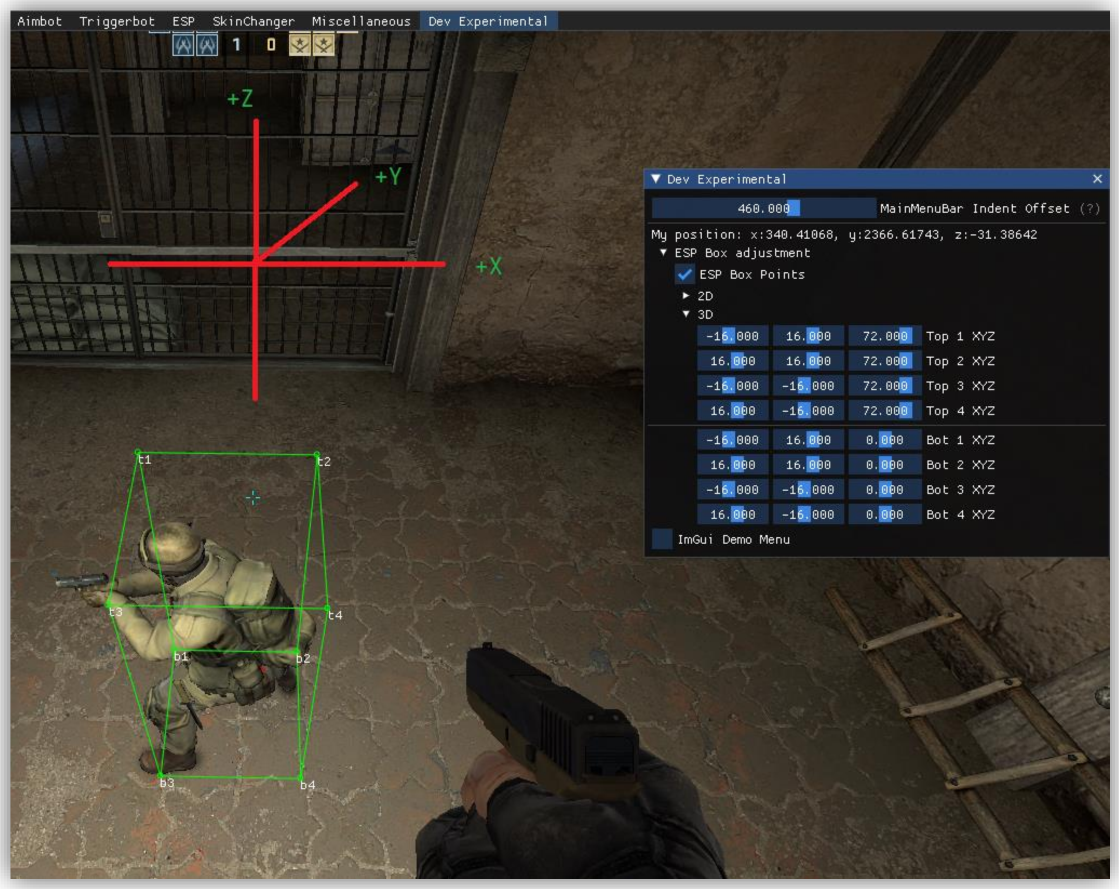
Figure 7.7.1 – The pattern scanner concept.

Backlog state: Appendix 13.2.1.4

7.8 Sprint 5 – MVP & First Usability Phase

The fifth sprint had accomplished the MVP, leading to the conduction of the first usability test. It involved reversing the Player structure, which introduced the first cheating feature, Extra-Sensory Perception. An ESP can include everything that can be located in the game or represents an entity. It is achieved by transforming a 3D coordinate vector from the game's world space into the screen's 2D space.

During this stage, the ESP only included snaplines from the player to the enemies' feet, though it improved over time by considering the feedback from the usability test. The final version, as seen in the figure below, even includes a self-configured axis-aligned bounding box (AABB) for the enemies.



```
▼ Dev Experimental
460.000   MainMenuBar Indent Offset (?)
My position: x:340.41068, y:2366.61743, z:-31.38642
▼ ESP Box adjustment
  ✓ ESP Box Points
    ▸ 2D
    ▾ 3D
      -16.000   16.000   72.000   Top 1 XYZ
       16.000   16.000   72.000   Top 2 XYZ
      -16.000  -16.000   72.000   Top 3 XYZ
       16.000  -16.000   72.000   Top 4 XYZ
      -16.000   16.000    0.000   Bot 1 XYZ
       16.000   16.000    0.000   Bot 2 XYZ
      -16.000  -16.000    0.000   Bot 3 XYZ
       16.000  -16.000    0.000   Bot 4 XYZ
☐ ImGui Demo Menu
```

Figure 7.8.1 – Configuration of the 3D ESP box.

Backlog state: Appendix 13.2.1.5

7.9 Sprint 6 – Neutral features

The sixth sprint involved the development of two neutral features, Bunny Hop and Crosshair, described in section 2.1.1.3. The sprint was fast-paced and did not demand much effort. The Bunny Hop feature required understanding whether the player is on the ground, which would perform a jump based on that. The Crosshair is simply a statically rendered marker in the middle of the screen.

Backlog state: Appendix 13.2.1.6

7.10 Sprint 7 – Trigger Bot & Anti-Flash

The seventh sprint required more reversal to achieve the Trigger Bot and Anti-Flash features. Implementing the Anti-Flash was reasonably straightforward, requiring scanning for differences indicating whether the blindness effect was applied. However, the Trigger Bot presented some difficulties. It was implemented using the game’s crosshair indicator to determine whether the player is aiming at an enemy, as illustrated in the figure below.

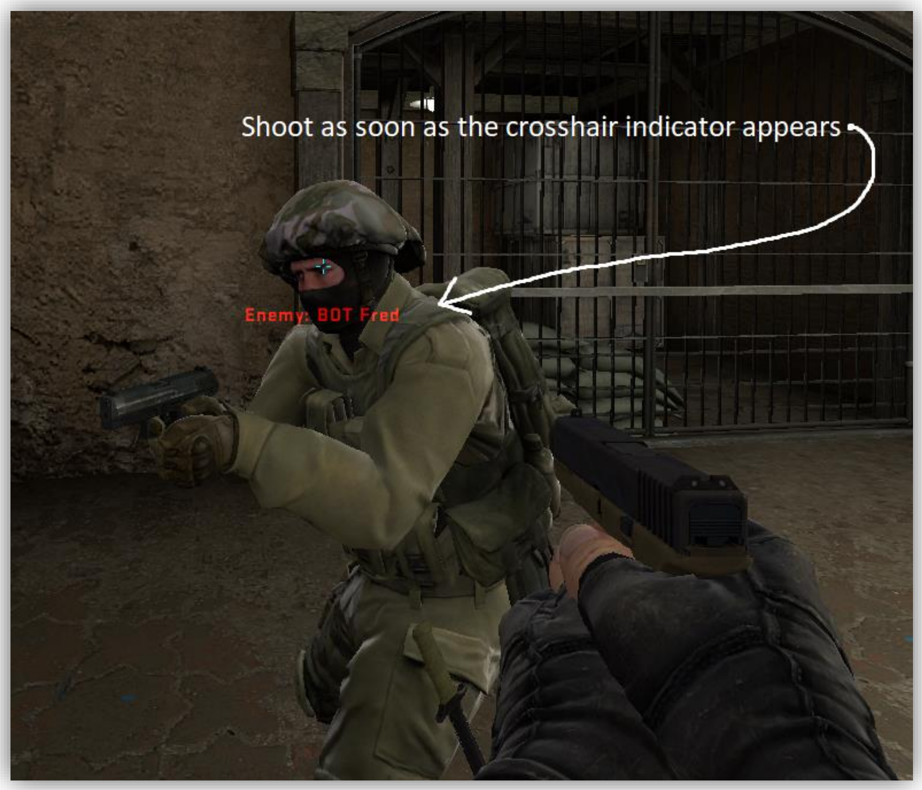


Figure 7.10.1 – The Trigger Bot concept.

Unfortunately, this method has some delay. The most performant and instant method would be to directly get the result from the original function that casts rays from the player toward his aiming direction to understand what it collides with. The function should naturally be present in any shooter's game engine; however, the attempts at finding it were unsuccessful.

Backlog state: Appendix 13.2.1.7

7.11 Sprint 8 – Skin Changer & GUI (Direct3D)

The eighth sprint is characterized by the biggest change, introducing the Skin Changer and transitioning from CLI to a graphical user interface, as requested during the first usability test. Implementing the Skin Changer involved finding the memory address of the currently held weapon and changing the paint kit number bound to it according to the desired skin.

Although drawing the cheat's GUI using the Direct3D API was a major step forward, it was sooner replaced due to its complications.

Backlog state: Appendix 13.2.1.8

7.12 Sprint 9 – Aim Bot & Second Usability Phase

The ninth sprint involved implementing the last cheating feature, which prepared the software for the final usability phase. Although an Aim Bot simply requires finding the memory addresses of the enemies' positions, the actual implementation of the aiming process proved challenging.

Generally, there are two ways of achieving the aiming functionality. The easy but unprecise method involves transforming the 3D in-game coordinates to 2D screen coordinates, then simulating a mouse movement to them. The problematic but pixel-perfect technique used in the cheat requires trigonometry calculations to modify the player's Euler angles. The concept is illustrated in the figure below.

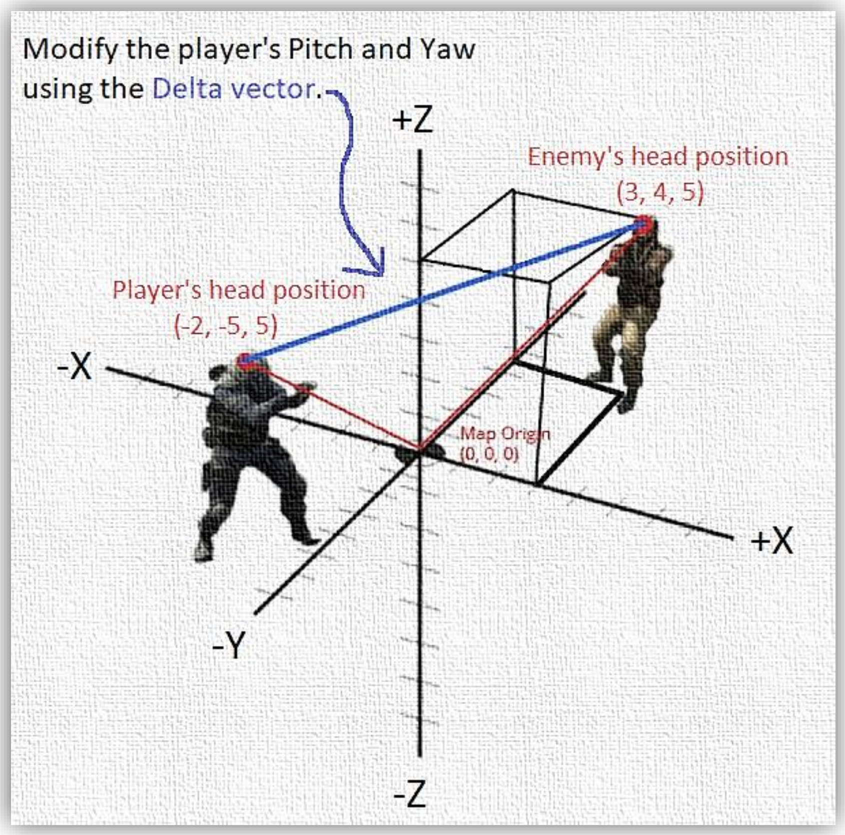


Figure 7.12.1 – Sketch of two player's positions on a coordinate system.

Backlog state: Appendix 13.2.1.9

7.13 Sprint 10 – Injector & GUI (ImGui)

The final sprint initially focused on making corrections from the last usability test and rehauling the cheat's GUI with the help of ImGui, representing a shockingly powerful assortment of GUI functionality and customization possibilities encapsulated in a mere library. However, at the last minute, it was also decided to implement a personal Injector due to the public alternatives being either closed-source or having unsuitable licensing to distribute it as part of this project.

An Injector simply facilitates loading a DLL inside a target process, as it cannot function independently. Injectors can achieve more than meets the eye and be very intricate. However, for the sake of this project, the most basic version was developed that creates a thread in the target process and forces it to call the function 'LoadLibraryA', which should in turn load the DLL. As a side note, the term 'Injector' is merely a classifying term for the aforementioned actions used in the hacking community. Nonetheless, they are all effectively executing code injection, which is a recognized OWASP security concern (Zhong et al., n.d.).

This sprint also focused on implementing a recommended feature called Silent Aim, which was identified in the last usability test. Unlike a regular Aim Bot, it gives the appearance that the player is not aiming at the enemy and instead has bullets that appear to teleport to the target. That is achieved by canceling multiple in-game animations that would otherwise delay the aiming process. However, locating the in-game code responsible for character animations presented difficulties, so the feature could not be implemented.

Backlog state: Appendix 13.2.1.10

8. Testing

Testing and gaining feedback is another essential aspect of Agile, which is indubitably a necessity for this project. Frequent testing is essential to reverse engineering because the complexity of the process is never without uncertainties and doubt. It is among the top strategies for ensuring software refinement and resilience as it develops, which is why the project has employed different types and stages of testing.

8.1 Memory Testing

Handling raw memory without awareness of its internal function and purpose conveys as much hope as defusing a bomb without an instruction manual. The software is constantly accessing the game's memory. Due to its dynamic nature, modifying even a bit off of data or using a data type other than what the game intends makes it very prone to undefined behavior and crashes, just like one tiny wire can set off a bomb.

Although the software is intended only to read and modify data tested during the reversal process and is allegedly safe, it still has safeguards such as memory validation and security checks where appropriate. These refer to ‘try-catch’ statements, ‘nullptr’ checks, ‘VirtualQuery’ for querying memory-related information, and ‘VirtualProtect’ for extra safety during memory modifications.

8.1.1 Pattern Scanning

The pattern scanner resembles unit testing in a way but is tailored for memory testing. It is used to locate the necessary offsets and addresses for the cheat, indicating its ability to function correctly. Pattern scanning happens during the software’s initialization, and if any of the patterns fail, the software will shut down.

8.2 Manual / Functional Testing

Unlike developing regular software, which could allow the implementation of an extensive feature set before testing due to either being able to follow a guide or having prior experience, developing cheating software is different. There is never documentation to put trust in, and the development is often about conducting heavy research, compiling multiple clues, and making educated guesses.

Consequently, due to its high susceptibility to errors, the software has undergone constant manual testing in different in-game scenarios for even the smallest changes, let alone feature implementations. Moreover, it tended to break on different build configurations in Visual Studio. For example, it has drained a significant amount of time figuring out that some unbeknown compiler setting in Release mode was tempering with some of the cheat’s data.

8.3 Usability Testing

Usability testing was conducted to ensure the application’s user-friendliness and understand whether the users’ expectations are fulfilled, even though the software is not supposed to solve any problems per se.

As anticipated, organizing the testing proved to be daunting due to the skepticism and fear among potential participants, who were concerned about the risk of account bans even after explaining the possibility of playing in an *unsecured mode*. As a result, the volunteers were mostly the author’s acquaintances, recruiting eight and eleven participants for the first and second usability phases, respectively. The sample’s demographics consisted exclusively of male participants aged at least seventeen, which corresponds to the game’s rating on the Entertainment Software Rating Board (ESRB, n.d.). The cheat’s target audience is assumed to have prior experience with the game, which had aligned with every participant. Otherwise, using the software without knowing what it is designed for would not make much sense.

The entire procedure was conducted verbally through Discord and involved soliciting feedback on the application’s usability and gathering recommendations for improvement. For a more accurate assessment of the application’s performance, users were not given any indications apart from injecting the software into the game, which is a somewhat abstruse task.

8.3.1 Phase 1

The first usability test was conducted immediately after the software's objectives had reached the MVP's requirements. The feedback results are illustrated in the figure below.

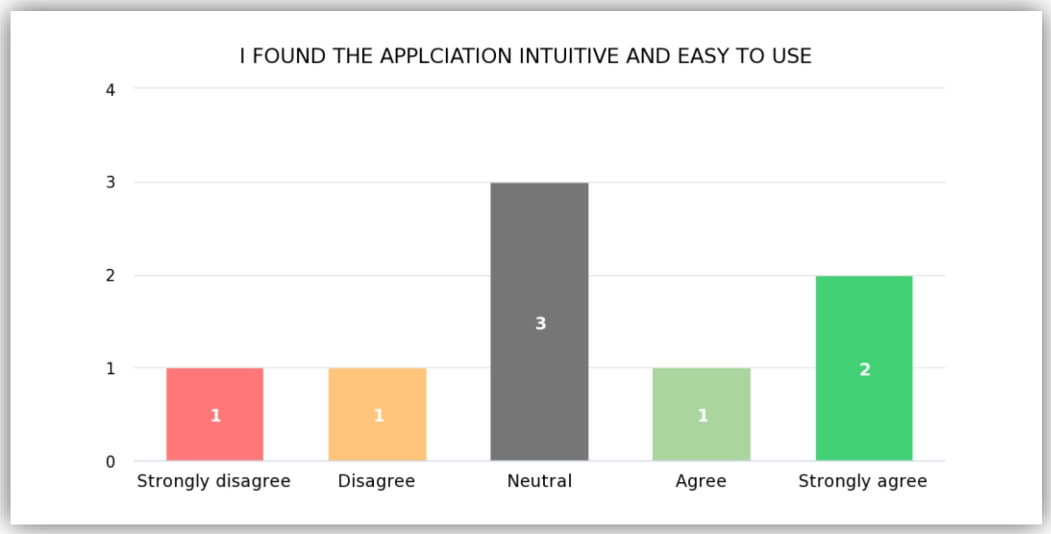


Figure 8.3.1.1 – Phase 1 questionnaire results.

The questionnaire results indicate that the software averages more neutral and negative ratings than positive ones. The neutral ratings can be attributed to the software's limited functionality and navigation options at the time. The negative ratings were due to the users' dissatisfaction with the cheat's CLI, which required them to switch out of the game to navigate it.

Many of the recommendations received were related to the adoption of a GUI, which prompted more serious consideration of this option. Additionally, several users suggested adding more ESP options and other features that were already planned for implementation.

8.3.2 Phase 2

The second usability phase was conducted right after achieving every set objective. The figure below depicts the analysis results.

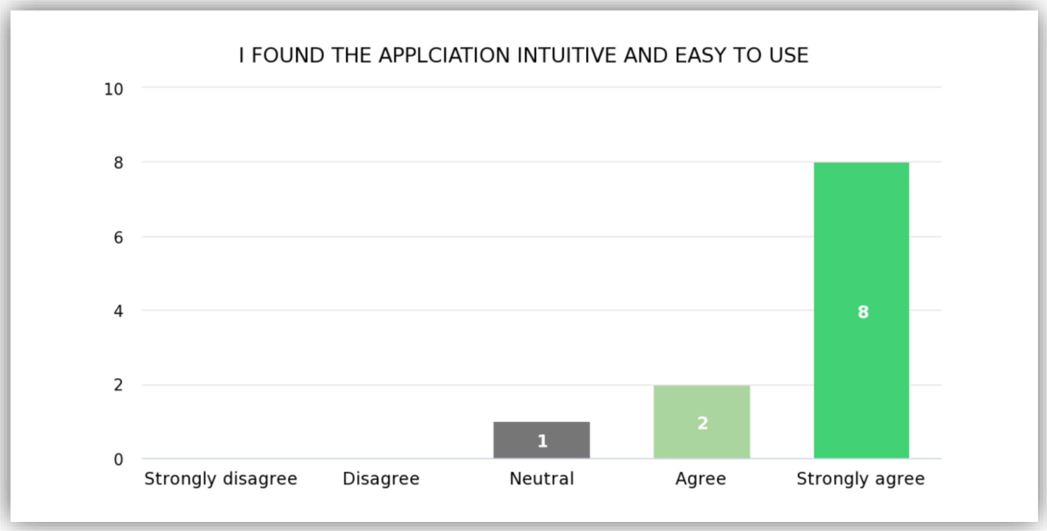


Figure 8.3.2.1 – Phase 2 questionnaire results.

The results starkly contrast the previous test, with markedly more positive outcomes. Multiple participants attributed this improvement to the introduction of a graphical user interface. However, some participants still recommended changes to the GUI. As a result, it was decided to switch to ImGui, which resolved all issues.

9. End-Project Report

9.1 Summary

Overall, the project was a success *considering the heights it aimed for*. However, it should be noted that cheats have too much room for improvement to achieve perfection, which is a matter of time and experience. Nevertheless, the final product is a fully functional cheat with various features as initially intended. The most significant outcome of this project is the knowledge and experience gained in reverse engineering. Additionally, working in a professional setting and following a well-defined methodology, including considering any implications with the software and managing a complex software development project, has further expanded the author's skillset.

9.2 Objectives Review

Besides exploring the fundamentals of reverse engineering, the project aimed to implement several objectives categorized in neutral and assisting cheating features. Although each feature was implemented, there is still vast room for improvement, considering the possibilities that reverse engineering offers for 'modding' the game. The project aimed to achieve each feature's *most basic and raw version* considering the time constraints and unfamiliarity with the subject matter. For instance, the Skin Changer only allows for weapon appearance configuration, while it could be extended to every other asset in the game. That applies to every feature, as their configurability is practically limitless. Additionally, some objectives were added based on usability recommendations, such as the GUI, while others, as previously mentioned, could not be implemented.

9.3 Changes Review

During the development process, the project underwent several significant changes that were deliberately analyzed and ultimately positively impacted the project’s outcomes.

The most significant change in the project was a transition to a different game, as discussed earlier in the report. Specifically, the project was overhauled to target CS: GO, which provided more room for improvement and was a game the author was more familiar with. As part of this transition and upon gaining more knowledge, the cheat’s architecture was also changed from External to Internal access, as these types of cheats offered more powerful advantages. Another necessary change was the transition from a CLI to a GUI, which significantly improved the usability of the cheat. Finally, to increase the project’s credibility, it was decided to avoid including a closed-source third-party Injector as part of its deliverables and instead develop a personalized one.

10. Project Post-Mortem

10.1 Development Process Evaluation

Despite the challenges encountered, the development process was a highly stimulating aspect of the project. Rather than leading to discouragement, these challenges served as a source of motivation for the author to persist in researching and experimenting with the project's concept. Witnessing the project attain objectives that *were once a mystery* is deeply gratifying. Cracking the black box open and figuring out its inner workings is the most rewarding aspect of reverse engineering. Furthermore, utilizing various tools, following a robust methodology, and adhering to the software development lifecycle further enriched the process.

10.2 Developer Performance Evaluation

As the project’s author, I confidently assert that I performed well throughout its development. It comes down to the motivation and commitment to the project. The opportunity to delve deeply into reverse engineering as part of a large-scale project served as an immense inspiration. A significant amount of time had been allocated for the project’s achievement during its development phase and before it had even commenced. The project’s idea goes way back, demanding extensive preparation before it could be undertaken. With enough dedication and collected experience, it finally came to fruition.

Despite my efforts, there were obstacles and periods of stagnation during the journey, primarily due to the project's low-level nature. The mistakes were a great learning point, and the challenges presented a trial of my adaptability. Notably, the initial deliberation regarding the game choice, which was ultimately switched. While it could be viewed as a misstep and a waste of time, it served as a solid foundation for future development. I would rather work in small but steady increments than recklessly aim for a more significant but uncertain goal.

10.3 Future Work

As previously stated, this project serves as a proof of concept and highlights the limitless potential for improvement in cheating software. Given my profound interest in reverse engineering and this project, I intend to continue working on it to enhance my skills further.

One potential avenue for future exploration is reverse engineering the game's anti-cheat module. However, the challenge lies in the multiple encryption and hashing methods used to obfuscate its code, which would require extensive experience and research. Another prospect is adding cross-platform support for Linux. Although it is not as challenging, adapting to its distinct memory layout and API would require significant time.

Finally, the upcoming rebranding of the game CS: GO to Counter-Strike 2 in the summer of 2023 presents an exciting opportunity for this project. The game will be based on Valve's newer engine, Source 2, which will shift its architecture to x64. It is remarkable, as the newer architecture *had been avoided* during the initial game selection due to its notable complications over x32. However, and hopefully, after gaining so much experience, exploring the intricacies of the new territory will feel like a breeze.

11. Conclusion

In conclusion, this dissertation has chronicled the journey of a project from its inception to realization. The project was approached with the utmost seriousness and responsibility, successfully meeting the established requirements and objectives.

Given its scale, the project was managed with professionalism in accordance with the standards expected of a Computer Scientist. That involved employing a rigorous methodology, adhering to project management principles, conducting research and user testing, and developing high-quality software.

Undertaking a project in a subject area I am passionate about has made the acquisition of skills and knowledge even more rewarding. This experience has laid the foundation for reverse engineering expertise, which will undoubtedly prove valuable in my future career.

12. References

Agile Alliance. (2001). Manifesto for Agile Software Development, Agile Alliance. Available at: https://agilemanifesto.org/ (Accessed: April 15, 2023).

Agile Alliance. (n.d.a) Agile 101, Agile Alliance. Available at: https://www.agilealliance.org/agile101/ (Accessed: April 15, 2023).

Agile Alliance. (n.d.b) Minimum Viable Product (MVP), Agile Alliance. Available at: https://www.agilealliance.org/glossary/mvp (Accessed: April 18, 2023).

Ajmal, A. et al. (2022) Stress-relieving video game and its effects: A POMS case study, Computational Intelligence and Neuroscience. Hindawi. Available at: https://www.hindawi.com/journals/cin/2022/4239536/ (Accessed: April 2, 2023).

Brown, K. (2019) What Is GitHub, and What Is It Used For?, How. How-To Geek. Available at: https://www.howtogeek.com/180167/htg-explains-what-is-github-and-what-do-geeks-use-it-for/ (Accessed: April 30, 2023).

Copyright, Designs and Patents Act. (1998). Chapter 48. Available at: https://www.legislation.gov.uk/ukpga/1988/48/contents (Accessed: April 25, 2023).

Edwards, B. (2021) What is the Konami Code, and how do you use it?, How-To Geek. How-To Geek. Available at: https://www.howtogeek.com/659611/what-is-the-konami-code-and-how-do-you-use-it/ (Accessed: April 4, 2023).

ESRB (n.d.) Ratings guide. Available at: https://www.esrb.org/ratings-guide/ (Accessed: May 1, 2023)

Lutkevich, B. (2021) What is Reverse-engineering? How Does It Work?, TechTarget. TechTarget. Available at: https://www.techtarget.com/searchsoftwarequality/definition/reverse-engineering (Accessed: April 6, 2023).

Martins, J. (2022) What is Kanban? Here’s what your Agile team needs to know, Asana. Asana. Available at: https://asana.com/resources/what-is-kanban (Accessed: April 22, 2023).

Microsoft Developer Network. (2021). NTQuerySystemInformation function. Available at: https://learn.microsoft.com/en-us/windows/win32/api/winternl/nf-winternl-ntquerysysteminformation (Accessed: April 27, 2023).

Moore, J. (2022) PUBG Mobile Cheat Makers Ordered to Pay $10 Million in Damages, IGN. IGN. Available at: https://www.ign.com/articles/pubg-mobile-cheat-makers-pay-10-million-damages (Accessed: April 26, 2023).

Peil, C. (2019) The Legalities of Reverse Engineering, peillaw. Available at: https://peillaw.com/the-legalities-of-reverse-engineering/ (Accessed: April 25, 2023).

Prado, C.G. and Erickson, J. (2018) Solving Ad-hoc Problems with Hex-Rays API, FireEye. FireEye. Available at: https://web.archive.org/web/20220602140613/https://www.fireeye.com/blog/threat-research/2018/04/solving-ad-hoc-problems-with-hex-rays-api.html (Accessed: April 23, 2023).

Preston, R. (2023) What Is Assembly Language? (With Components and Example), indeed. indeed. Available at: https://www.indeed.com/career-advice/career-development/what-is-assembly-language (Accessed: April 23, 2023).

Ritterfeld, U. and Weber, R., 2012. Video games for entertainment and education. Playing video games: Motives, responses, and consequences, pp.399-413. ISBN: 9781135257477

Valve Corporation. (2012). Counter-Strike: Global Offensive [Video game]. Steam. Available at: https://store.steampowered.com/app/730/CounterStrike_Global_Offensive/ (Accessed: April 7, 2023).

Valve Corporation. (2023a). Valve Anti-Cheat (VAC) System. Steam Support. Available at: https://help.steampowered.com/en/faqs/view/571A-97DA-70E9-FF74 (Accessed: April 9, 2023).

Valve Corporation. (2023b). CS:GO - Overwatch System. Steam Support. Available at: https://help.steampowered.com/en/faqs/view/65DA-BD12-0DE9-9853 (Accessed: April 12, 2023).

Valve Corporation. (2023c). Steam Subscriber Agreement. Available at: https://store.steampowered.com/subscriber_agreement (Accessed: April 25, 2023).

What Is a High-Level Project Plan? Definition and Importance (2023) indeed. Indeed Editorial Team. Available at: https://www.indeed.com/career-advice/career-development/high-level-project-plan (Accessed: April 20, 2023).

What is Trello? (2022) Atlassian Support. Atlassian. Available at: https://support.atlassian.com/trello/docs/what-is-trello/ (Accessed: April 28, 2023).

Zhong, W. et al. (no date) Code Injection, OWASP. Available at: https://owasp.org/www-community/attacks/Code_Injection (Accessed: April 24, 2023).

13. Appendices

13.1 Appendix 1: User Guide

Note
The software refers to both the cheat '.dll' and the Injector executable.
The game refers to Counter-Strike: Global Offensive (CS: GO)

13.1.1 Minimum System Requirements

The software

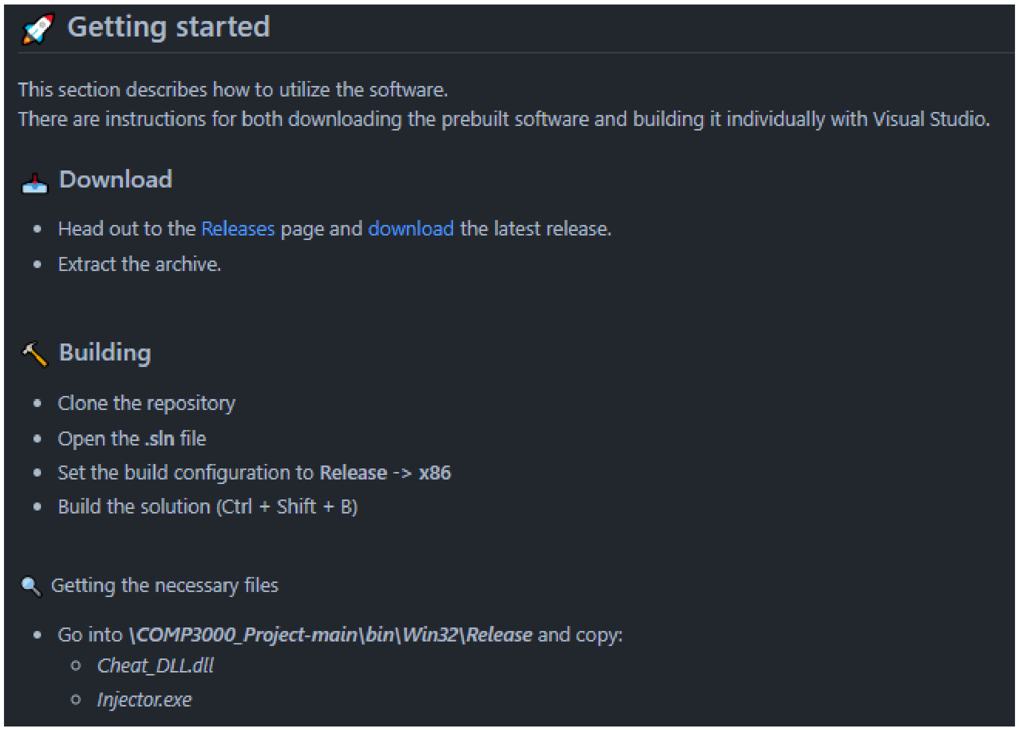
- Operrating system: Windows 10

The game

- OS: Windows® 7/Vista/XP
- Processor: Intel® Core™ 2 Duo E6600 or AMD Phenom™ X3 8750
- Memory: 2 GB RAM
- Graphics: Video card must be 256 MB or more and should be a DirectX 9-compatible with support for Pixel Shader 3.0
- DirectX: Version 9.0c
- Storage: 15 GB available space

13.1.2 Installation

The software
The GitHub repository offers both a pre-compiled version of the software as well as instructions for compiling yourself. Below is a snapshot from the GitHub:


Getting started

This section describes how to utilize the software.
There are instructions for both downloading the prebuilt software and building it individually with Visual Studio.

Download

- Head out to the Releases page and download the latest release.
- Extract the archive.

Building

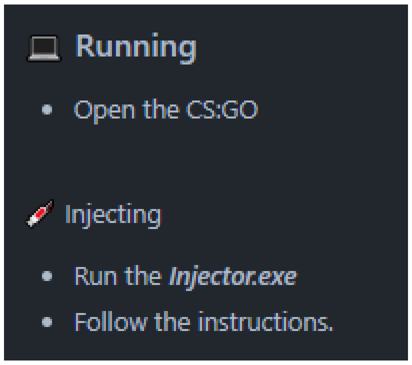
- Clone the repository
- Open the .sln file
- Set the build configuration to **Release** -> **x86**
- Build the solution (Ctrl + Shift + B)

Getting the necessary files

- Go into ***\COMP3000_Project-main\bin\Win32\Release*** and copy:
  - *Cheat_DLL.dll*
  - *Injector.exe*

The game
CS: GO can be installed from Steam. Follow the instructions here: link

13.1.3 Running the Application

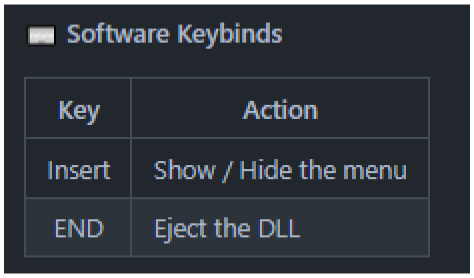

Running

- Open the CS:GO

Injecting

- Run the ***Injector.exe***
- Follow the instructions.

13.1.4 Keybindings

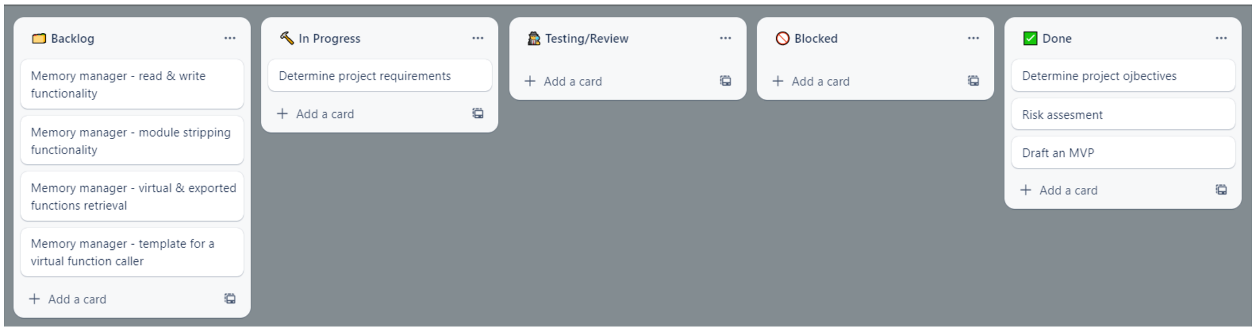

Software Keybinds

Key	Action
Insert	Show / Hide the menu
END	Eject the DLL

13.2 Appendix 2: Project Management

13.2.1 Trello Boards

2.1.1 Sprint 1



Backlog
- Memory manager - read & write functionality
- Memory manager - module stripping functionality
- Memory manager - virtual & exported functions retrieval
- Memory manager - template for a virtual function caller

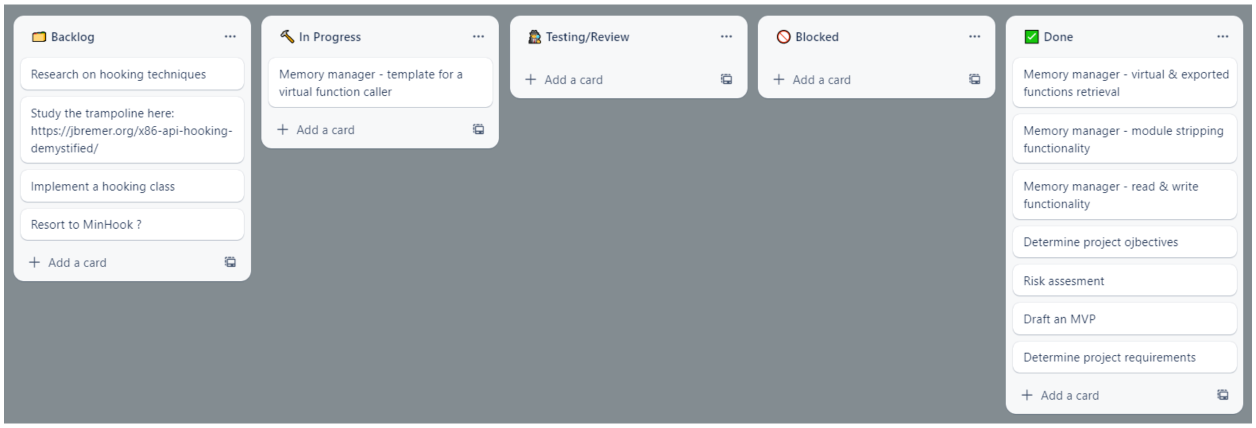
In Progress
- Determine project requirements

Testing/Review

Blocked

Done
- Determine project ojbectives
- Risk assesment
- Draft an MVP

2.1.2 Sprint 2



Backlog
- Research on hooking techniques
- Study the trampoline here: https://jbremer.org/x86-api-hooking-demystified/
- Implement a hooking class
- Resort to MinHook ?

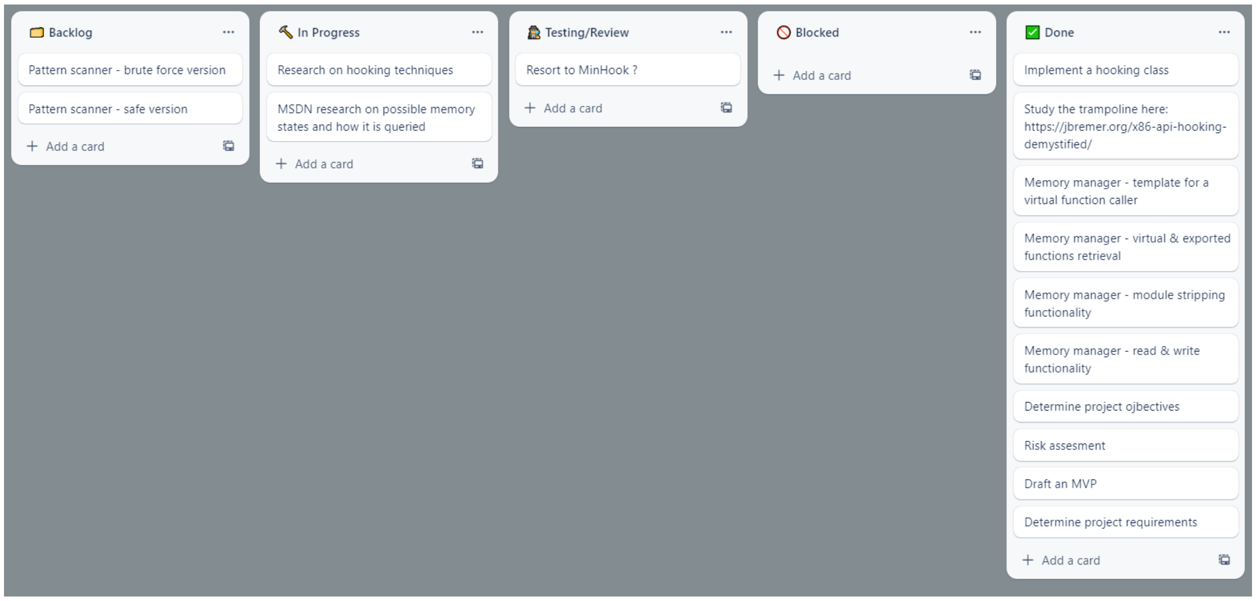
In Progress
- Memory manager - template for a virtual function caller

Testing/Review

Blocked

Done
- Memory manager - virtual & exported functions retrieval
- Memory manager - module stripping functionality
- Memory manager - read & write functionality
- Determine project ojbectives
- Risk assesment
- Draft an MVP
- Determine project requirements

2.1.3 Sprint 3



Backlog
- Pattern scanner - brute force version
- Pattern scanner - safe version
- Add a card

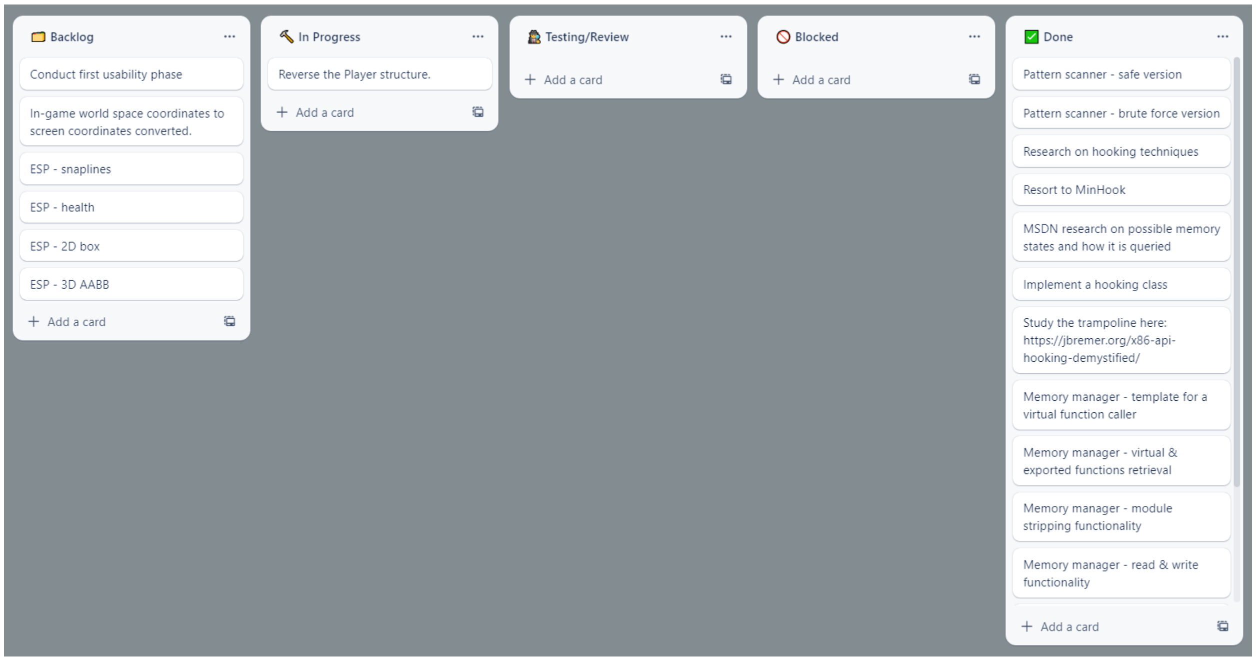
In Progress
- Research on hooking techniques
- MSDN research on possible memory states and how it is queried
- Add a card

Testing/Review
- Resort to MinHook ?
- Add a card

Blocked
- Add a card

Done
- Implement a hooking class
- Study the trampoline here: https://jbremer.org/x86-api-hooking-demystified/
- Memory manager - template for a virtual function caller
- Memory manager - virtual & exported functions retrieval
- Memory manager - module stripping functionality
- Memory manager - read & write functionality
- Determine project ojbectives
- Risk assesment
- Draft an MVP
- Determine project requirements
- Add a card

2.1.4 Sprint 4



Backlog
- Conduct first usability phase
- In-game world space coordinates to screen coordinates converted.
- ESP - snaplines
- ESP - health
- ESP - 2D box
- ESP - 3D AABB
- Add a card

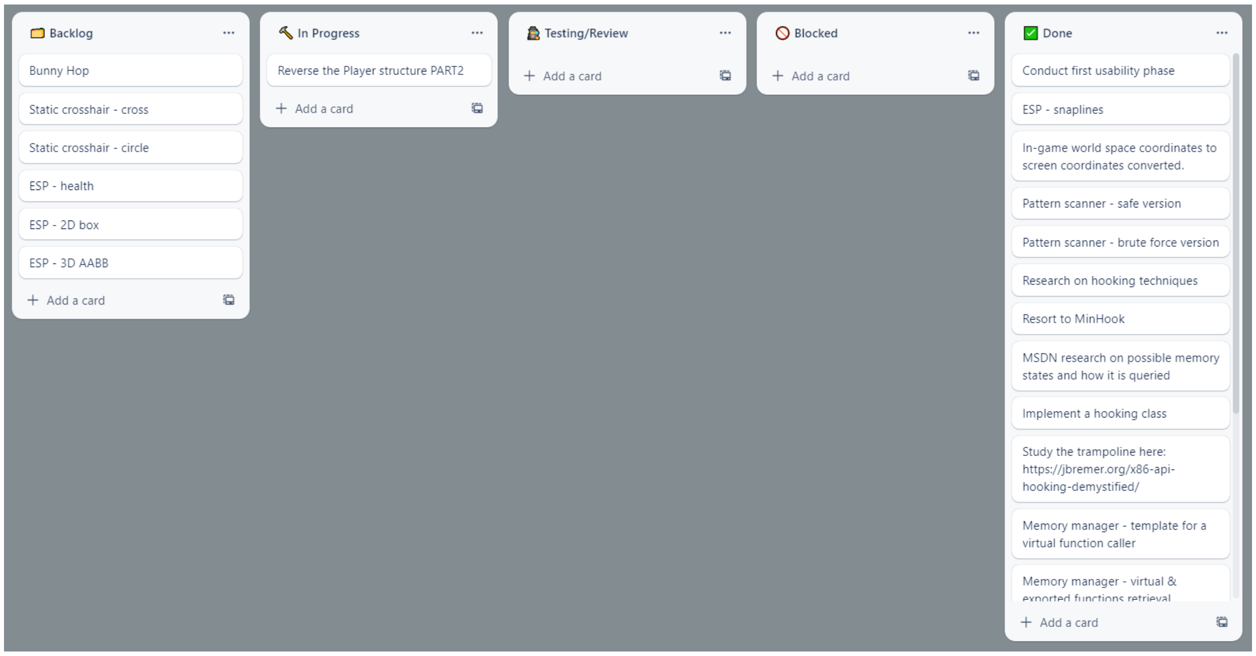
In Progress
- Reverse the Player structure.
- Add a card

Testing/Review
- Add a card

Blocked
- Add a card

Done
- Pattern scanner - safe version
- Pattern scanner - brute force version
- Research on hooking techniques
- Resort to MinHook
- MSDN research on possible memory states and how it is queried
- Implement a hooking class
- Study the trampoline here: https://jbremer.org/x86-api-hooking-demystified/
- Memory manager - template for a virtual function caller
- Memory manager - virtual & exported functions retrieval
- Memory manager - module stripping functionality
- Memory manager - read & write functionality
- Add a card

2.1.5 Sprint 5



Backlog
- Bunny Hop
- Static crosshair - cross
- Static crosshair - circle
- ESP - health
- ESP - 2D box
- ESP - 3D AABB
- Add a card

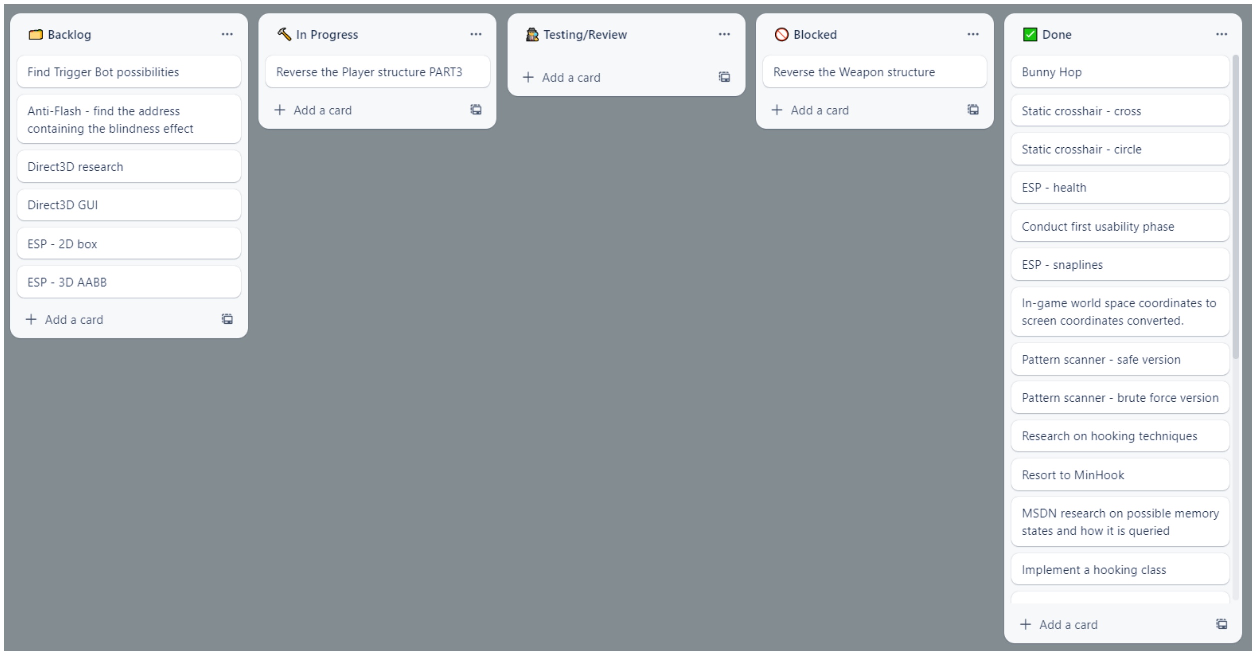
In Progress
- Reverse the Player structure PART2
- Add a card

Testing/Review
- Add a card

Blocked
- Add a card

Done
- Conduct first usability phase
- ESP - snaplines
- In-game world space coordinates to screen coordinates converted.
- Pattern scanner - safe version
- Pattern scanner - brute force version
- Research on hooking techniques
- Resort to MinHook
- MSDN research on possible memory states and how it is queried
- Implement a hooking class
- Study the trampoline here: https://jbremer.org/x86-api-hooking-demystified/
- Memory manager - template for a virtual function caller
- Memory manager - virtual & exported functions retrieval
- Add a card

2.1.6 Sprint 6



Backlog
- Find Trigger Bot possibilities
- Anti-Flash - find the address containing the blindness effect
- Direct3D research
- Direct3D GUI
- ESP - 2D box
- ESP - 3D AABB
- Add a card

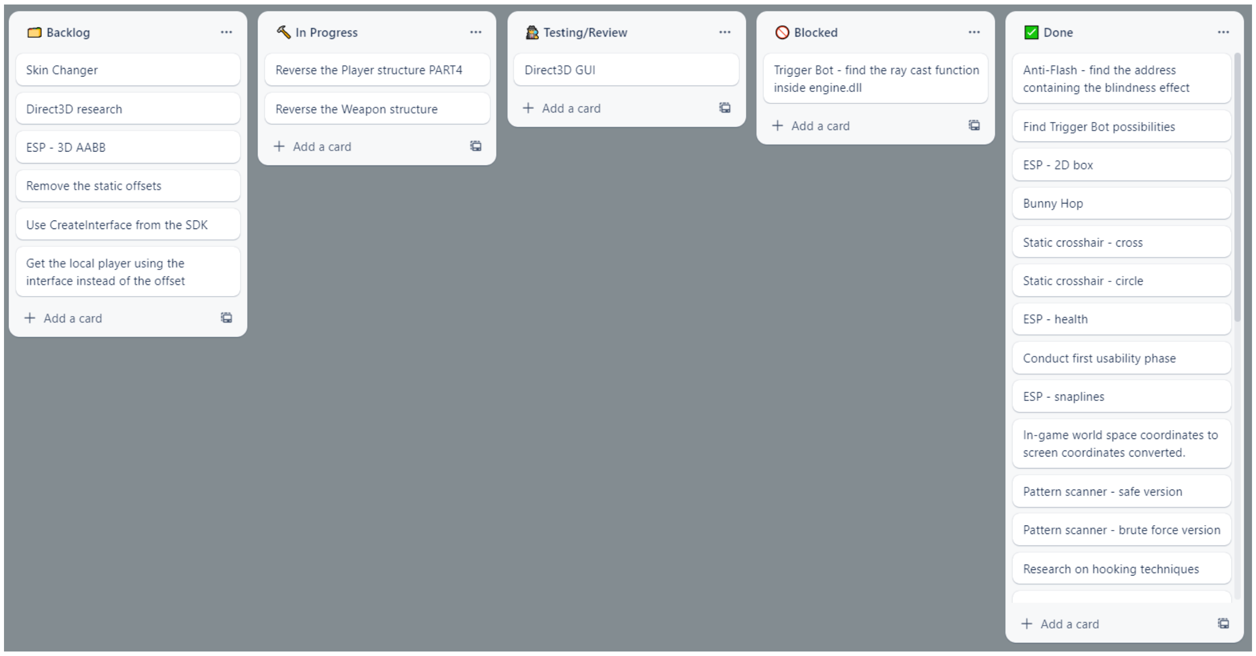
In Progress
- Reverse the Player structure PART3
- Add a card

Testing/Review
- Add a card

Blocked
- Reverse the Weapon structure
- Add a card

Done
- Bunny Hop
- Static crosshair - cross
- Static crosshair - circle
- ESP - health
- Conduct first usability phase
- ESP - snaplines
- In-game world space coordinates to screen coordinates converted.
- Pattern scanner - safe version
- Pattern scanner - brute force version
- Research on hooking techniques
- Resort to MinHook
- MSDN research on possible memory states and how it is queried
- Implement a hooking class
- Add a card

2.1.7 Sprint 7



Backlog
- Skin Changer
- Direct3D research
- ESP - 3D AABB
- Remove the static offsets
- Use CreateInterface from the SDK
- Get the local player using the interface instead of the offset

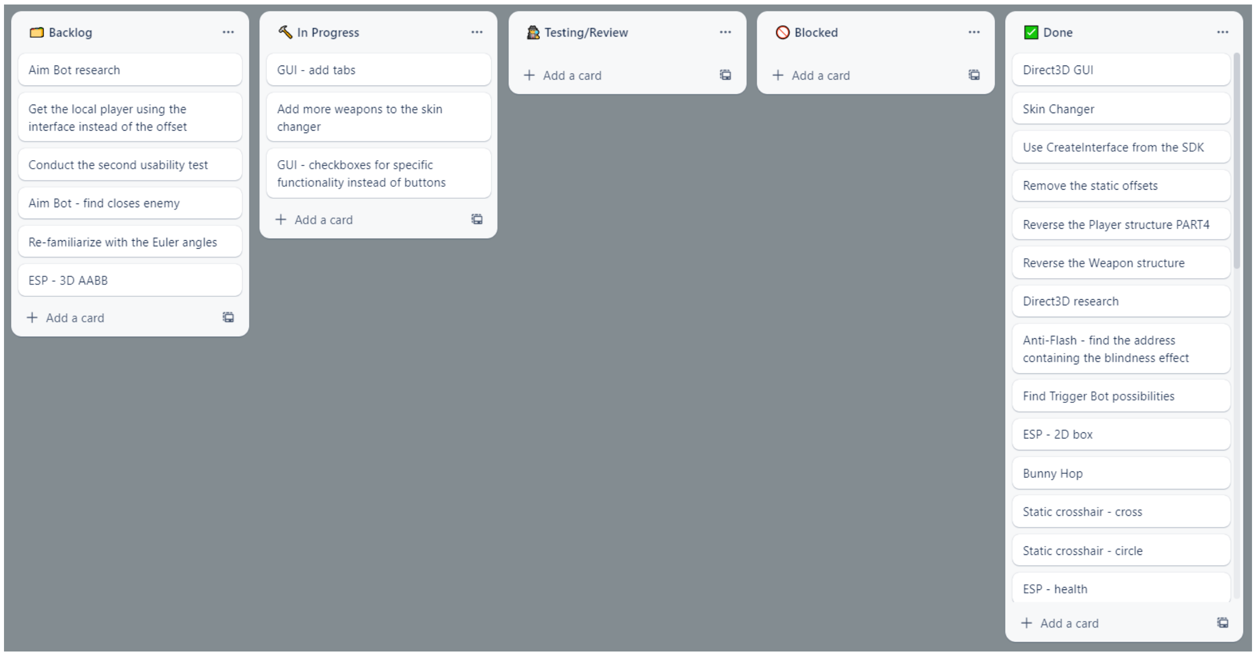
In Progress
- Reverse the Player structure PART4
- Reverse the Weapon structure

Testing/Review
- Direct3D GUI

Blocked
- Trigger Bot - find the ray cast function inside engine.dll

Done
- Anti-Flash - find the address containing the blindness effect
- Find Trigger Bot possibilities
- ESP - 2D box
- Bunny Hop
- Static crosshair - cross
- Static crosshair - circle
- ESP - health
- Conduct first usability phase
- ESP - snaplines
- In-game world space coordinates to screen coordinates converted.
- Pattern scanner - safe version
- Pattern scanner - brute force version
- Research on hooking techniques

2.1.8 Sprint 8



Backlog
- Aim Bot research
- Get the local player using the interface instead of the offset
- Conduct the second usability test
- Aim Bot - find closes enemy
- Re-familiarize with the Euler angles
- ESP - 3D AABB

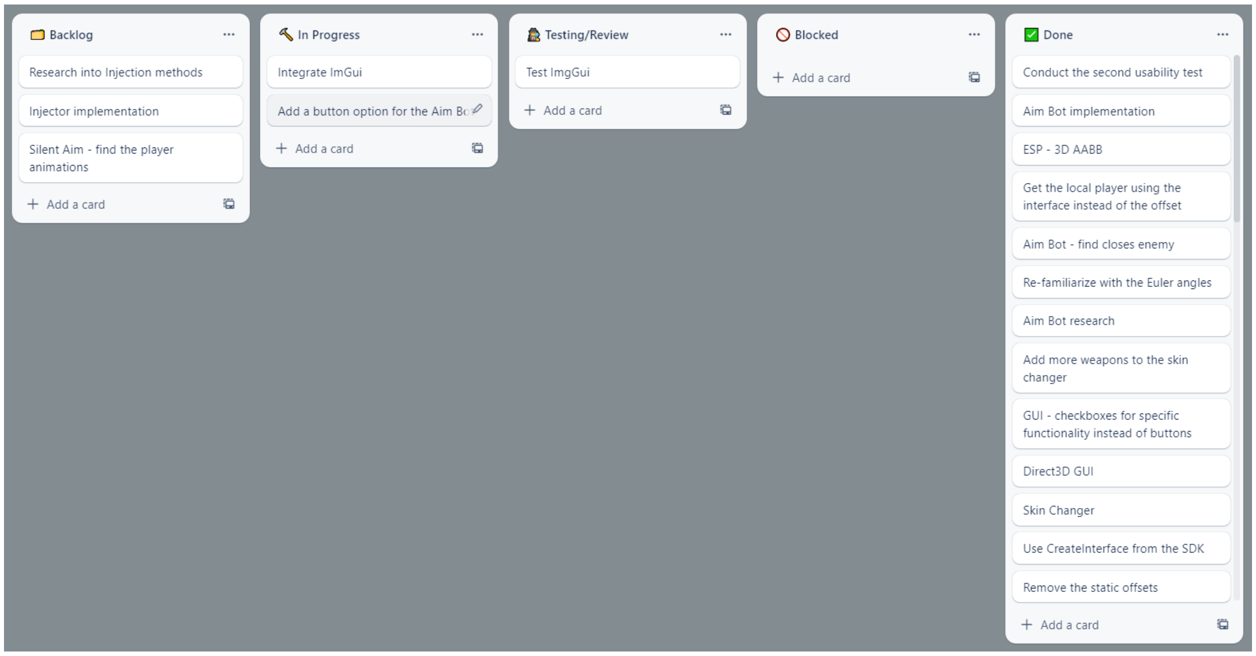
In Progress
- GUI - add tabs
- Add more weapons to the skin changer
- GUI - checkboxes for specific functionality instead of buttons

Testing/Review

Blocked

Done
- Direct3D GUI
- Skin Changer
- Use CreateInterface from the SDK
- Remove the static offsets
- Reverse the Player structure PART4
- Reverse the Weapon structure
- Direct3D research
- Anti-Flash - find the address containing the blindness effect
- Find Trigger Bot possibilities
- ESP - 2D box
- Bunny Hop
- Static crosshair - cross
- Static crosshair - circle
- ESP - health

2.1.9 Sprint 9



Backlog
- Research into Injection methods
- Injector implementation
- Silent Aim - find the player animations

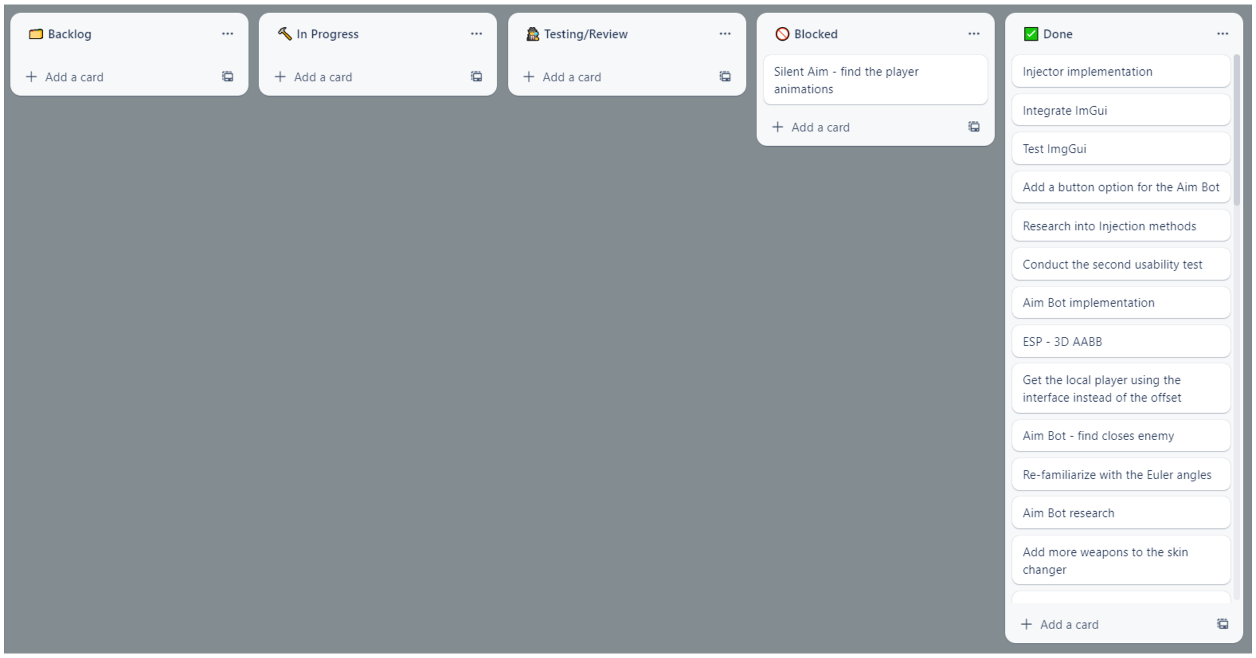
In Progress
- Integrate ImGui
- Add a button option for the Aim Bo

Testing/Review
- Test ImgGui

Blocked

Done
- Conduct the second usability test
- Aim Bot implementation
- ESP - 3D AABB
- Get the local player using the interface instead of the offset
- Aim Bot - find closes enemy
- Re-familiarize with the Euler angles
- Aim Bot research
- Add more weapons to the skin changer
- GUI - checkboxes for specific functionality instead of buttons
- Direct3D GUI
- Skin Changer
- Use CreateInterface from the SDK
- Remove the static offsets

2.1.10 Sprint 10



Backlog

In Progress

Testing/Review

Blocked
- Silent Aim - find the player animations

Done
- Injector implementation
- Integrate ImGui
- Test ImgGui
- Add a button option for the Aim Bot
- Research into Injection methods
- Conduct the second usability test
- Aim Bot implementation
- ESP - 3D AABB
- Get the local player using the interface instead of the offset
- Aim Bot - find closes enemy
- Re-familiarize with the Euler angles
- Aim Bot research
- Add more weapons to the skin changer