

Video Gaming for the Impaired

COMP8045 – Major Project

Justin Chau – A00767848
4-5-2018

Table of Contents

1.	Introduction	4
1.1.	Student Background	4
1.2.	Project Description	4
1.2.1.	Essential Problems.....	5
1.2.2.	Goals and Objectives	5
2.	Body	5
2.1.	Background.....	5
2.2.	Project Statement.....	6
2.3.	Possible Alternative Solutions	6
2.4.	Chosen Solution	6
2.5.	Details of Design and Development	6
2.5.1.	Deliverables	6
2.5.2.	System Diagrams / Architecture	7
2.5.3.	Offsets.....	8
2.5.4.	User Interface	13
2.5.4.1.	User Interface State Diagram	13
2.5.4.2.	User Interface Sample	14
2.5.4.3.	User Interface Code Snippet.....	14
2.5.5.	Aim Assist.....	15
2.5.5.1.	Aim Assist State Diagram.....	15
2.5.5.2.	Aim Assist Algorithm.....	16
2.5.5.3.	Aim Assist Code Snippet	16
2.5.6.	Trigger.....	18
2.5.6.1.	Trigger State Diagram	18
2.5.6.2.	Trigger Code Snippet	19
2.5.7.	Glow Extrasensory Perception (Glow ESP)	20
2.5.7.1.	Glow ESP State Diagram	20
2.5.8.	Glow ESP Code Snippet.....	21
2.5.9.	Bunny Hopping	22
2.5.9.1.	Bunny Hopping State Diagram	22
2.5.9.2.	Bunny Hopping Code Snippet.....	23
2.5.10.	Pattern Scanner	24

2.5.10.1.	Pattern Scanner State Diagram	26
2.5.10.2.	Pattern Scanner Code Snippet.....	27
2.5.11.	Memory Manager.....	28
2.5.11.1.	Memory Manager Main.....	28
2.5.11.1.1.	Main State Diagram	28
2.5.11.1.2.	Main Method Code Snippet	29
2.5.11.2.	Memory Manager Attach Process	30
2.5.11.2.1.	Attach Process State Diagram	30
2.5.11.2.2.	Attach Process Code Snippet.....	31
2.5.11.3.	Memory Manager Get Module.....	32
2.5.11.3.1.	Get Module State Diagram	32
2.5.11.3.2.	Get Module Code Snippet	33
2.5.12.	User Manual	33
2.6.	Testing Details and Results.....	35
2.6.1.	Acceptance Testing.....	35
2.6.1.1.	Verifying Auto Aim.....	35
2.6.1.2.	Verifying Field of View	35
2.6.1.2.1.	Verifying Field of View < 20	35
2.6.1.2.2.	Verifying Field of View < 90	37
2.6.1.2.3.	Verifying Field of View MAX	38
2.6.1.3.	Verifying Glow Extrasensory Perception	39
2.6.1.4.	Verifying Triggerbot.....	42
2.6.1.5.	Verifying Signature Scan	43
2.6.1.6.	Verifying Valve Anti Cheat Undetected	44
2.7.	Implications of Implementation	44
2.8.	Innovation.....	45
2.9.	Complexity	45
2.10.	Research in New Technologies	45
2.11.	Future Enhancements.....	46
2.12.	Timeline and Milestones	46
3.	Conclusion	47
3.1.	Lessons Learned.....	47
3.2.	Closing Remarks.....	47

4.	References	48
5.	Change Log	48
6.	Appendix.....	48
6.1.	Project Supervisor Approvals	48
6.2.	Approved Proposal	49

1. Introduction

1.1. Student Background

Education

British Columbia Institute of Technology (BCIT) – Burnaby, BC

- **Bachelor of Technology** | **Graduating 2018**
 - Network Security Administration Option
- **Diploma** | **Graduated 2016**
 - Computer Information Technology

Langara College – Vancouver, BC

- General Studies

Killarney Secondary High School – Vancouver, BC

- High School Diploma | **Graduated 2010**

Experience

Employment

Cowell Auto Group, IT Support – Richmond, BC

| **Apr 2017 – Present**

Description: Monitors and controls computers and peripheral data processing equipment. Enters commands using computer terminal and manages controls on computer and peripheral equipment. Monitors the system for failure or errors and responds by addressing/troubleshooting issues. Experience in location migration of a mid-size company into two new building and assisted in setup of infrastructure.

Projects

ICON Network, Practicum Student – Vancouver, BC

| **Sept 2014 – Dec 2014**

Description: Worked with Product Lead and teammates on researching and developing program to help manage their database. The program identified non-standardized data using the Google API to validate and convert data into a standardized form.

1.2. Project Description

This project is for the half practicum and I plan to work alone on this project.

The project's goal is to create a malicious software that would assist players on a specific game to make it simpler. Our software game of focus is Counter-Strike: Global Offensive (CS: GO), an online first-person shooting game (FPS). The software will assist users in areas such as hand-eye coordination and in-game knowledge. Since we are dealing with a first person shooting game, hand-eye coordination is very important on the success of how you play the video-game. Aiming for the head produces the most amount of damage and would be the quickest way of winning a duel against an opponent. Another important aspect of this hack is having advanced knowledge about opponents which this software will provide. In the end, this software will provide an advantage for players.

1.2.1. Essential Problems

The problem this project solves is one where a player, who may have had a life altering incident causing physical or visual impairments such as hand tremors or blindness, is having a difficult time playing or learning how to play a video game (Refer to Appendix 16.3 for examples). CS: GO is played online so there are no difficulty settings for the game. This project will have features that would assist the players in learning the game and allow them to adjust the difficulty of the game if it turns out to be too challenging. This game has been chosen due to the fact that it is an online game with software security measures therefore being able to play online with friends is important.

1.2.2. Goals and Objectives

The project's goal is to create a solution that will provide users with visual and physical aids when playing Counter-Strike: Global Offensive through reading and writing into the computer's memory. These solutions are:

- Visual
 - Extrasensory Perception
- Physical
 - Aim Assist
 - Trigger (Auto shooting when target is in crosshair)
 - Auto bunny hopping

2. Body

2.1. Background

Counter-Strike: Global Offensive is a very popular multiplayer first-person shooter game developed by Valve. It is the fourth game in the Counter-Strike series where the first game was released in 2000. Counter-Strike: Global Offensive is played having two teams, terrorist and counter-terrorist. The goal of the terrorist team is to plant the bomb and detonate the bomb on one of the two bomb sites provided on a map or eliminate every counter-terrorist before the round ends. It is the counter-terrorist goal to defend both bomb sites from being planted and detonated by either eliminating all terrorist before a planted bomb or defusing a planted bomb within the bomb timer to detonate. The game is played in a 5 on 5 setting where you can play with your friends. Therefore, in order to excel at a first person shooting video-game it requires good hand-eye coordination some users may not have due to previous accidents or impairments.

This software is considered a hack so it will be protected by their own anti-cheating system and for us to allow a user with disabilities to play with their friends we need to bypass their system. Video games developed by Valve is protected by their anti-cheat system called Valve Anti-Cheat (VAC). VAC sends client application challenges, if expected responses are not received, the user will be flagged for a violation. It uses Signature Scanning to detect possible cheats when scanning computer's memory and processes. When an anomaly is detected, it is compared to their database of banned applications or analyzed by Valve engineers. If the code is confirmed as a new cheat, it will be added to their database.

My interest in this project came from my previous experiences with this video-game. I have thought this video game was a great game to play with friends and about the people who could not play this game due to their own physical impairments.

2.2. Project Statement

Ignore your visual and physically limitations and allow yourself to play one of the most popular first person shooting games with your friends! Using this software, you can slowly practice with your friends to become better and more importantly have fun.

2.3. Possible Alternative Solutions

There are three different types of solutions that can be created for this situation:

1. **Internal:** a dynamic-link library (DLL) must be injected directly into the game and from there you can hook the video game functions. This method allows you to directly read and write the games memory through raw pointers.
2. **Internal/External hybrid:** a DLL that must be injected into any 32-bit application and creates a handle to the game process and read/write.
3. **External:** an execution(EXE) file that directly creates a handle to read/write the game process. The main disadvantage is that they are not in sync with the game's thread which may sometimes lead to unexpected results.
4. **Kernel Mode:** A driver which can perform read and write memory of a user mode program using IOCTL.

2.4. Chosen Solution

With our time constraint, an **external** software will be created. The software will consist of a few components which are **memory management and signature scanning**. These three functions will allow us to grab and manipulate memory information and help build the user features.

2.5. Details of Design and Development

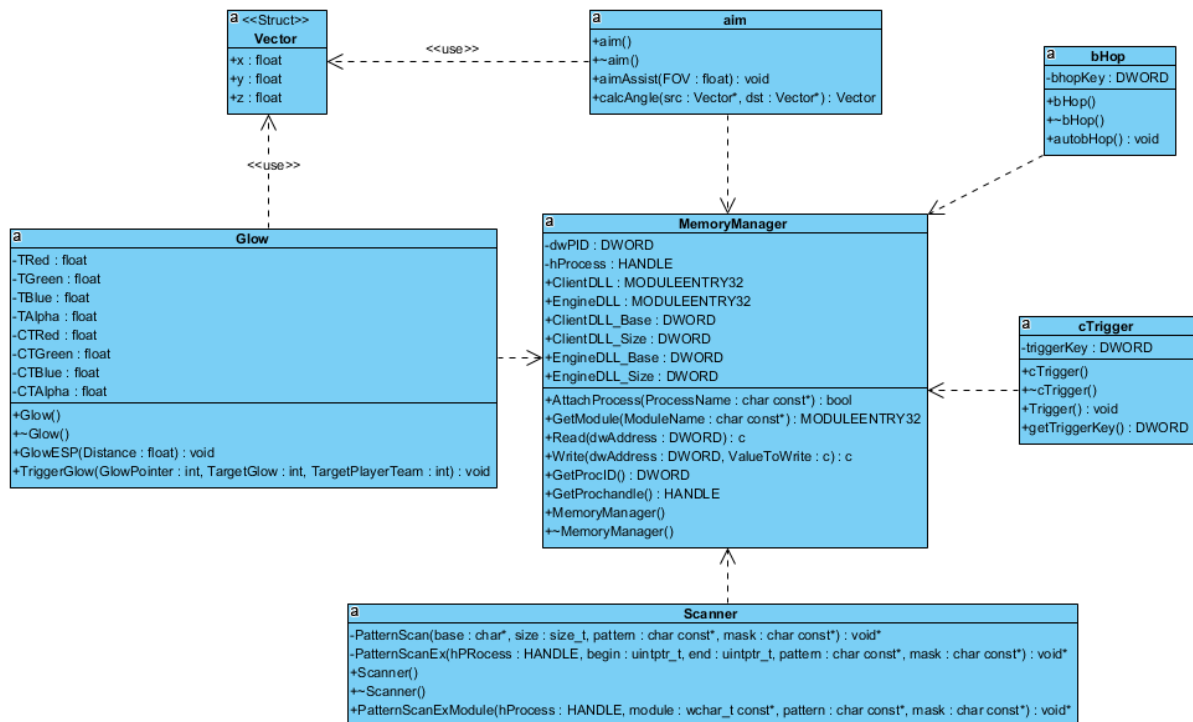
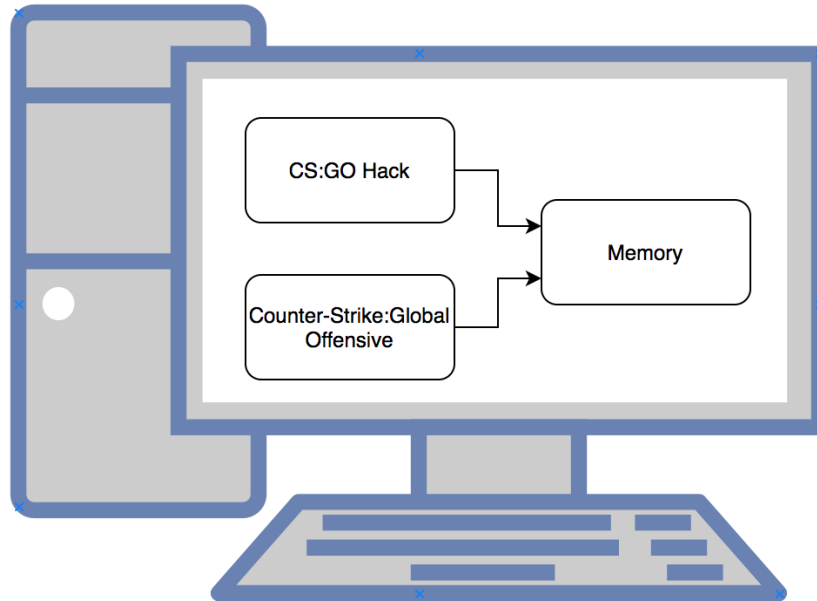
2.5.1. Deliverables

The deliverables for this project are as follows:

1. **Counter-Strike: Global Offensive Hack** - an executable that will be use with CS: GO
2. **Final Report** - final report for the practicum course.
3. **User Guide** - How to use the executable with CS: GO

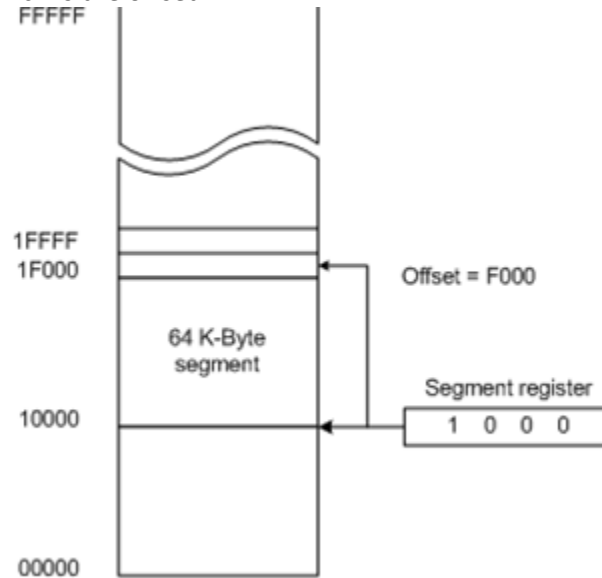
2.5.2. System Diagrams / Architecture

The following diagram is a software architecture diagram that visually represents the interaction of programs and features



2.5.3. Offsets

What is an offset? An offset is a value that indicates the distance from a beginning of an object given the base address. Looking at the diagram below, we can set the base address at 1000 and the information we want is from 1F000. The difference in distance between each other in hexadecimal is F000 which is the offset.



In this project, when CS: GO is launched it will be assigned a base address and an offset is the distance where certain information is stored relative to that base address.

What information do we want? The information we mainly want is our local player and other players. From gathering the local player, other information is stored within an offset of the local players base offset in the game's module.

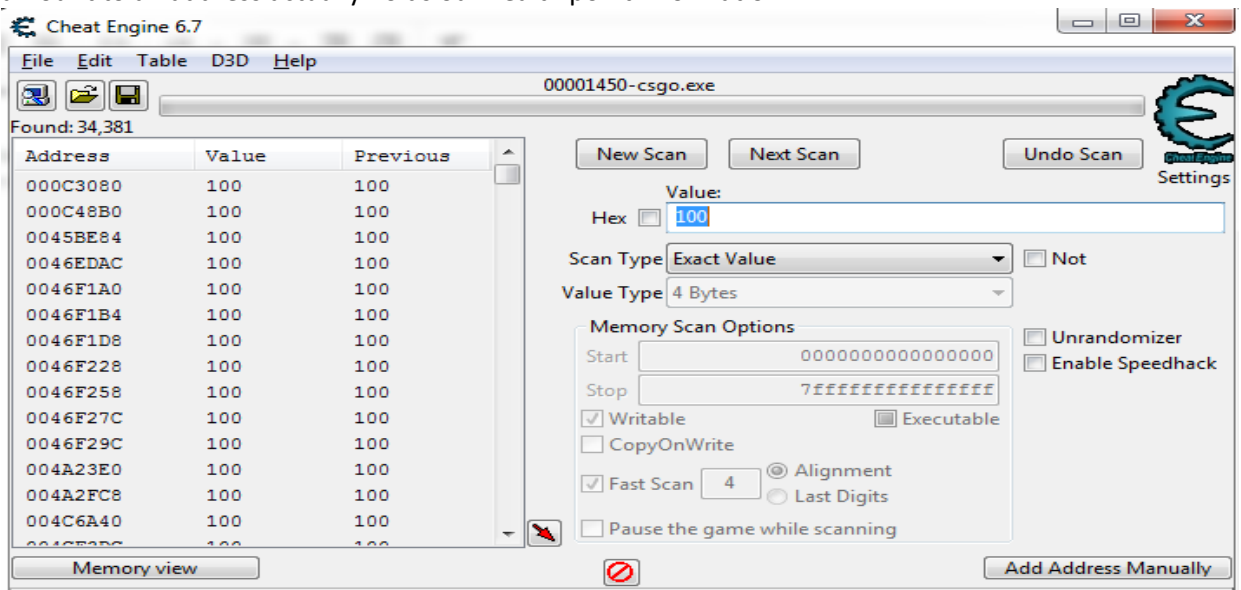
How do we find an offset of our local player? By using cheat engine and a series of steps, we can find the offset of our local player

To start, we can use our health point (HP) value to find our local player. When our local player spawns, he has a health point of 100. So, we will scan for 100 in cheat engine after attaching to CS: GO.

The health point is shown on the bottom left corner



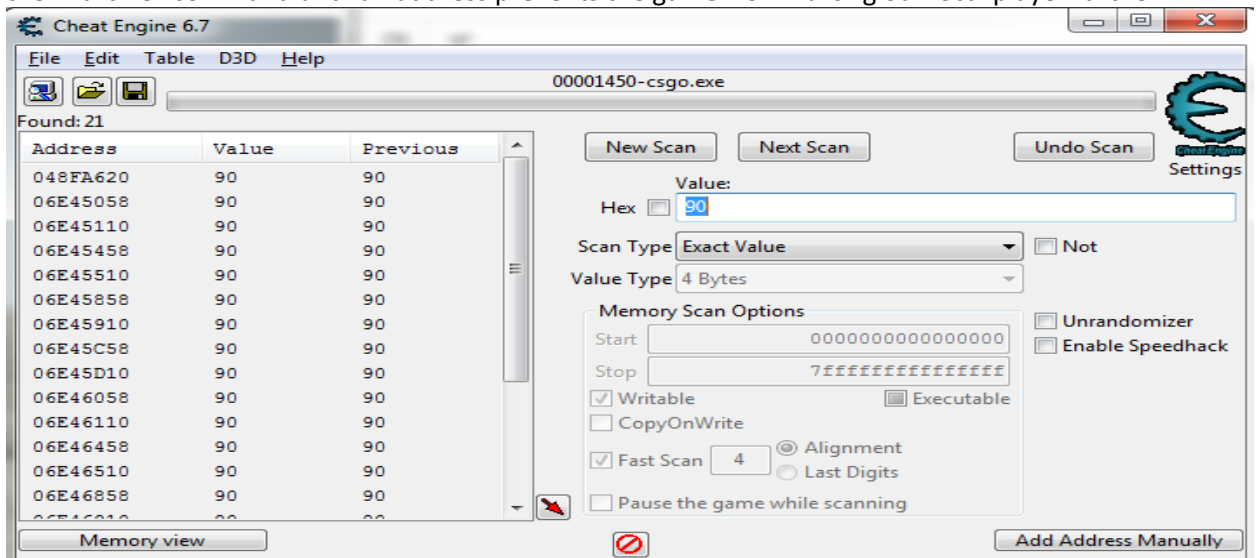
Scanning for the health point value 100, we find about 34,000 addresses that match our health point value. Going through 34,000 addresses will be time consuming so we want to lower that amount to an address actually holds our health point information.



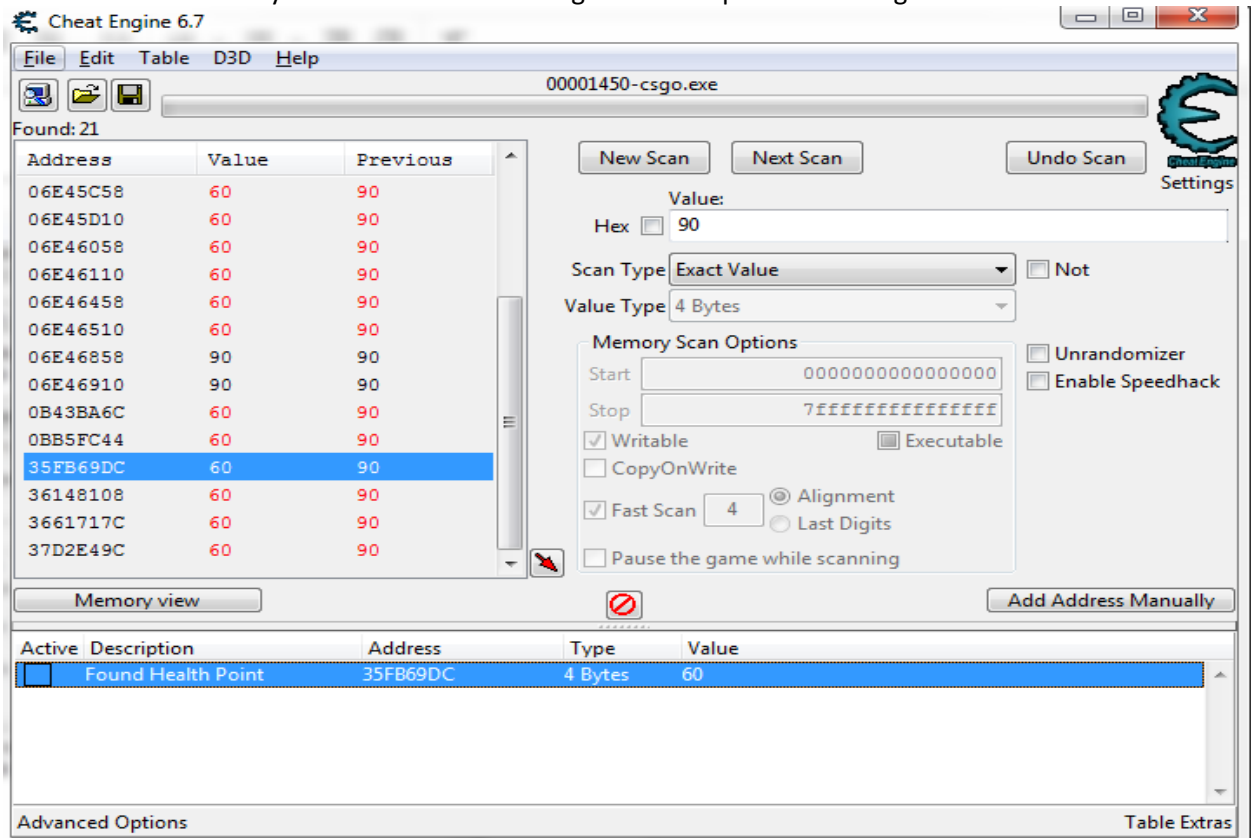
In order to lower the results, we have to change our value. We can change our HP value using a built in-game command “hurtme” to lower the health point to 90



Now, we will scan the next value at 90. Our scan matched 21 addresses which is more manageable than the previous scan. From here, we would freeze each address value and trigger the “hurtme” command until an address prevents the game from hurting our local player further.



The address in memory we found that is freezing our health point value in-game is “35FB69DC”

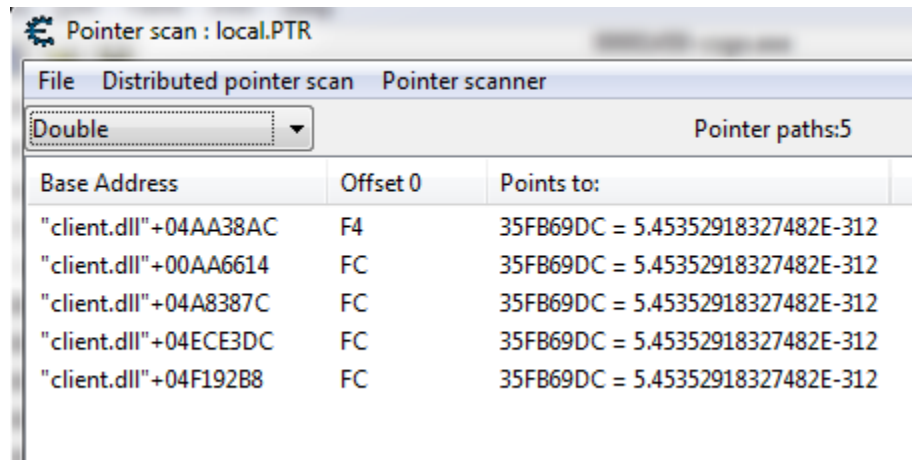


We can find our health point offset in reference to our local player by looking at what accesses this address. By looking at the set of opcodes, it shows that there is a pointer address plus **FC** that stores our health point. **FC** is our health offset from our local player base.

The following opcodes accessed 35FB69DC

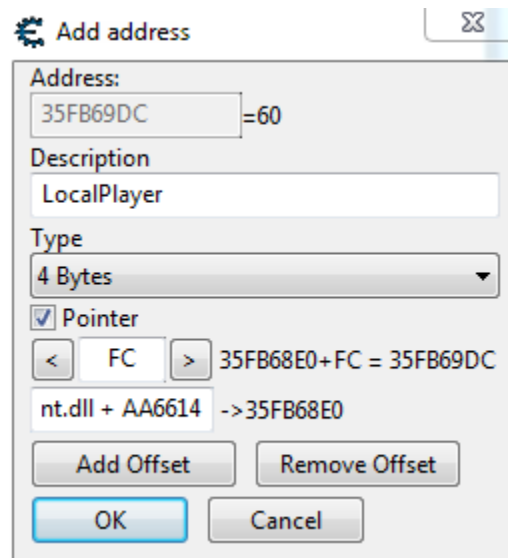
Count	Instruction
11247	19C07F90 - 8B 81 FC000000 - mov eax,[ecx+000000FC]
920	1A22EBD9 - 89 17 - mov [edi],edx
920	1A22EBBD - F3 0F7F 4F 10 - movdqu [edi+10],xmm1
1377	19DC2300 - 83 B9 FC000000 00 - cmp dword ptr [ecx+000000FC],00
1377	19DC234A - 83 B9 FC000000 00 - cmp dword ptr [ecx+000000FC],00
1377	19DC24A5 - 83 B9 FC000000 00 - cmp dword ptr [ecx+000000FC],00
1377	19DC39BF - 83 B9 FC000000 00 - cmp dword ptr [ecx+000000FC],00
1607	1A22EBD7 - 8B 16 - mov edx,[esi]
1837	1A22EBB4 - F3 0F6F 4E 10 - movdqu xmm1,[esi+10]

By using the instruction address “**19DC2300**” to find our local player base, we do a pointer scan. Our results show that there are 5 paths that points to that location. Given these 5 paths, we can use Cheat Engine to manually add these values to test the right offset.



Base Address	Offset 0	Points to:
"client.dll"+04AA38AC	F4	35FB69DC = 5.45352918327482E-312
"client.dll"+00AA6614	FC	35FB69DC = 5.45352918327482E-312
"client.dll"+04A8387C	FC	35FB69DC = 5.45352918327482E-312
"client.dll"+04ECE3DC	FC	35FB69DC = 5.45352918327482E-312
"client.dll"+04F192B8	FC	35FB69DC = 5.45352918327482E-312

Testing the first offset **AA6614**, we get a value of our local player but we’re not certain that this is the value. So, we add **client.dll + AA6614** as our pointer and the health offset relative to our local player, FC. We should get the value of our local player’s in-game health point (60 HP in our case) which confirms that we have the right local player and health point offset.



Address: 35FB69DC =60

Description: LocalPlayer

Type: 4 Bytes

☒ Pointer

< FC > 35FB68E0+FC = 35FB69DC

nt.dll + AA6614 -> 35FB68E0

Add Offset Remove Offset

OK Cancel

In the end, we find that our local player offset is **0xAA6614** and our health point offset is **0xFC**.

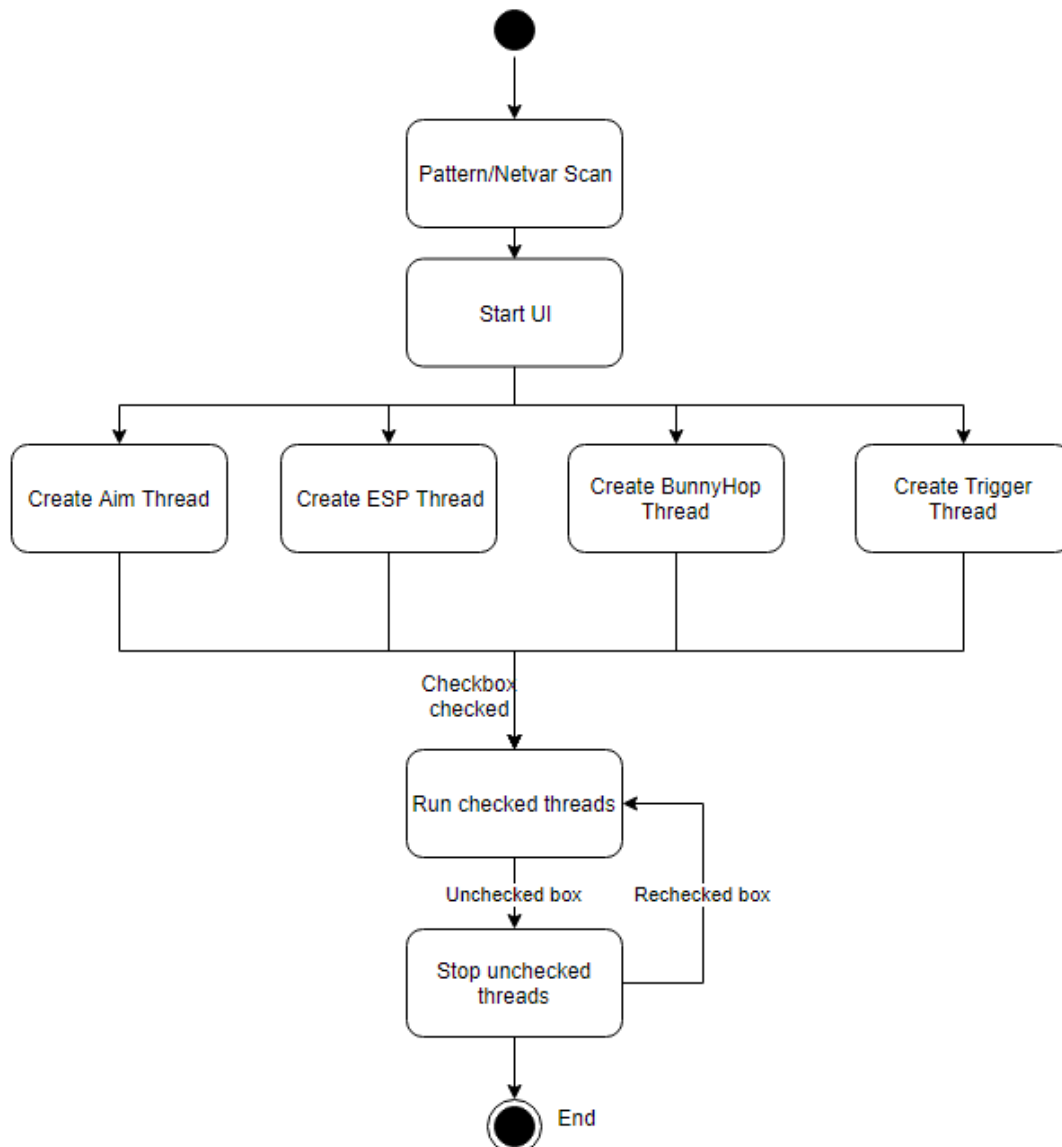
Generally, this is how to get most offset values just by scanning a common value using Cheat Engine and can be done with other players in the video game.

2.5.4. User Interface

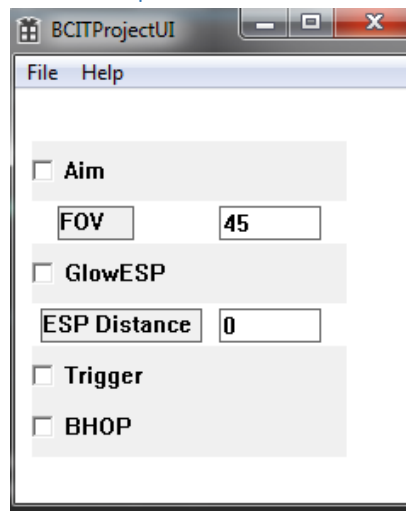
Our user interface is really basic since it is not the main focus of this practicum. This is the runner method that executes the other features.

The user interface starts by running the pattern scan to grab the offsets for the other functions to work properly. This is the main method and initiates the pattern scan then creates threads for all the functions to work independently without blocking. Each thread is constantly checking if any checkboxes are checked off. When a feature is activated, a flag is set to trigger the function to run until the program exits or the function is unchecked.

2.5.4.1. User Interface State Diagram



2.5.4.2. User Interface Sample



2.5.4.3. User Interface Code Snippet

```
CreateWindowEx(NULL, TEXT("button"), TEXT("Aim"),
    WS_VISIBLE | WS_CHILD | BS_AUTOCHECKBOX,
    10, 30, 185, 35,
    hWind, (HMENU)IDC_AIMBOX, ((LPCREATESTRUCT)lParam)->hInstance, NULL);

CreateWindowEx(NULL, TEXT("button"), TEXT("GlowESP"),
    WS_VISIBLE | WS_CHILD | BS_AUTOCHECKBOX,
    10, 90, 185, 35,
    hWind, (HMENU)IDC_ESPBOX, ((LPCREATESTRUCT)lParam)->hInstance, NULL);

CreateWindowEx(NULL, TEXT("button"), TEXT("Trigger"),
    WS_VISIBLE | WS_CHILD | BS_AUTOCHECKBOX,
    10, 150, 185, 35,
    hWind, (HMENU)IDC_TRIGGERBOX, ((LPCREATESTRUCT)lParam)->hInstance, NULL);

CreateWindowEx(NULL, TEXT("button"), TEXT("BHOP"),
    WS_VISIBLE | WS_CHILD | BS_AUTOCHECKBOX,
    10, 180, 185, 35,
    hWind, (HMENU)IDC_HOPBOX, ((LPCREATESTRUCT)lParam)->hInstance, NULL);

CreateWindow(TEXT("static"), TEXT("FOV"), WS_CHILD | WS_VISIBLE | WS_BORDER, 25, 68, 45, 20, hWind, NULL, NULL, NULL);
CreateWindow(TEXT("static"), TEXT("ESP Distance"), WS_CHILD | WS_VISIBLE | WS_BORDER, 15, 128, 95, 20, hWind, NULL, NULL, NULL);
FOVtext = CreateWindow(TEXT("Edit"), TEXT("45"), WS_CHILD | WS_VISIBLE | WS_BORDER, 120, 68, 60, 20, hWind, NULL, NULL, NULL);
ESPtext = CreateWindow(TEXT("Edit"), TEXT("0"), WS_CHILD | WS_VISIBLE | WS_BORDER, 120, 128, 60, 20, hWind, NULL, NULL, NULL);

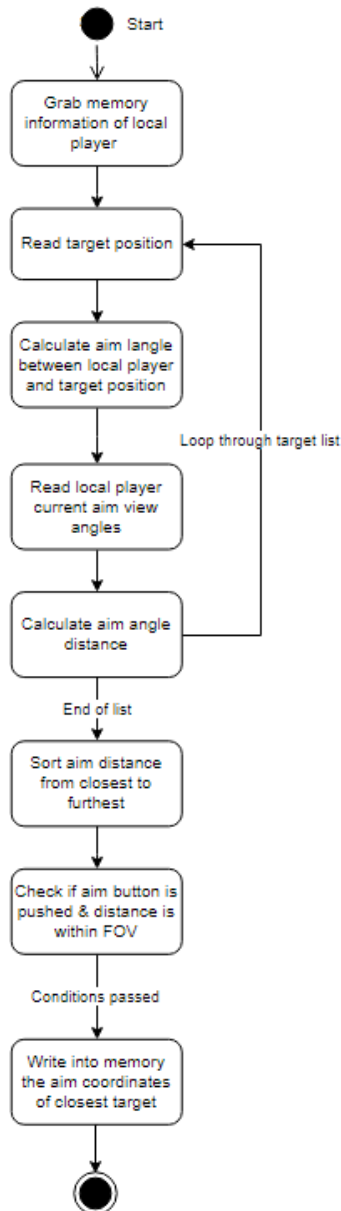
break;
```

```
HANDLE aimThread = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)aimThreadTask, 0, CREATE_SUSPENDED, NULL);
HANDLE glowThread = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)ESPThreadTask, 0, CREATE_SUSPENDED, NULL);
HANDLE triggerThread = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)TriggerThreadTask, 0, CREATE_SUSPENDED, NULL);
HANDLE hopThread = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)HopThreadTask, 0, CREATE_SUSPENDED, NULL);
```

2.5.5. Aim Assist

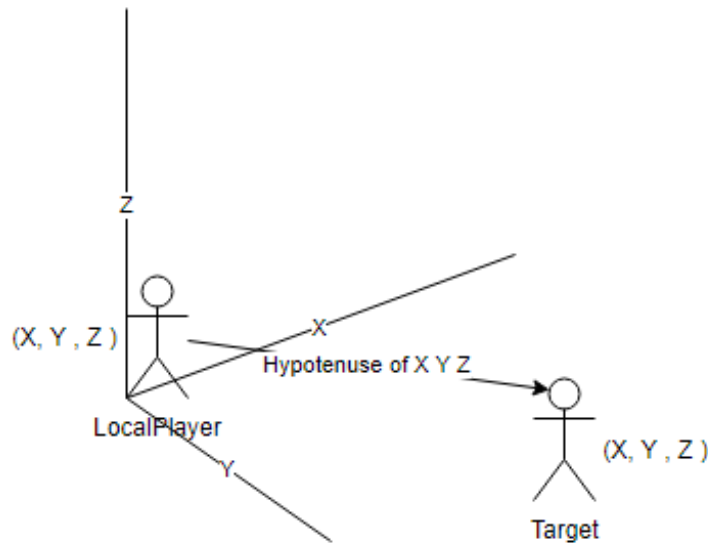
This feature will assist the user and help them aim at a target that is within the field of view value. There are two calculations in this feature which are aim angle and aim distance. Aim angle calculation uses an algorithm to figure out what the X, Y coordinates needed to aim at a target. Aim distance is for sorting purposes to set priority of targets to aim at from closest to furthest. It uses the X, Y coordinates of the local player's aim view and the location of the targets head relative to the local's aim view to calculate how far the distance between the crosshair and targets head is.

2.5.5.1. Aim Assist State Diagram



2.5.5.2. Aim Assist Algorithm

Given that we know the X, Y, Z Coordinates of the local player and the target. We subtract their coordinates to get the difference and use Pythagorean theorem to get the hypotenuse of the value. From the hypotenuse, we need can gather the coordinates of the end tip where our local player needs to aim at on an X, Y grid.



2.5.5.3. Aim Assist Code Snippet

Calculate aim angle coordinates for the local player to aim at.

Parameters:

src - X, Y, Z coordinates of our local player

dst - X, Y, Z coordinates of a target player

return - X, Y coordinates of the target player relative to local players view angle.

```
Vector aim::calcAngle(Vector* src, Vector* dst)
{
    Vector distance;
    Vector newAngle;

    //subtract coordinates from local player(src) and target player(dst)
    distance.x = dst->x - src->x;
    distance.y = dst->y - src->y;
    distance.z = dst->z - src->z;

    //get the hypotenuse of the x, y , z
    float hyp = sqrt((distance.x * distance.x) + (distance.y * distance.y) + (distance.z * distance.z));

    //calculate the new X, Y angle of the target relative to the current local player
    newAngle.y = atan2(distance.y, distance.x) * 180 / 3.14;
    newAngle.x = -atan2(distance.z, hyp) * 180 / 3.14;
    newAngle.z = 0;

    return newAngle;
}
```

Using the calculated X, Y coordinates of the enemy's head, it can be passed through the aim distance function. The aim distance calculates the distance from the crosshair to the head of the target. Once the aim distance is calculated, it can be compared to the field of view value to see if the target is close enough to assist in aiming.

Parameters:

Local - X, Y coordinates of our local player's view angle

Enemy - X, Y coordinates of a target player relative to our local player's view angle

return – the distance between the two X, Y coordinates

```
float aimDistance(Vector * Local, Vector * Enemy)
{
    float Distance;
    float DistanceX;
    float DistanceY;

    //subtract the X,Y to get the distance
    DistanceX = (Enemy->x - Local->x) * (Enemy->x - Local->x);
    DistanceY = (Enemy->y - Local->y) * (Enemy->y - Local->y);

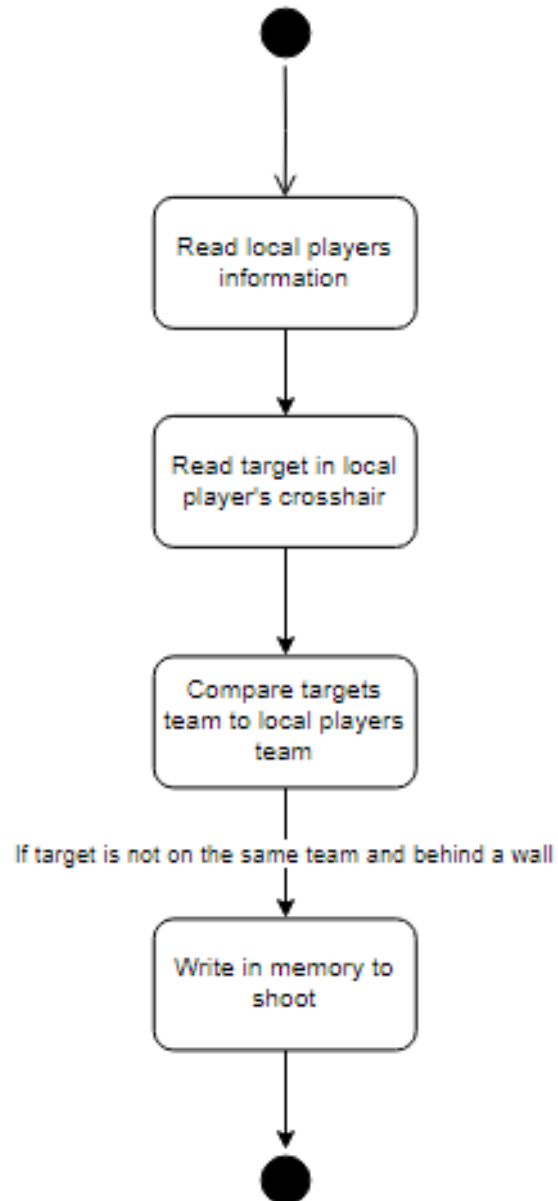
    //hypotenuse to get the hypotenesus distance
    //this is the value we use to compare with FOV
    Distance = sqrt(DistanceX + DistanceY);

    return Distance;
}
```

2.5.6. Trigger

Trigger will automatically shoot when an enemy is within crosshairs. It will grab information about a player inside the user's crosshair. Using conditions, it will check whether the player inside the crosshair is on the enemy team. If the player is on the enemy team, it will trigger in-game shoot function.

2.5.6.1. Trigger State Diagram



2.5.6.2. Trigger Code Snippet

```
void cTrigger::Trigger()
{
    //Retrieve player information
    DWORD LocalPlayer = Mem->Read<DWORD>(Mem->ClientDLL_Base + playerBase); //LOCALPLAYER
    int LocalPlayerCrosshair = Mem->Read<int>(LocalPlayer + iCrosshairID); //CROSSHAIRID
    int LocalPlayerTeam = Mem->Read<int>(LocalPlayer + teamOffset); //TEAMNUM

    //Retrieve information about the target in the local players crosshairs
    DWORD TargetPlayer = Mem->Read<DWORD>(Mem->ClientDLL_Base + entityBase + ((LocalPlayerCrosshair - 1) * EntLoopDist));
    int TargetTeam = Mem->Read<int>(TargetPlayer + teamOffset); //targets team

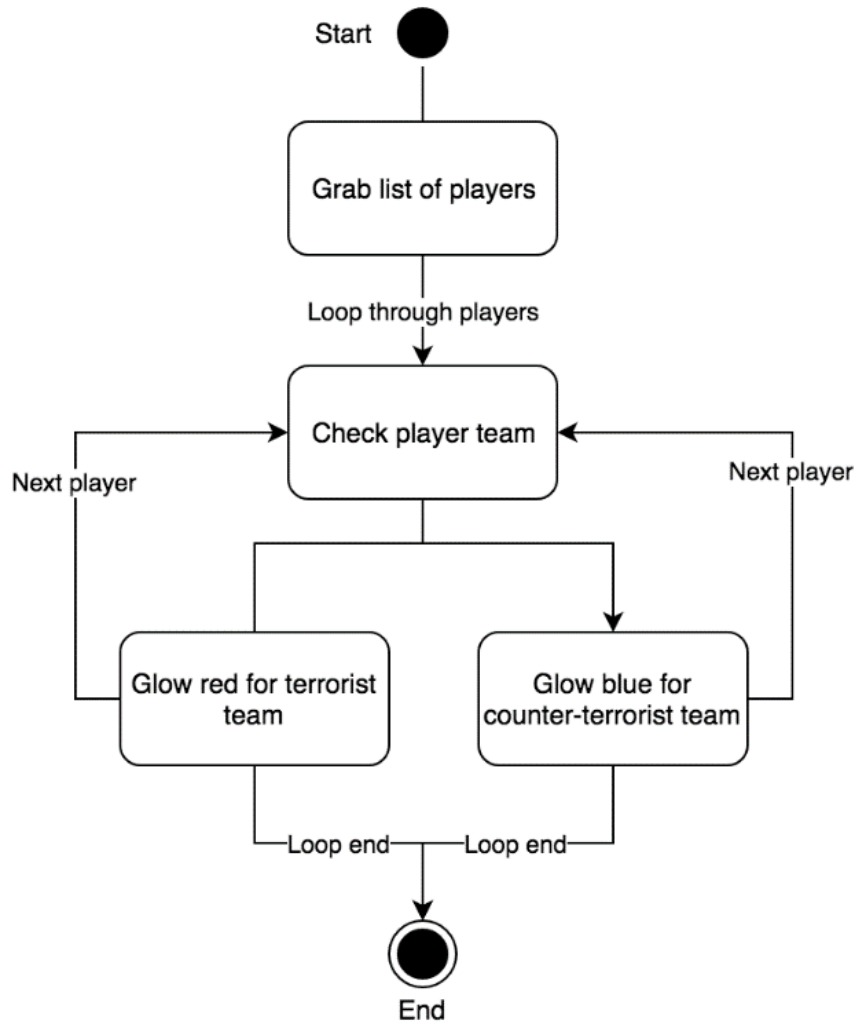
    //Read if the target is behind an object
    bool TargetDormant = Mem->Read<bool>(TargetPlayer + Dormant); //HARDCODED DORMANT

    //Checking if target is behind a wall or on the other team
    if ((LocalPlayerCrosshair > 0 && LocalPlayerCrosshair <= 64) && (TargetPlayer != NULL) && (TargetTeam != LocalPlayerTeam))
    {
        Sleep(10); //Delay before shooting
        Mem->Write<int>(Mem->ClientDLL_Base + ForceAttack, 5); //Shoot
        Sleep(10); //Delay between shots
        Mem->Write<int>(Mem->ClientDLL_Base + ForceAttack, 4); //Stop shooting
        Sleep(10); //Delay after shooting
    }
}
```

2.5.7. Glow Extrasensory Perception (Glow ESP)

Glow ESP helps with information about other players in the game. It will glow a player creating an outline of the players shape if they are within the distance value. Using the previous algorithm in calculating the aim angle, the hypotenuse of that algorithm is the distance between the local player and any other player in the game. With that value, we can compare it with a value that a user sets for ESP distance to decide whether to glow certain players.

2.5.7.1. Glow ESP State Diagram



2.5.8. Glow ESP Code Snippet

Parameters:

distance – the distance between our local player and other targets within the game

return – none

```
void Glow::GlowESP(float distance)
{
    //Grab local player
    DWORD dwLocalPlayer = Mem->Read<DWORD>(Mem->ClientDLL_Base + playerBase);

    //Glow Pointer for ESP
    int glowPointer = Mem->Read<DWORD>(Mem->ClientDLL_Base + GlowObject);

    //For adjustable ESP
    int Distance;

    Vector localCoords, targetCoords;

    //Loop through every player in the game
    for (int i = 0; i <= 32; i++)
    {
        //Read info of target player
        int TargetPlayer = Mem->Read<int>(Mem->ClientDLL_Base + entityBase + i * EntLoopDist);
        int TargetPlayerGlow = Mem->Read<int>(TargetPlayer + GlowIndex); //Targets glow index value
        int TargetPlayerTeam = Mem->Read<int>(TargetPlayer + teamOffset); //Targets team

        ReadProcessMemory(Mem->GetProchandle(), (PBYTE*)(TargetPlayer + m_vecOrigin), &targetCoords, sizeof(float[3]), 0);
        ReadProcessMemory(Mem->GetProchandle(), (PBYTE*)(dwLocalPlayer + m_vecOrigin), &localCoords, sizeof(float[3]), 0);

        //Check distance between the target and local
        Distance = getDistances(&localCoords, &targetCoords);

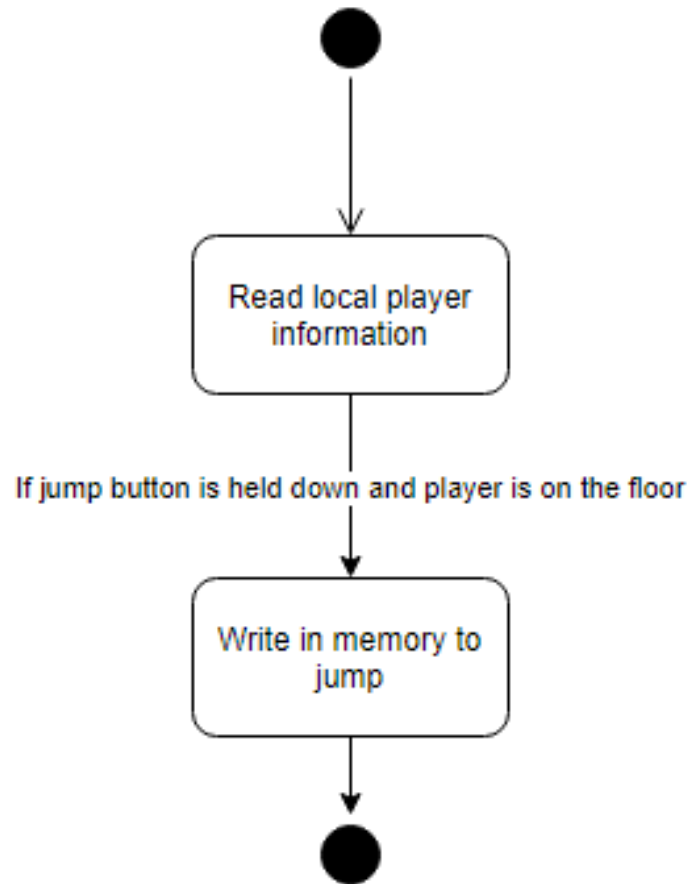
        //Default 0 to fully activate Glow
        if (distance == 0)
        {
            //Activate function to write into memory
            TriggerGlow(glowPointer, TargetPlayerGlow, TargetPlayerTeam);
        }

        //If distance is set
        else if (Distance < distance)
        {
            //Activate function to write into memory
            TriggerGlow(glowPointer, TargetPlayerGlow, TargetPlayerTeam);
        }
    }
}
```

2.5.9. Bunny Hopping

In terms of design, bunny hopping is similar to the trigger function but gathers different information. Instead of enemy, it gathers information about the local user and if they are touching the ground. As long as the user is touching the ground and holding onto the jump button, this feature will write into memory the in-game jump function making the user continuously jump.

2.5.9.1. Bunny Hopping State Diagram



2.5.9.2. Bunny Hopping Code Snippet

```
void bHop::autobHop()
{
    //Grab local player
    DWORD dwLocalPlayer = Mem->Read<DWORD>(Mem->ClientDLL_Base + playerBase);

    //Grab flags to see if player is on ground
    BYTE mFlag = Mem->Read<BYTE>(dwLocalPlayer + Flags);

    //If spacebar (jump button is pressed)
    if (GetAsyncKeyState(VK_SPACE) & 0x8000)
    {
        //If player is on the ground
        if (mFlag & (1 << 0))
        {
            //Jump
            Mem->Write(Mem->ClientDLL_Base + ForceJump, 6);
        }
    }
}
```


2.5.10. Pattern Scanner

What is a Pattern Scanner? A pattern scanner grabs offsets dynamically by scanning the game's modules. CS: GO developers are always combating people who decide to create these types of software to use on their game therefore they frequently update their offsets. Pattern scanning eliminates the need for manually updating offsets.

How do we find a signature?

We will use the same steps to find an offset by using the values of the health point. Since it is known that our HP offset is 0xFC, we can make a pattern and mask to match our signature. As mentioned earlier, when we can find the health point offset by looking at what accesses that specific address.

The following opcodes accessed 368EFC0C

Count	Instruction
12381	1B817F90 - 8B 81 FC000000 - mov eax,[ecx+000000FC]
948	1BE3EBD9 - 89 17 - mov [edi],edx
948	1BE3EBBD - F3 0F7F 4F 10 - movdqu [edi+10],xmm1
1512	1B9D2300 - 83 B9 FC000000 00 - cmp dword ptr [ecx+000000FC],00
1512	1B9D234A - 83 B9 FC000000 00 - cmp dword ptr [ecx+000000FC],00
1512	1B9D24A5 - 83 B9 FC000000 00 - cmp dword ptr [ecx+000000FC],00
1512	1B9D39BF - 83 B9 FC000000 00 - cmp dword ptr [ecx+000000FC],00
1749	1BE3EBD7 - 8B 16 - mov edx,[esi]
1986	1BE3EBB4 - F3 0F6F 4E 10 - movdqu xmm1,[esi+10]

We take the first line at '1B9D2300' and search that area of the code in a disassembler to look at the surrounding bytes.

Memory Viewer

client.dll+392300

Address	Bytes	Opcode
client.dll+392300	83 B9 FC000000 00	cmp dword ptr [ecx+000000FC],00
client.dll+392307	7F 2D	jg client.dll+392336
client.dll+392309	8B 01	
client.dll+39230B	8B 80 74040000	
client.dll+392311	FF D0	
client.dll+392313	83 F8 06	
client.dll+392316	74 1E	
client.dll+392318	8B 47 08	
client.dll+39231B	C7 40 2C 00000000	mov [eax+2C],00000000
client.dll+392322	8B 47 08	mov eax,[edi+08]
client.dll+392325	C7 40 30 00000000	mov [eax+30],00000000
client.dll+39232C	8B 47 08	mov eax,[edi+08]
client.dll+39232F	C7 40 34 00000000	mov [eax+34],00000000

Goto Address

Fill in the address you want to go to

1B9D2300

OK Cancel

compare two operands

As shown in the screen capture, we can see that the surrounding pattern of FC000000 is 83 B9 ? ? ? ? 00 7F 2D 8B 01 allowing us to make the mask of xx????xxxx where '?' is a wild card for the offset we want. The mask is a pattern where we let the program know if that's the area where the value can change. When Valve decides to update the offsets, the FC may change to an unknown value therefore the wildcard will allow us to read any changed in that area. Usually, games keep the surrounding pattern the same while changing the offset of where they store information.

To testing this pattern, we compare our hardcoded value to the value that is dynamically scanned.

```
std::cout << "BCIT Project" << std::endl;
cout << "===== " << endl;

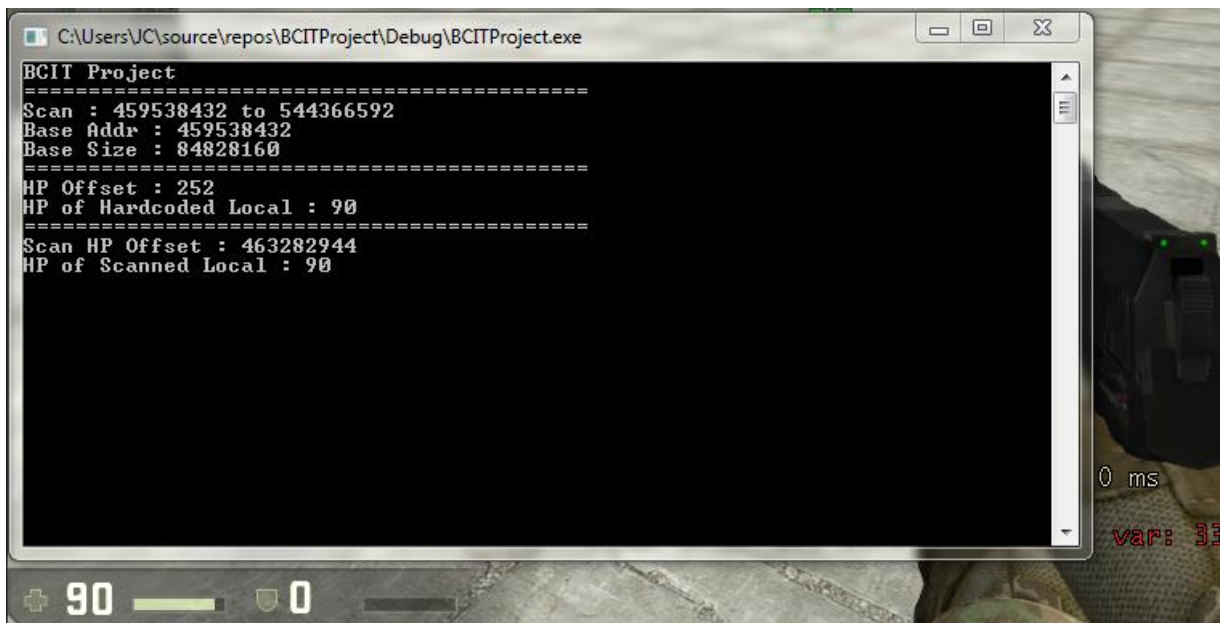
DWORD scanHPoff = (DWORD)nScanner->PatternScanEXModule(Mem->GetProchandle(), L"client.dll", "\x83\xB9\x00\x00\x00\x00\x7F\x2D\x8B\x01", "xx????xxxx");
DWORD realLocal = Mem->Read<DWORD>(Mem->ClientDLL_Base + playerBase);
int HP = Mem->Read<int>(realLocal + healthOffset);

cout << "Base Addr : " << Mem->ClientDLL_Base << endl;
cout << "Base Size : " << Mem->ClientDLL.modBaseSize << endl;
cout << "===== " << endl;
cout << "HP Offset : " << healthOffset << endl;
cout << "Reading Memory from Real Local : " << realLocal << endl;
cout << "HP of Real Local : " << HP << endl;

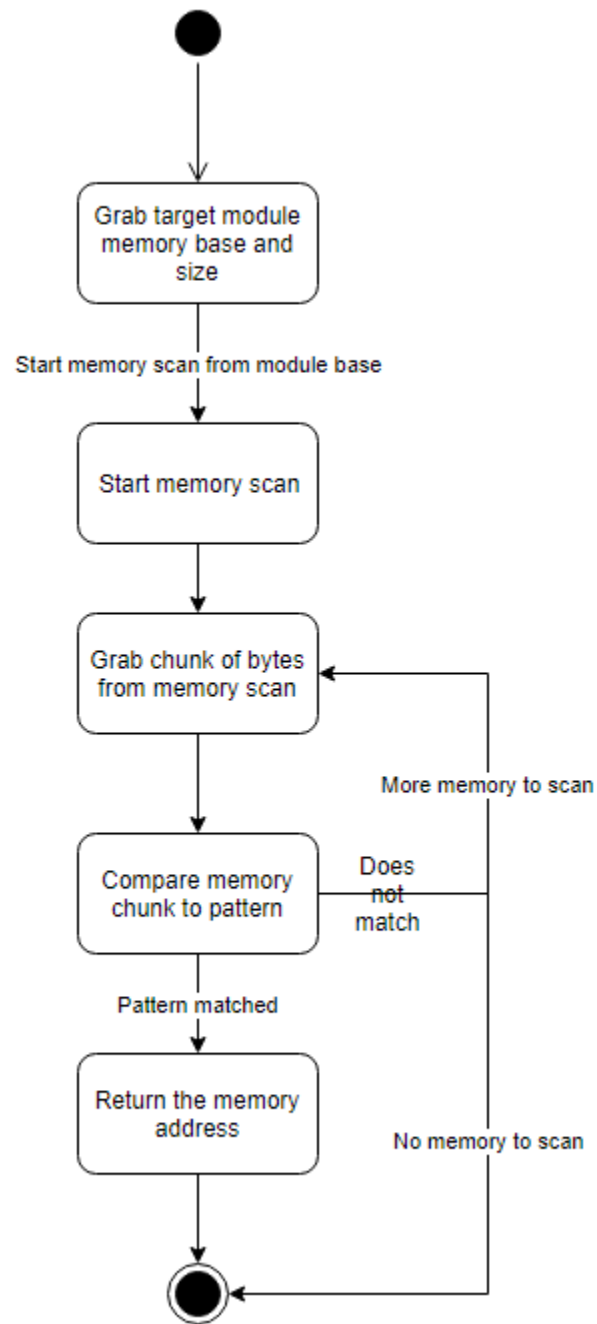
int scanHP, scanGlow;
ReadProcessMemory(Mem->GetProchandle(), (PBYTE*)(scanHPoff + 0x2), &scanHP, sizeof(int), NULL);

cout << "===== " << endl;
cout << "Scan HP Offset : " << scanHPoff << endl;
cout << "HP of Scan Local : " << Mem->Read<int>(realLocal + scanHP) << endl;
```

Our results seem correct when we compare our hardcoded local health point to scanned local health point.



2.5.10.1. Pattern Scanner State Diagram



2.5.10.2. Pattern Scanner Code Snippet

Parameters:

base – base address of a module

size – address size of a module

pattern – the byte pattern to scan between the base address and base address plus size

mask – the pattern to signal which value can be a wild card

return – the wildcard values

```
void * Scanner::PatternScan(char* base, size_t size, const char* pattern, const char* mask)
{
    unsigned int patternLength = strlen(mask);

    for (unsigned int i = 0; i < size - patternLength; i++)
    {
        bool found = true;
        for (unsigned int j = 0; j < patternLength; j++)
        {
            if (mask[j] != '?' && pattern[j] != *(base + i + j))
            {
                found = false;
                break;
            }
        }
        if (found)
        {
            return (void*)(base + i);
        }
    }
    return nullptr;
}
```

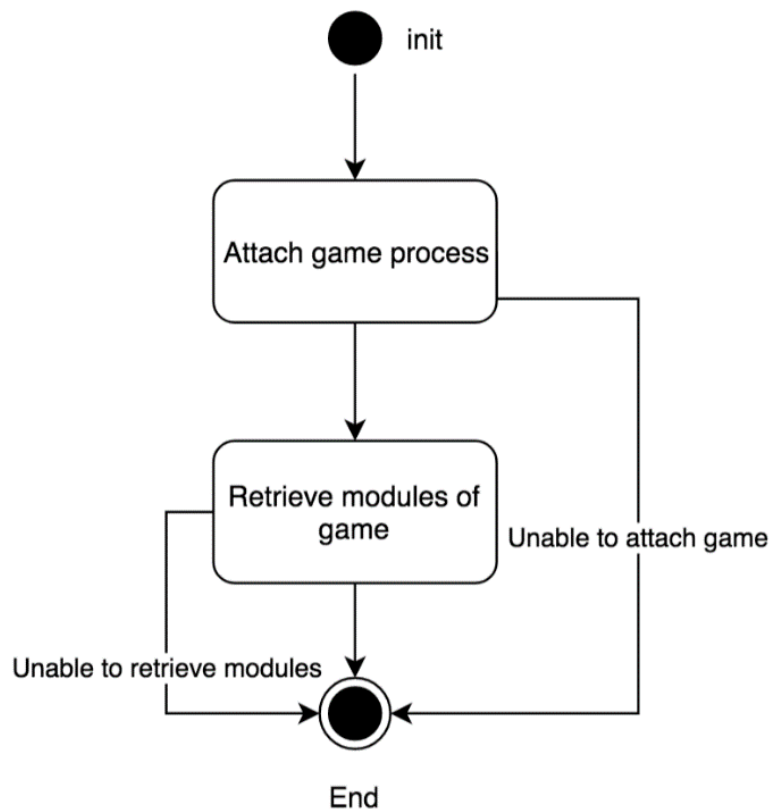
2.5.11. Memory Manager

Memory Manager attaches the software to the correct process and modules and provides the software with the ability to read and write into memory. This is the brain of the whole program without this function, the software will not work.

2.5.11.1. Memory Manager Main

The initiating sequence for this function is to attach itself to the game and grab the correct modules related to the game.

2.5.11.1.1. Main State Diagram



2.5.11.1.2. Main Method Code Snippet

```
MemoryManager()
{
    this->hProcess = NULL;
    this->dwPID = NULL;

    try {
        if (!AttachProcess("csgo.exe")) throw 1;
        this->ClientDLL = GetModule("client.dll");
        this->EngineDLL = GetModule("engine.dll");

        this->ClientDLL_Base = (DWORD)this->ClientDLL.modBaseAddr;
        this->EngineDLL_Base = (DWORD)this->EngineDLL.modBaseAddr;

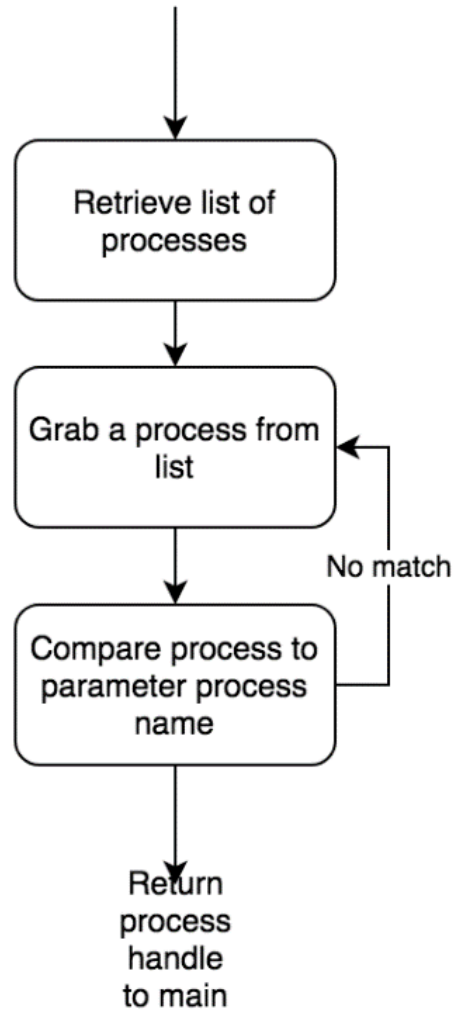
        if (this->ClientDLL_Base == 0x0) throw 2;
        if (this->EngineDLL_Base == 0x0) throw 3;

        this->ClientDLL_Size = this->ClientDLL.dwSize;
        this->EngineDLL_Size = this->EngineDLL.dwSize;
        this->ServerDLL_Size = this->ServerDLL.dwSize;
    }
    catch (int numErr) {
        switch (numErr)
        {
            case 1: MessageBoxA(NULL, "CS: GO is not running", "ERROR", MB_ICONSTOP | MB_OK); exit(0); break;
            case 2: MessageBoxA(NULL, "Client.dll not found", "ERROR", MB_ICONSTOP | MB_OK); exit(0); break;
            case 3: MessageBoxA(NULL, "Engine.dll is not running", "ERROR", MB_ICONSTOP | MB_OK); exit(0); break;
        }
    }
    catch (...) {
        MessageBoxA(NULL, "Unknown exeception thrown", "ERROR", MB_ICONSTOP | MB_OK);
        exit(0);
    }
}
```

2.5.11.2. Memory Manager Attach Process

Attach game function in memory management grabs the list every process running on the computer and compares the name to the game's name process. After finding the correct process, it will return it to the main function where it is stored for later reading and manipulation.

2.5.11.2.1. Attach Process State Diagram



2.5.11.2.2. Attach Process Code Snippet

Parameters:

ProcessName – the name of the process you want to attach

return – true on success, false on failure

```
bool AttachProcess(const char* ProcessName)
{
    //creates a snapshot of the specified processes including modules
    HANDLE hPID = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, NULL);
    //check if the snapshot is valid
    if (hPID == INVALID_HANDLE_VALUE) return false;

    //create helper struct that will contain info about process
    PROCESSENTRY32 procEntry;
    //set the dwsize of procEntry
    procEntry.dwSize = sizeof(procEntry);

    //grab the processname and compare it to the list of processes after
    const WCHAR *procNameChar;
    int nChars = MultiByteToWideChar(CP_ACP, 0, ProcessName, -1, NULL, 0);
    procNameChar = new WCHAR[nChars];
    MultiByteToWideChar(CP_ACP, 0, ProcessName, -1, (LPWSTR)procNameChar, nChars);

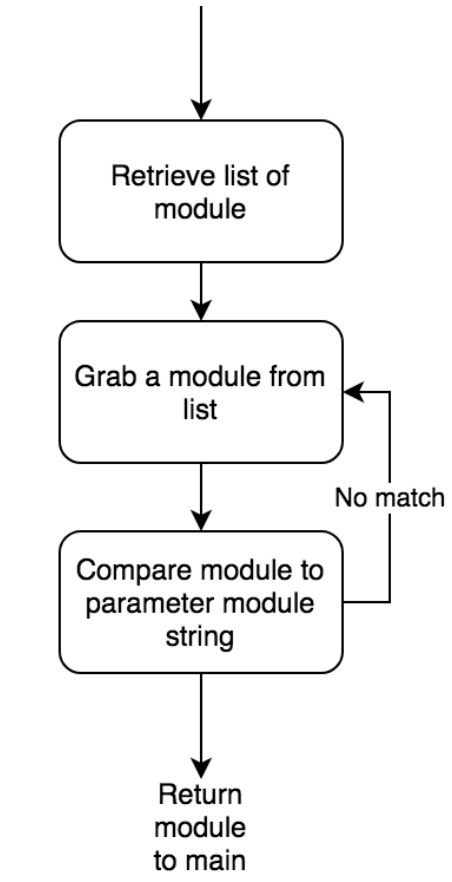
    do
    {
        //compare the two string
        if (!wcscmp(procEntry.szExeFile, procNameChar))
        {
            this->dwPID = procEntry.th32ProcessID;
            CloseHandle(hPID);
            this->hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, this->dwPID);
            return true;
        }
    } while (Process32Next(hPID, &procEntry));

    CloseHandle(hPID);
    return false;
}
```


2.5.11.3. Memory Manager Get Module

There are two modules to grab from CS: GO which are client.dll and engine.dll. This function works similar to attaching a process. It grabs all the modules and compares the list to the given module names. When it finds the correct module, it returns the modules size and base address.

2.5.11.3.1. Get Module State Diagram



2.5.11.3.2. Get Module Code Snippet

Parameters:

ModuleName – the name of the module you want to retrieve

return – the module that matches module name given

```
MODULEENTRY32 GetModule(const char * ModuleName)
{
    //creates a snapshot of the specified processes including modules
    HANDLE hModule = CreateToolhelp32Snapshot(TH32CS_SNAPMODULE, dwPID);

    MODULEENTRY32 mEntry;
    mEntry.dwSize = sizeof(mEntry);

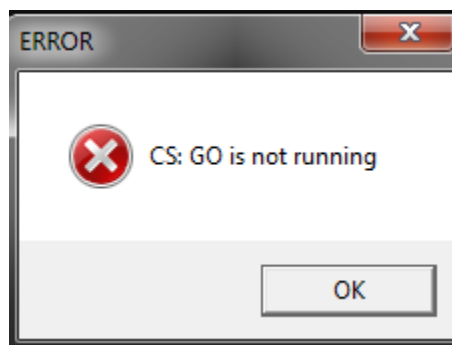
    const WCHAR *modNameChar;
    int nChars = MultiByteToWideChar(CP_ACP, 0, ModuleName, -1, NULL, 0);
    modNameChar = new WCHAR[nChars];
    MultiByteToWideChar(CP_ACP, 0, ModuleName, -1, (LPWSTR)modNameChar, nChars);

    do
    {
        if (!wcscmp(mEntry.szModule, modNameChar))
        {
            CloseHandle(hModule);
            return mEntry;
        }
    } while (Module32Next(hModule, &mEntry));

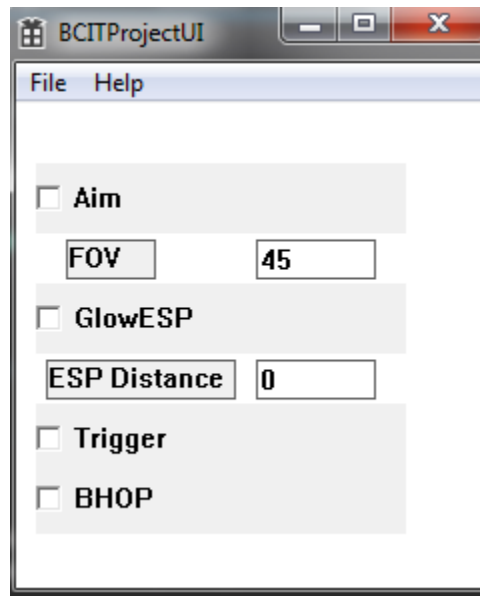
    CloseHandle(hModule);
    mEntry.modBaseAddr = 0x0;
    return mEntry;
}
```

2.5.12. User Manual

1. To use this software, you will need a copy of Counter-Strike: Global Offensive purchased off steam.
2. Counter-Strike: Global Offensive should be running before you run this software or an error may occur as shown below.



3. When started correctly, the user interface should look like the image shown below.



4. There are 4 features in software including aim assist, extra sensory perception, trigger shooting, and bunny hopping. Check the box beside to use the following feature:

a. **Aim Assist:** When activated, the right click on the mouse will assist the user in aiming.

FOV: The field of view is the minimum distance needed between the local players crosshair to the target players head before activating aim assist. Default is 45 with the value ranges from 0 - 360.

b. **Glow ESP:** Will show a glowing outline of all players within the ESP Distance.

ESP Distance: The distance between the local player and all players before activating the Glow ESP. Default is 0 for full distance with value ranges of 0 - 999.

c. **Trigger:** Automatically shoots when an enemy target is in the crosshairs

d. **BHOP:** Continuously jump when spacebar is held down.

2.6. Testing Details and Results

2.6.1. Acceptance Testing

2.6.1.1. Verifying Auto Aim

Test: Move around a bot standing still without using the mouse to aim

Pass: Crosshair locks onto a target player without using mouse and target distance from the crosshair is 0.

Fail: The crosshair comes off the target when moving

Result: Pass



2.6.1.2. Verifying Field of View

Test: Three distances: long (180), medium (90), short (20) ranges.

Pass: Crosshair locks onto the bot within that general range

Fail: The crosshair constantly locks on or does not lock within the distance given.

Debugging by displaying the distance from crosshair to the targets head.

2.6.1.2.1. Verifying Field of View < 20

Set Condition Field of View: 20

```
void *aimThreadTask(aim* aAim)
{
    while (true)
    {
        aAim->aimAssist(20);
    }

    return 0;
}
```

Distance < 20

Expected: The crosshair to lock onto target

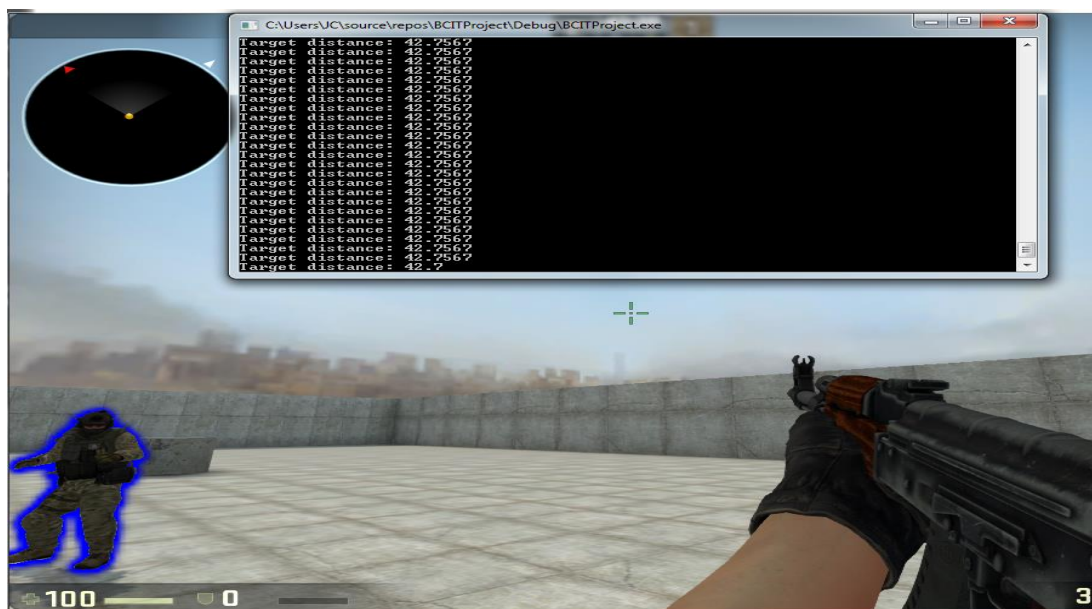
Result: **PASS**



Distance > 20

Expected Result: The crosshair does not lock

Result: **PASS**



2.6.1.2.2. Verifying Field of View < 90

Set Condition Field of View: 90

```
void *aimThreadTask(aim* aAim)
{
    while (true)
    {
        aAim->aimAssist(90);
    }

    return 0;
}
```

Distance < 90

Expected Result: The crosshair aims at the targets head

Result: PASS



Distance > 90

Expected Result: The crosshair does not aim at target

Result: PASS



2.6.1.2.3. Verifying Field of View MAX

Set Condition Field of View: 180 (MAX)

Expected Result: The crosshair locks to the target no matter the distance

Result: PASS



2.6.1.3. Verifying Glow Extrasensory Perception

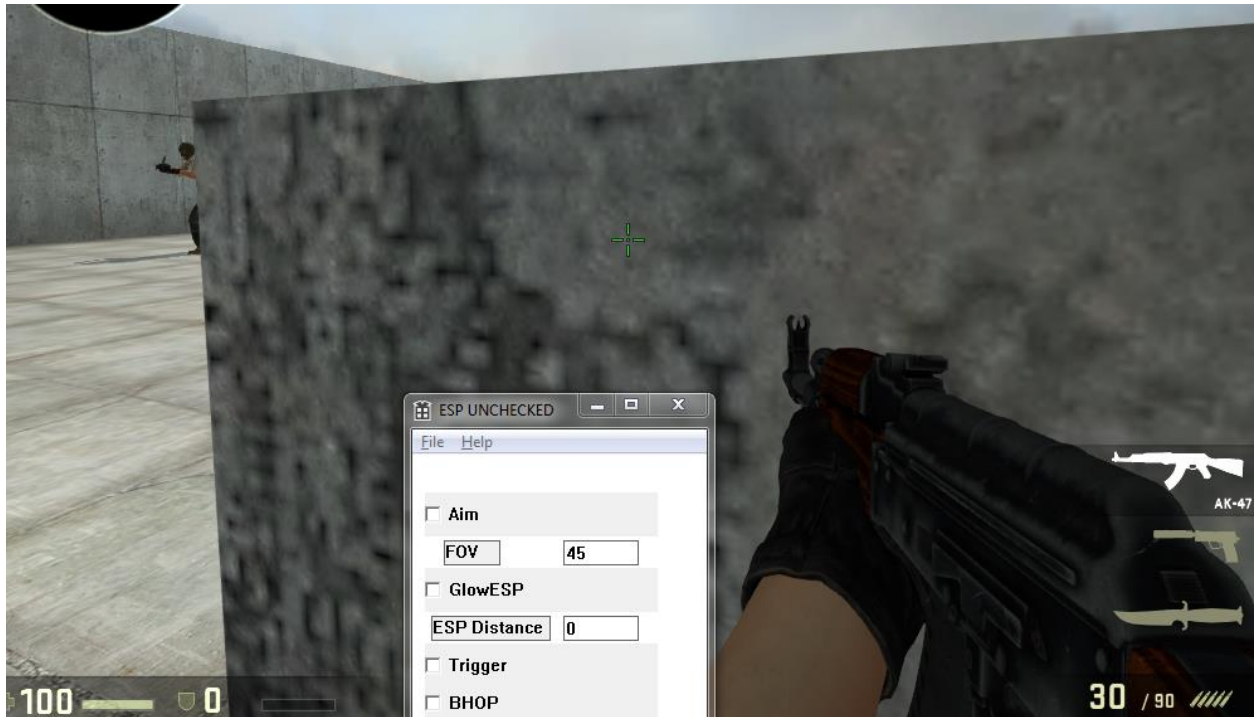
Test: Place a player on the other side of a wall to see if information can be retrieved

Pass: Information is given about the bot including equipment, name, health point

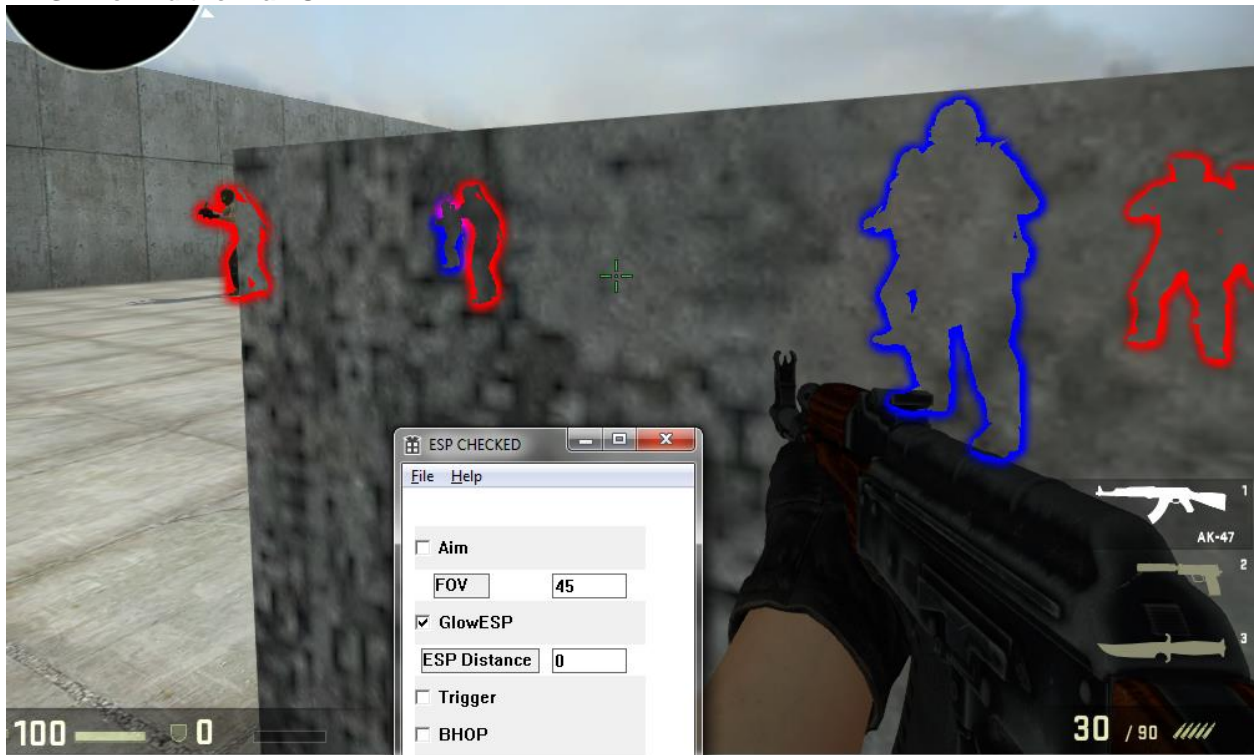
Fail: There is no information given

Results: PASS

ESP Behind the Wall OFF



ESP Behind the Wall ON



ESP OFF

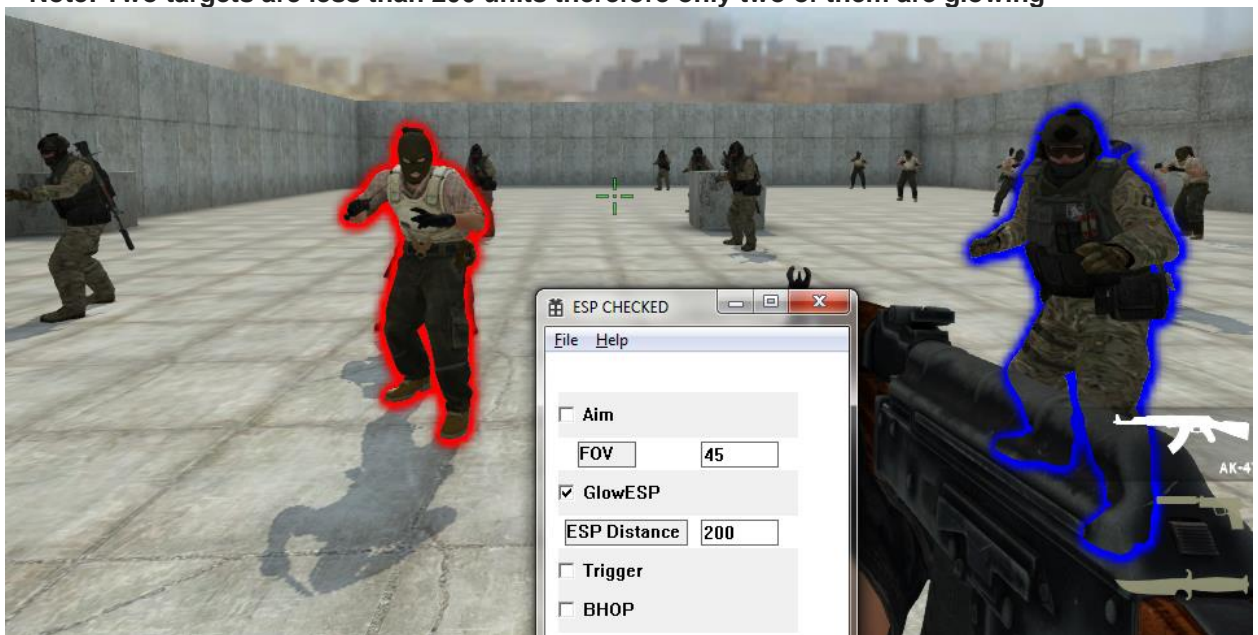


Full ESP



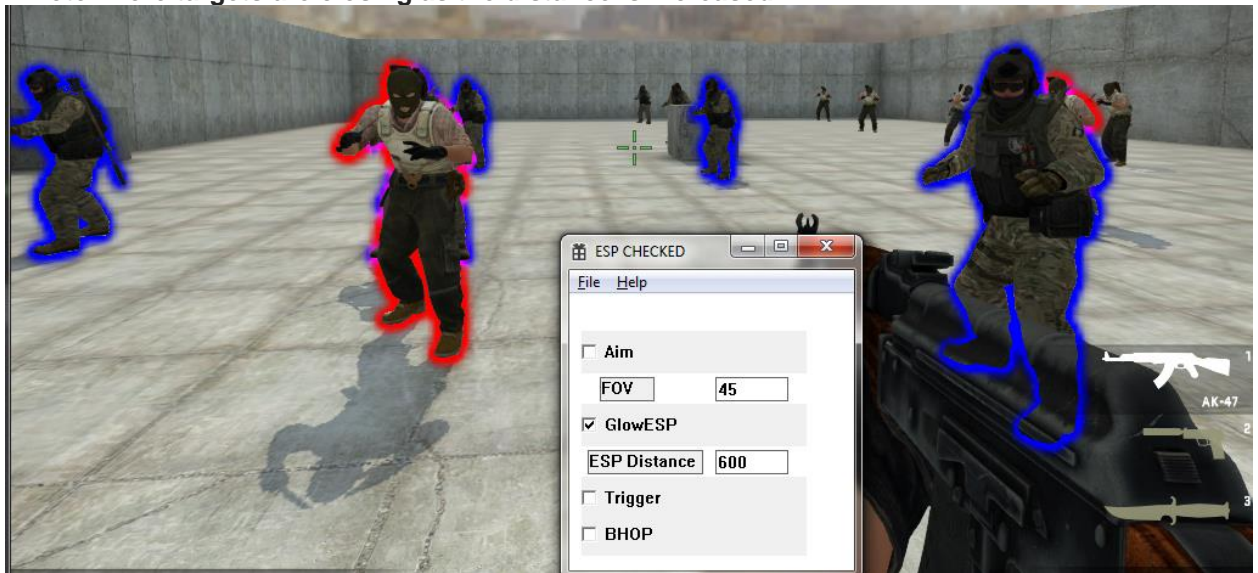
ESP Low Distance: 200 units

Note: Two targets are less than 200 units therefore only two of them are glowing

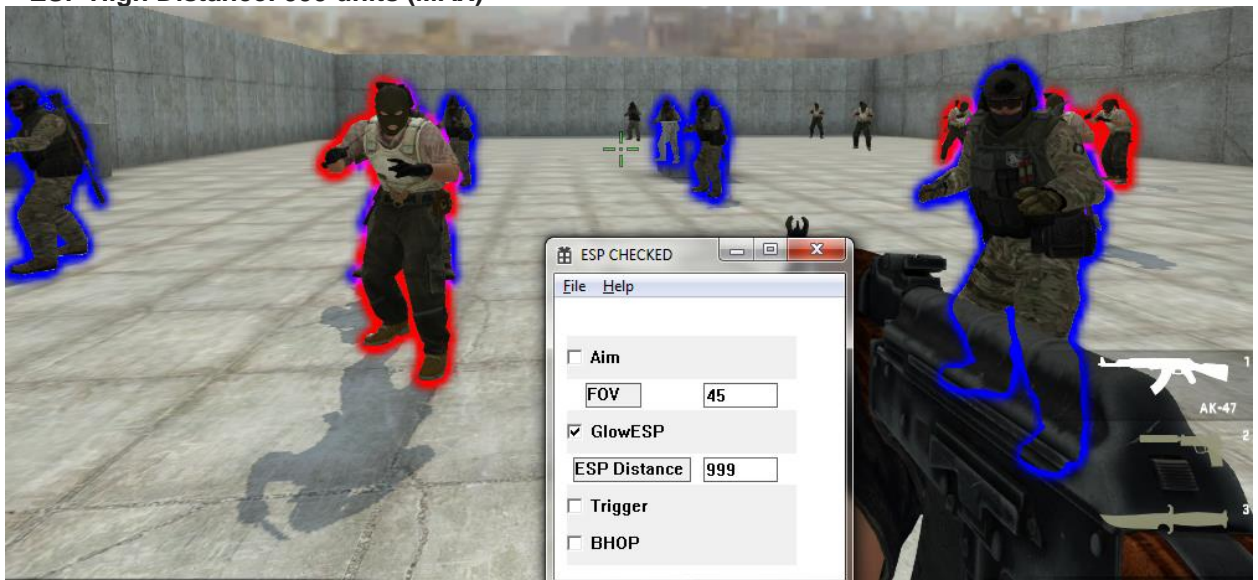


ESP Medium Distance: 600 units

Note: More targets are closing as the distance is increased



ESP High Distance: 999 units (MAX)



2.6.1.4. Verifying Triggerbot

Test: Place the crosshair in a spot where the bot can run into

Pass: When the weapon will fire without use of any buttons

Fail: The weapon will not fire

Results: PASS

2.6.1.5. Verifying Signature Scan

Test: Compare values of scanned offsets against hardcoded offsets

Pass: All scanned values output the same values of hardcoded values

Fail: Scanned output does not match hardcoded values

Results: PASS

Hardcoded values

```
const DWORD teamOffset = 0xF0;
const DWORD healthOffset = 0xFC;
const DWORD iCrosshairID = 0xB2A4;
```

Testing function

Our testing function will test 3 offsets: HP, Crosshair ID, Team Offset

The function will output hardcoded values then output scanned values. This test is a pass if the values are matching.

```
std::cout << "BCIT Project" << std::endl;
cout << "===== " << endl;

DWORD scanHPoff = (DWORD)nScanner->PatternScanExModule(Mem->GetProchandle(), L"client.dll", "\x83\x89\x00\x00\x00\x00\x7F\x2D\x8B\x01", "xxx???xxxx");
DWORD scanCrosshairOff = (DWORD)nScanner->PatternScanExModule(Mem->GetProchandle(), L"client.dll", "\x8B\x81\x00\x00\x00\x00\x85\xC0\x75\x19", "xxx???xxxx");
DWORD scanTeamOff = (DWORD)nScanner->PatternScanExModule(Mem->GetProchandle(), L"client.dll", "\xCC\x8B\x89\x00\x00\x00\x00\xE9\x00\x00\x00\xCC", "xxx???x???x");
DWORD realLocal = Mem->Read<DWORD>(Mem->ClientDLL_Base + playerBase);
int HP = Mem->Read<int>(realLocal + healthOffset);

cout << "Base Addr : " << Mem->ClientDLL_Base << endl;
cout << "Base Size : " << Mem->ClientDLL.modBaseSize << endl;
cout << "===== " << endl;
cout << "HARDCODED VALUES" << endl;
cout << "===== " << endl;
cout << "Crosshair Offset : " << iCrosshairID << endl;
cout << "HP Offset : " << healthOffset << endl;
cout << "TeamID Offset : " << teamOffset << endl;
cout << "TeamID of Hardcoded Team : " << Mem->Read<int>(realLocal + teamOffset) << endl;
cout << "ID of Hardcoded Crosshair : " << Mem->Read<int>(realLocal + iCrosshairID) << endl;
cout << "HP of Hardcoded Local : " << HP << endl;

int scanHP, scanCrosshair, scanTeam;
ReadProcessMemory(Mem->GetProchandle(), (PBYTE*)(scanHPoff + 0x2), &scanHP, sizeof(int), NULL);
ReadProcessMemory(Mem->GetProchandle(), (PBYTE*)(scanCrosshairOff + 0x2), &scanCrosshair, sizeof(int), NULL); // Gets LocalPlayer
ReadProcessMemory(Mem->GetProchandle(), (PBYTE*)(scanTeamOff + 0x3), &scanTeam, sizeof(int), NULL); // Gets LocalPlayer

cout << "===== " << endl;
cout << "SCANNED VALUES" << endl;
cout << "===== " << endl;
cout << "Scan Crosshair Offset : " << scanCrosshair << endl;
cout << "Scan HP Offset : " << scanHP << endl;
cout << "Scan TeamID Offset : " << scanTeam << endl;
cout << "ID of Scanned Team : " << Mem->Read<int>(realLocal + scanTeam) << endl;
cout << "ID of Scanned Crosshair : " << Mem->Read<int>(realLocal + scanCrosshair) << endl;
cout << "HP of Scanned Local : " << Mem->Read<int>(realLocal + scanHP) << endl;
```

Output

```
C:\Users\UC\source\repos\BCIT Project\Debug\BCIT Project.exe
BCIT Project
=====
Scan : 442630144 to 527458304
Scan : 442630144 to 527458304
Scan : 442630144 to 527458304
Base Addr : 442630144
Base Size : 84828160
=====
HARDCODED VALUES
=====
Crosshair Offset :45732
HP Offset : 252
TeamID Offset : 240
TeamID of Hardcoded Team : 2
ID of Hardcoded Crosshair : 7
HP of Hardcoded Local : 80
=====
SCANNED VALUES
=====
Scan Crosshair Offset :45732
Scan HP Offset : 252
Scan TeamID Offset : 240
ID of Scanned Team : 2
ID of Scanned Crosshair : 7
HP of Scanned Local : 80
```

2.6.1.6. Verifying Valve Anti Cheat Undetected

Test: Turn on the Valve Anti Cheat to play online with others

Pass: Player is able to play with the cheat for an extended period of time

Fail: Player is permanently banned by the anti-cheat system

Results: UNKNOWN

2.7. Implications of Implementation

The implications of implementing this project:

- An active video game that is frequently updated are constantly changing how information is managed making it difficult.
- All first-person shooting games can be reversed engineered with the same design.

The implications of implementing an external software:

- Valve's anti-cheating system can detect and scan for this software whether if it can recognize the behavior is another problem.
- Upgrading from external to kernel mode will require the software to be redesigned.

The implications of implementing a pattern scanner:

- Provides flexibility when the game is updated by Valve
- May have a slower start up time as it has to scan through memory to find all of the offsets.

2.8. Innovation

The following are the highlights of the major innovative components of this project

1. **The software is undetected by the Valve Anti-Cheating system allowing users to play with their friends.** This capability is provided through the software designed. The code signature is unique compared to previously detected signatures. Existing software that I have seen are detected by the anti-cheat system.
2. **Pattern Scanning** is unnecessary in this situation and I have not seen any easily available software use this because existing software tend to be detected before the offsets are even changed. With that said, it is still a good innovative component to add since it provides flexibility to prolong the usage of software.
3. **Development for the use of the visually and physically impaired** by providing settings in the program to adjust their difficulty level. Adjusting levels such as brightness of an enemy player, how much the aim assist will help, and how far the extrasensory perception distance will show. All existing software developed have a more malicious intent and is just an on or off switch of features.

2.9. Complexity

In this project, we are dealing with a fully developed game with an anti-cheating system. The goal is to create a software that will allow players to cheat offline. For the hack to work online, it requires a unique signature that Valve Anti-Cheat System does not have in its database. The possibility of being able to bypass the cheating system on the first try would be very difficult as we need to create a unique signature that has not been detected by their anti-cheating system. Each failed attempt can cost about \$15 CAD (cost of the game).

The project will be written in C++ and will require the knowledge of advanced programming, reverse engineering, and the use of tools such as Ollydbg or Cheat Engine to find how to manipulate information stored in memory from the game.

2.10. Research in New Technologies

Technologies that were considered or are used in this project include:

Pattern Scanning: a method to dynamically retrieve offsets for information. Instead of hardcoding offsets into the software, pattern scanning will dynamically scan memory for patterns that match an area that would store offsets. When the game is updated, the offset is changed but the surrounding data will be the same.

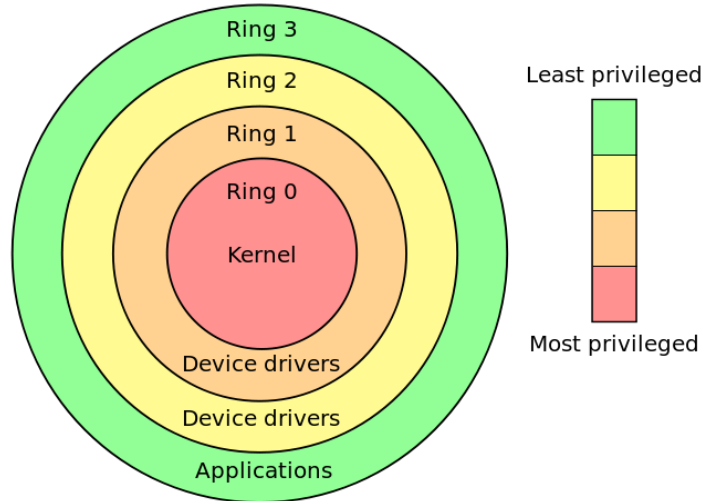
Kernel version: a type of software/driver that operates at a ring 0 level preventing this specific anti-cheating system from scanning this application.

Bone structure for player models: Currently the aim function only aims at the head. Implementing bone structure will provide the ability to change the location of where the aim assist will aim at. Bone structure will add neck, body, legs to the target list.

Recently, a Valve developer was at a conference giving a presentation about their newly developed machine learning technology that gave them the ability to detect software similar to this project which can be very troublesome. Trying to be undetected by a company that have a lot of resources can be difficult and would require innovative thinking to be successful.

2.11. Future Enhancements

In the alternate solutions section above, there are 3 different solutions that can be made and enhanced. Future enhancements can be either to create an internal injection of code into the game where the user interface can be embedded into the games interface. The best enhancement that can be used is making a kernel software. Valve's anti cheating system operates only on Ring 3 alongside this software created for this project meaning that it can only look at Ring 3 applications. If a kernel mode software is created, the chances of being detected by the anti-cheat system lowers considerably. With that being said, Valve's anti cheating system is very tight lipped and not a lot has been said about their development.



2.12. Timeline and Milestones

#	Feature Milestones	Duration	Timeline
1	Project Proposal	1 Week (40 hrs.)	Feb 3 rd 2018
2	Research using Cheat Engine / Ollydbg <ul style="list-style-type: none"> Memory Management Offsets 	1 Week (40 hrs.)	Feb 5 th 2018 – Feb 10 th 2018
3	Extrasensory Perception (Health Point, Name, Equipment) <ul style="list-style-type: none"> Design Code Test / Debug 	2 Weeks (80 hrs.)	Feb 12 th – Feb 24 th 2018
4	Aim assist with Field of View <ul style="list-style-type: none"> Design Code Test / Debug 	2 Weeks (80 hrs.)	Feb 26 th – Mar 10 th 2018
5	Trigger bot <ul style="list-style-type: none"> Design Code Test / Debug 	½ Week (20 hrs.)	Mar 12 th – Mar 14 th 2018
6	Auto Bunny Hop <ul style="list-style-type: none"> Design Code Test / Debug 	½ Week (20 hrs.)	Mar 15 th – Mar 19 th 2018

6	User Interface Compilation of Features <ul style="list-style-type: none"> • Design • Code / Compile • Test / Debug 	½ Week (20 hrs.)	Mar 21 th – Mar 24 th 2018
7	Valve Anti-Cheat Undetected <ul style="list-style-type: none"> • Research 	1 Week (40 hrs.)	Mar 25 th – Mar 30 th 2018
8	Pattern Scanning <ul style="list-style-type: none"> • Design • Code • Test / Debug 	1 Week (40 hrs.)	Mar 31 st 2018 - April 5 th 2018
9	Final Report	1 Week (40 hrs.) Total: 420 hours	Apr 5 th – April 11 th 2018

3. Conclusion

This practicum project will help enhance many of my skills in terms of software development, analyzing software/application operations, reverse engineering software/applications, and understanding signature coding. I have been interested in this topic for some time, but never had the opportunity and technical knowledge to work on it until now. I believe the scope of this project will provide a challenge given my knowledge and past experiences.

3.1. Lessons Learned

While developing this project, I learned about many new technical topics including:

- Memory management by applications
- Detection systems such as anti-cheat
- Analyzing applications at a low level
- Reading/Manipulating information stored in memory
- C++ Language

I was also able to practice and improve upon many existing skills:

- Functional Programming
- Object-Oriented Programming
- Technical Writing

3.2. Closing Remarks

Reverse engineering a video game as always been an interest for me. Most of my childhood has been spent playing games and there is always a side of video gaming that was a bit malicious but unknown. My curiosity always wanted to explore how others developed programs similar to this practicum. After spending four years at BCIT, I felt that I had enough technical knowledge to tackle this problem and have achieved my goal. This practicum has provided me the opportunity to learn about reverse engineering and align my personal goals with my academic goals. I plan to continue and create a kernel mode software as a personal hobby.

4. References

Counter-Strike. (2017, December 17). In *Wikipedia, The Free Encyclopedia*. Retrieved 02:42, February 3, 2018, from <https://en.wikipedia.org/w/index.php?title=Counter-Strike&oldid=815776887>

Counter-Strike: Global Offensive. (2018, February 3). In *Wikipedia, The Free Encyclopedia*. Retrieved 02:42, February 3, 2018, from <https://en.wikipedia.org/w/index.php?title=Counter-Strike: Global Offensive&oldid=823722635>

Valve Anti-Cheat. (2018, February 2). In *Wikipedia, The Free Encyclopedia*. Retrieved 02:43, February 3, 2018, from <https://en.wikipedia.org/w/index.php?title=Valve Anti-Cheat&oldid=823694564>

5. Change Log

#	Change	Page	Section	Date
1	Changed the problem statement to be clearer and with a proper purpose	47	6.2	Feb 16 th 2018
2	Divided problem into sub problems	47	6.2	Feb 16 th 2018
3	Listed key components of the software	49	6.2	Feb 16 th 2018
4	Added software state diagram	49	6.2	Feb 17 th 2018
5	Added pattern scanning to the project for innovation	52	6.2	Feb 18 th 2018
6	Added example users in appendix	58	6.2	Feb 19 th 2018
7	Changed timeline making pattern scan last feature	44	2.12	Mar 12 th 2018
7	Created Final Report			April 5 th 2018

6. Appendix

6.1. Project Supervisor Approvals

[Include written approvals from the project supervisor indicating that they've approved both the proposal and report. Also include any changes that have been approved by the project supervisor. Please note that without written approvals from the project supervisor, the committee may not review the final report.]

6.2. Approved Proposal

Student Background

Education

British Columbia Institute of Technology (BCIT) – Burnaby, BC

- **Bachelor of Technology** | **Graduating 2018**
 - Network Security Administration Option
- **Diploma** | **Graduated 2016**
 - Computer Information Technology

Langara College – Vancouver, BC

- General Studies

Killarney Secondary High School – Vancouver, BC

- High School Diploma | **Graduated 2010**

Experience

Employment

Cowell Auto Group, IT Support – Richmond, BC

| **Apr 2017 – Present**

Description: Monitors and controls computers and peripheral data processing equipment. Enters commands using computer terminal and manages controls on computer and peripheral equipment. Monitors the system for failure or errors and responds by addressing/troubleshooting issues. Experience in location migration of a mid-size company into two new building and assisted in setup of infrastructure.

Projects

ICON Network, Practicum Student – Vancouver, BC

| **Sept 2014 – Dec 2014**

Description: Worked with Product Lead and teammates on researching and developing program to help manage their database. The program identified non-standardized data using the Google API to validate and convert data into a standardized form.

Project Description

This project is for the half practicum and I plan to work alone on this project.

The project's goal is to create a malicious software that would assist players on a specific game to make it simpler. Our software game of focus is Counter-Strike: Global Offensive (CS: GO), an online first-person shooting game (FPS). The software will assist users in areas such as hand-eye coordination and in-game knowledge. Since we are dealing with a first person shooting game, hand-eye coordination is very important on the success of how you play the video-game. Aiming for the head produces the most amount of damage and would be the quickest way of winning a duel against an opponent. Another important aspect of this hack is having advanced knowledge about opponents which this software will provide. In the end, this software will provide an advantage for players.

Problem Statement and Background

Background

Counter-Strike: Global Offensive is a very popular multiplayer first-person shooter game developed by Valve. It is the fourth game in the Counter-Strike series where the first game was released in 2000. Counter-Strike: Global Offensive is played having two teams, terrorist and counter-terrorist. The goal of the terrorist team is to plant the bomb and detonate the bomb on one of the two bomb sites provided on a map or eliminate every counter-terrorist before the round ends. It is the counter-terrorist goal to defend both bomb sites from being planted and detonated by either eliminating all terrorist before a planted bomb or defusing a planted bomb within the bomb timer to detonate. The game is played in a 5 on 5 setting where you can play with your friends. Therefore, in order to excel at a first person shooting video-game it requires good hand-eye coordination some users may not have due to previous accidents or impairments.

This software is considered a hack so it will be protected by their own anti-cheating system and for us to allow a user with disabilities to play with their friends we need to bypass their system. Video games developed by Valve is protected by their anti-cheat system called Valve Anti-Cheat (VAC). VAC sends client application challenges, if expected responses are not received, the user will be flagged for a violation. It uses Signature Scanning to detect possible cheats when scanning computer's memory and processes. When an anomaly is detected, it is compared to their database of banned applications or analyzed by Valve engineers. If the code is confirmed as a new cheat, it will be added to their database.

My interest in this project came from my previous experiences with this video-game. I have thought this video game was a great game to play with friends and about the people who could not play this game due to their own physical impairments.

Problem Statement

The problem this project solves is one where a player, who may have had a life altering incident causing physical or visual impairments such as hand tremors or blindness, is having a difficult time playing or learning how to play a video game (Refer to Appendix 16.3 for examples). CS: GO is played online so there are no difficulty settings for the game. This project will have features that would assist the players in learning the game and allow them to adjust the difficulty of the game if it turns out to be too challenging. This game has been chosen due to the fact that it is an online game with software security measures therefore being able to play online with friends is important.

Main Problem

An impaired person wants to play Counter-Strike online with his friends but discovers that the game is too difficult to play with his disability. (Refer to Appendix 16.3 for examples)

Sub-Problem 1

Person has a visual impairment including but not limited to color blindness, legally blind and etc.

Sub-Problem 2

Person has a physical impairment some of which are hand tremors, loss of limb, and etc.

Main Problem Solution

Develop a software that will provide visual and physical solutions to their disability and allow them to play with their friends.

Sub-Problem 1 Solution

Implement a feature that will allow them visual aid such as change of color or adjust enemy brightness.

Sub-Problem 2 Solution

Implement a feature that will assist them in hand-eye coordination.

Complexity

In this project, we are dealing with a fully developed game with an anti-cheating system. The goal is to create a software that will allow players to cheat offline. For the hack to work online, it requires a unique signature that Valve Anti-Cheat System does not have in its database. The possibility of being able to bypass the cheating system on the first try would be very difficult as we need to create a unique signature that has not been detected by their anti-cheating system. Each failed attempt can cost about \$15 CAD (cost of the game).

The project will be written in C++ and will require the knowledge of advanced programming, reverse engineering, and the use of tools such as Ollydbg or Cheat Engine to find how to manipulate information stored in memory from the game.

Scope and Depth

The minimum scope of the project is to create a software that will allow users to play the CS: GO offline with an advantage against bots. The extended scope of the project is developing a unique code that will be undetected by the anti-cheat system.

A list of features that would be included in this software are:

- Aim Assist – Assistance with aiming at the target's body or head.
 - FOV – Field of view distance where the aim lock will engage a target. The range of the distances will be 0 to 180 degrees.
 - Trigger bot – Assistance with firing a weapon automatically when target is in user's crosshair
- Extrasensory Perception (ESP) – Ability to know where the other players are located with information such as health point, name, and weapon. This feature will have an adjustable setting to adjust how far, in distance, information will be given.
- Auto Bunny Hopping – Ability to continuously jump while holding the jump button. This allows the player to go to areas quicker than normal.
- Undetected by Valve Anti-Cheat – Requires unique code in the hack that has not been stored in the database.

Test Plan

Acceptance Testing

Acceptance testing will be done after each feature of the project is completed. It will involve verifying that each feature passes the manual test described below. After completion of all features, there will be a final test that will verify the compilation.

#	Description
1	Verify auto aim feature Test: Move around a bot standing still without using the mouse to aim Pass: Crosshair locks onto the bot without using mouse Fail: The crosshair comes off the target when moving
2	Verify the Field of View feature Test: Three distances: long, medium, short ranges. Pass: Crosshair locks onto the bot within that general range Fail: The crosshair constantly locks on or does not lock within the distance given.
3	Verify trigger bot feature Test: Place the crosshair in a spot where the bot can run into Pass: When the weapon will fire without use of any buttons Fail: The weapon will not fire
4	Verify Extrasensory Perception feature Test: Place a bot on the other side of a wall to see if information can be retrieved Pass: Information is given about the bot including equipment, name, health point Fail: There is no information given
5	Verify Bunny Hopping feature Test: Holding the jump button will allow for continuous jumping Pass: Player is continuously jumping and increasing speed after each jump Fail: Player stops jumping after first attempt
6	Verify Valve Anti Cheat Undetected feature: Test: Turn on the Valve Anti Cheat to play online with others Pass: Player is able to play with the cheat for an extended period of time Fail: Player is permanently banned by the anti-cheat system

Methodology

Methodology and Approach

The feature requirements of this program are already known therefore the software development methodology that will be used for this project is waterfall. Each feature will be developed individually starting with design, coding and then testing. The success of each feature will allow for the progression of the project. In the final stages of the project, all features will be integrated into one with a user interface.

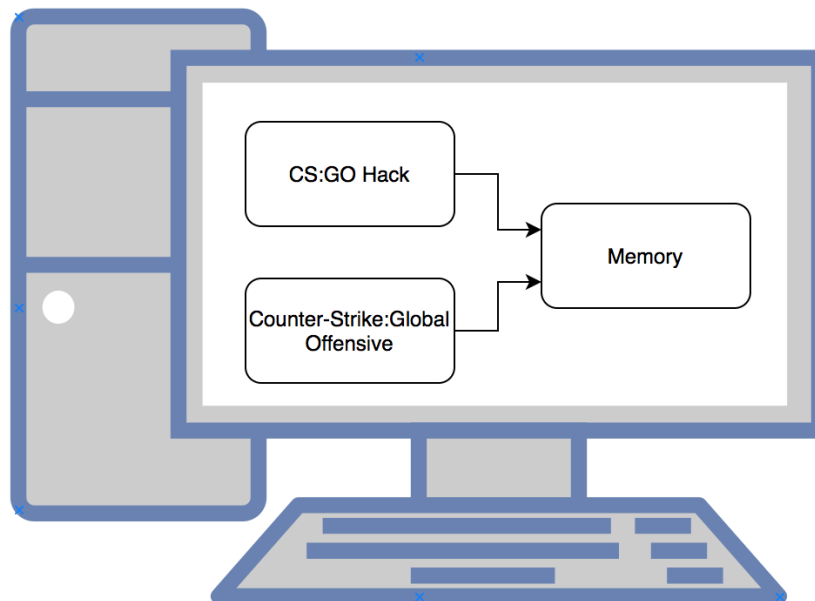
Technologies and Tools

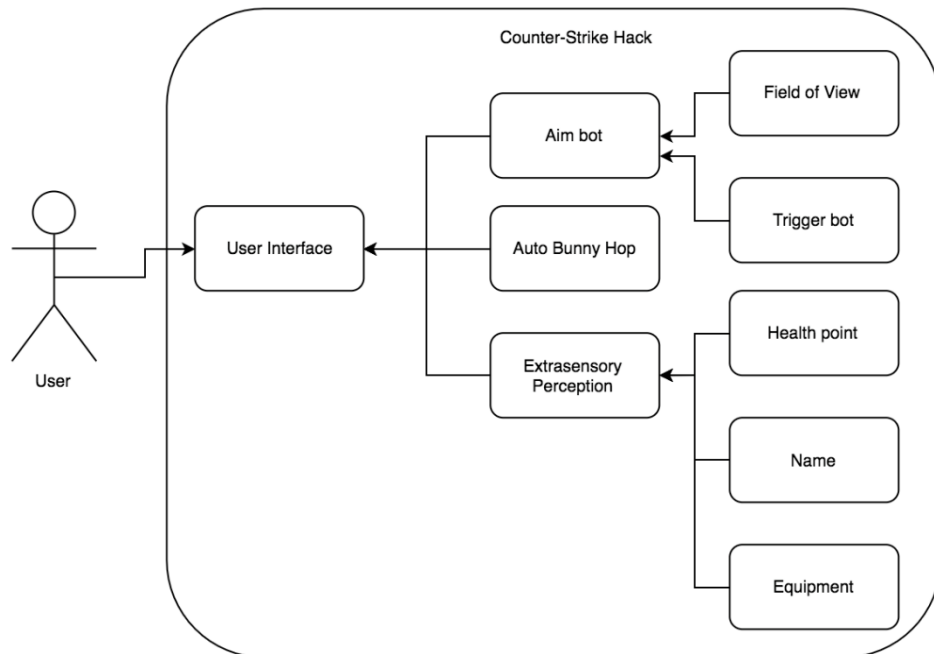
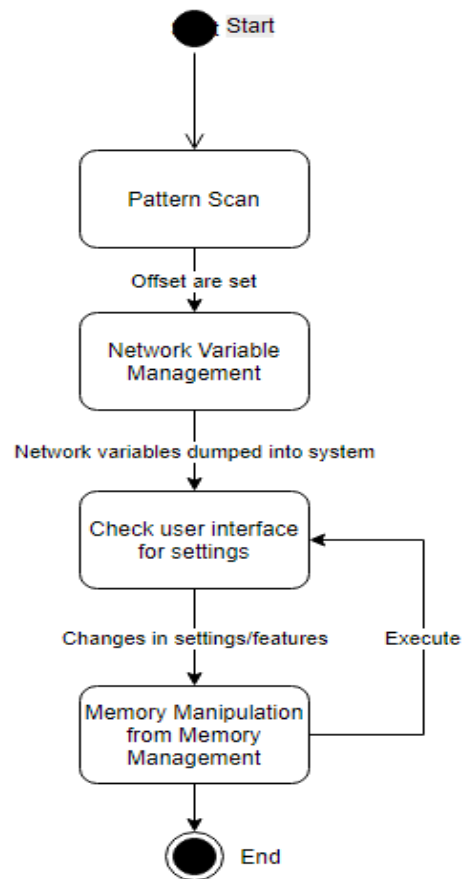
Software/Tools that will be used in this project:

- **Visual Studio** – Developing in C++. Visual Studio is an integrated development environment from Microsoft. It is used to develop computer programs, as well as web sites, web apps, web services and mobile apps
- **Olllydbg** – Code analyzer for game. Ollydbg is a 32-bit assembler level analyzing debugger for Windows
- **Cheat Engine** – Researching where the game stores information in memory. Cheat Engine is an open source tool designed to help you with modifying games running under window
- **Draw.io** – Used to design system/software architectural diagrams. Draw.io is a free online diagram software for making flowcharts, process diagrams, org charts, UML, ER and network diagram
- Copy of **Counter-Strike: Global Offensive**

System/Software Architecture Diagram

The following diagram is a software architecture diagram that visually represents the interaction of programs and features





There are three different types to software that can be created for this situation:

- **Internal:** a dynamic-link library (DLL) must be injected directly into the game and from there you can hook the video game functions. This method allows you to directly read and write the games memory through raw pointers.
- **Internal/External hybrid:** a DLL that must be injected into any 32-bit application and creates a handle to the game process and read/write.
- **External:** an execution(EXE) file that directly creates a handle to read/write the game process. The main disadvantage is that they are not in sync with the game's thread which may sometimes lead to unexpected results.

With our time constraint, an **external** software will be created. The software will consist of a few components which are **memory and netvar management, and pattern scanning**. These three functions will allow us to grab and manipulate memory information and help build the user features.

Memory management: This will allow the software to attach to the game process and grab modules information provided by the game. We will use functions from Windows Application Programming Interface (WinAPI) such as Read Process Memory (RPM) and Write Process Memory (WPM) to retrieve and manipulate information in memory.

Pattern Scanning: The game stores his memory by using offsets after picking a base address. Memory addresses are always changing after each time the game is restarted but the offsets are always the same except when developers update them periodically. Instead of hard coding the offsets each time the developers change them, this software will have a pattern scanner that will update the offsets dynamically.

Network Variable Management: Since this is an online game, there will be network variables that stores information about the online match. The manager will dynamically grab network variables and dump them for use.

After these components are developed, we can start to design and develop user functionality of the software such as aim assist, extrasensory perception and etc. For example, using a trigger bot will, utilize these three components and WinAPI mouse events, clicking the mouse to shoot given the enemy location is within the crosshairs.

Innovation

The following are the highlights of the major innovative components of this project

- **The software is undetected by the Valve Anti-Cheating system allowing users to play with their friends.** This capability is provided through the software designed. The code signature is unique compared to previously detected signatures. Existing software that I have seen are detected by the anti-cheat system.
- **Pattern Scanning** is unnecessary in this situation and I have not seen any easily available software use this because existing software tend to be detected before the offsets are even changed. With that said, it is still a good innovative component to add since it provides flexibility to prolong the usage of software.

- **Development for the use of the visually and physically impaired** by providing settings in the program to adjust their difficulty level. Adjusting levels such as brightness of an enemy player, how much the aim assist will help, and how far the extrasensory perception distance will show. All existing software developed have a more malicious intent and is just an on or off switch of features.

Technical Challenges

The technical challenges in this project includes:

- **Analyzing how the game operates at a low level:** Using open-sourced tools, we are able to analyze how games store their information in memory. This technique has been taught at a very basic level in our program and will be an important part of developing this cheat.
- **Reverse engineering the game:** Analyzing the operation of the game will allow us to reverse engineer the game. Developing a software that will manipulate the game at a low level.
- **Bypassing an anti-cheat system:** This will be the most challenging part. The anti-cheat system can be seen as a virus scanner and we are attempting to do is create malware that will go undetected when operated online. The developed cheat will be developed offline to get the basic understanding of the first two challenge before the attempt of creating a unique signature. This is the last part of the challenge as it can cost money if unsuccessful.

Development Schedule and Milestones

#	Feature Milestones	Duration	Timeline
1	Project Proposal	1 Week (40 hrs.)	Feb 3 rd 2018
2	Research using Cheat Engine / Ollydbg <ul style="list-style-type: none"> • Memory Management • Pattern Scanning • Network Variable Management 	1 Week (40 hrs.)	Feb 5 th 2018 – Feb 10 th 2018
3	Extrasensory Perception (Health Point, Name, Equipment) <ul style="list-style-type: none"> • Design • Code • Test / Debug 	2 Weeks (80 hrs.)	Feb 12 th 2018 – Feb 24 th 2018
4	Aim assist with Field of View <ul style="list-style-type: none"> • Design • Code • Test / Debug 	2 Weeks (80 hrs.)	Feb 26 th 2018 – Mar 10 th 2018
5	Trigger bot <ul style="list-style-type: none"> • Design • Code • Test / Debug 	1 Week (40 hrs.)	Mar 12 th 2018 – Mar 17 th 2018
6	Auto Bunny Hop <ul style="list-style-type: none"> • Design • Code • Test / Debug 	1 Week (40 hrs.)	Mar 19 th 2018 – Mar 24 th 2018

6	Compilation of Features <ul style="list-style-type: none"> • Design • Code / Compile • Test / Debug 	½ Week (20 hrs.)	Mar 26 th 2018 – Mar 28 th 2018
7	Valve Anti-Cheat Undetected <ul style="list-style-type: none"> • Research • Design • Code • Test 	1 Week (40 hrs.)	Mar 29 th 2018 – Apr 4 th 2018
8	Final Report	1 Week (40 hrs.) Total: 420 hours	Apr 5 th 2018 – April 11 th 2018

Deliverables

The deliverables for this project are as follows:

- **Counter-Strike: Global Offensive Hack**
- **Final Report**
- **User Guide**

Conclusion and Expertise Development

This practicum project will help enhance many of my skills in terms of software development, analyzing software/application operations, reverse engineering software/applications, and understanding signature coding. I have been interested in this topic for some time, but never had the opportunity and technical knowledge to work on it until now. I believe the scope of this project will provide a challenge given my knowledge and past experiences.

Proposal Appendix

16.1 Gameplay



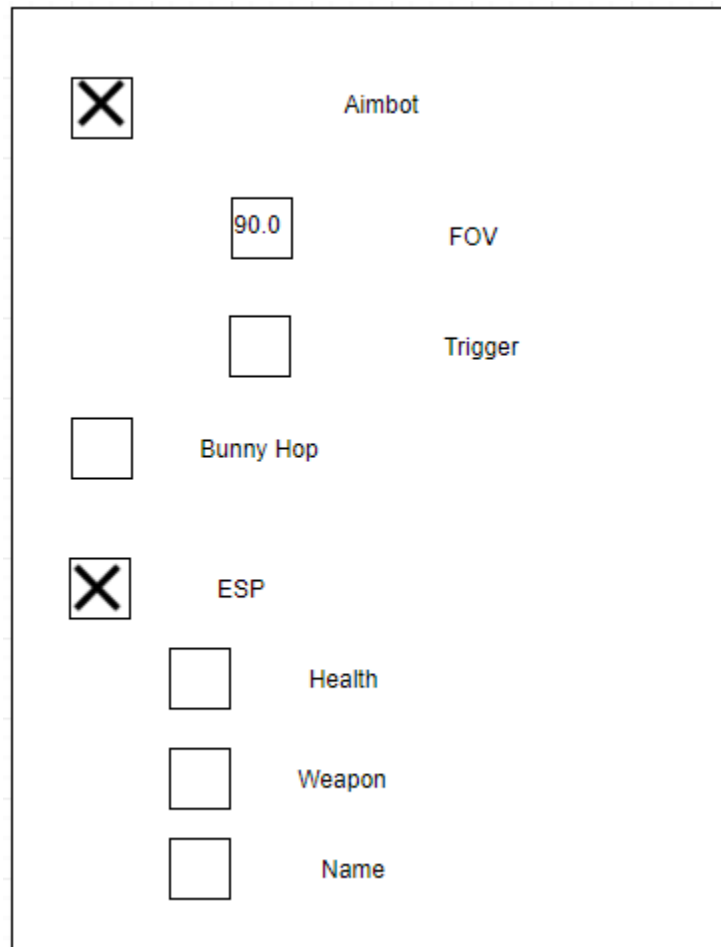
Visual of a regular game situation. In this scenario, there are three terrorists and one counter-terrorist in a duel

16.2 Extrasensory Perception



This is the potential visual of what the ESP function would see. Information such as name, weapon, health bar, and even the location (Note: There is a second counter-terrorist behind the wall that can be seen by using ESP)

16.2 UI Mockup



Rough draft of the UI. The UI interface will look different depending on if the cheat is built for internal or external as well. Internal hack would be built into the game visuals, when a user clicks a set button it will be an overlay of the game. External hack would provide an UI outside of the game like command prompt or program itself.

16.3 Example users



Retrieved from <https://www.youtube.com/watch?v=HZgUt7VHcis>

This is a man who goes by the alias, handi. He was born without limbs but still enjoys playing CS: GO. He was a well-known player streaming the video game on Twitch, a video streaming platform for gamers. This project can help players, similar to his situation, have a better experience playing the game.



Retrieved from <https://www.youtube.com/watch?v=nQOz-NSYv-w>

A legally blind and deaf man playing CS: GO. He goes by the alias loop and was a popular streamer for a short period of time. Even with birth defects, he still enjoys the challenge of the game. The project can help new players ease into the game with visual aids and lowering the learning curve.