Machine Learning Cheating System

This document has general project information regarding the requirements for CSS 497 as well as the design and implementation criteria specific to the product that is being created.

[**Introduction**](#_heading=h.8pt58tvceyn0) **2**

[**Goals & Requirements**](#_heading=h.1fob9te) **2**

[Project Goals - (objectives)](#_heading=h.uth973310w9q) 2

[Software Requirements](#_heading=h.o7yr8qxqyuv8) 3

[Functional Requirements](#_heading=h.3uyq7wr65fcc) 3

[Non-Functional Requirements](#_heading=h.908uypfua9zy) 3

[User Stories](#_heading=h.r31ikxdyqjzb) 3

[**Software Architecture & Design**](#_heading=h.mondxveeya6c) **4**

[Architecture & Design of Computer Vision Machine Learning Code](#_heading=h.hw16vist4aga) 4

[Activity Diagram](#_heading=h.pdspai6oo3nk) 4

[UML Code Level Diagram](#_heading=h.wzepcp3nid49) 5

[Components needed](#_heading=h.bsds3vaouqz3) 6

[Component Implementation Diagram](#_heading=h.s21kytblmq6e) 6

[Existing Technology](#_heading=h.4hl7ncim09wf) 6

[Python libraries](#_heading=h.ehifvmg4mpnz) 6

[Developed Technology](#_heading=h.96nsydwohvbx) 6

[**Timeline**](#_heading=h.3znysh7) **7**

[Deliverables](#_heading=h.y2r78nn7dyo5) 7

[Gantt Chart](#_heading=h.2hdplkoy99kk) 8

[Software Development Life Cycle](#_heading=h.xmd4m34bavot) 9

[**Implementation & Execution**](#_heading=h.7cqsfiej2nk) **9**

[Visual Stream](#_heading=h.myziwjl2e4wc) 9

[Remote Video Stream](#_heading=h.dwcok0y38e69) 9

[Window Capture](#_heading=h.rj9q4wyupda4) 10

[Matching and thresholding](#_heading=h.ghc81e6bw516) 10

[Matching](#_heading=h.s4r7l8ittdqs) 10

[Thresholding](#_heading=h.ycllxnj4tuj4) 10

[Overlapping Positive Matches](#_heading=h.b41rqz6xm44e) 11

[Grouping Matches](#_heading=h.c9p2qb635p8k) 11

[Click points and Rectangles](#_heading=h.i8fizs250dl3) 11

[HSV Thresholding](#_heading=h.k2sbxnnr0rvg) 12

[Color Conversion](#_heading=h.opkk41ba5i8y) 12

[Sliders](#_heading=h.u6wtg7zy272) 13

[Remote Botting](#_heading=h.wx1phth09ezp) 14

[**Conclusion**](#_heading=h.4351dzt8zfi9) **15**

[**Learnings**](#_heading=h.ux4om8ns29ch) **15**

[**Future Work**](#_heading=h.e6ysp0sa29ob) **15**

# Introduction

The Goal of this CSS 497 project is to create a machine learning-based system that will allow a user to circumvent kernel 0 anti-cheat-based systems. This will be achieved by implementing a two-pc system to transfer a screen grab of the game being played and sending it over to a secondary pc. The secondary pc will handle the target identification, tracking, and execution of the commands back to the gaming pc.

This project aims to demonstrate the skills and abilities acquired during my study at the University of Washington Bothell’s CSSE: Information Assurance and Cybersecurity program. As well as implement and gain a more thorough understanding of the software development process. Furthermore, I wish to challenge myself and the system and show the future of video game cheating and analyze how to counteract these future cheating solutions.

# Goals & Requirements

## Project Goals - (objectives)

This software is

1. Create a cheating program that can circumvent kernel 0 anti-cheat systems
2. Create a visual AI that can recognize targets inside of the game Aim Labs
3. Create an AI that can prioritize to the closest target and send commands back to the user's PC to move the cursor onto the nearest target and fire
4. Create an analysis of the defense against future cheating systems like this project
5. Demonstrate to video gaming companies that use kernel 0 anti-cheats while effective in the present atmosphere their invasiveness is not justified simply because they work.
6. Complete the “Gridshot” challenge in aim labs

## Software Requirements

### Functional Requirements

1. The software must be able to track one target
2. The software must be able to calculate and send commands to move the cursor over a target for the user
3. The software must be able to fire for the user at one target
4. The software should be able to track multiple targets
5. The software should be able to prioritize the closest target between two targets
6. The software should be undetectable to kernel 0 anti-cheat systems
7. The software should score higher than most users on aim challenges

### Non-Functional Requirements

## User Stories

1. As a video game cheater, I want to be able to cheat in any video game I want to regardless of the level of anti-cheat so that I can have fun in whatever game I decide to play.
2. As a video game cheater, I want to have my cheats aim for me so I can have the top record on every aiming challenge on the game Aim Labs
3. As a video game developer, I want to be able to see what the future of video game cheating may look like and how to protect my games from future cheating software

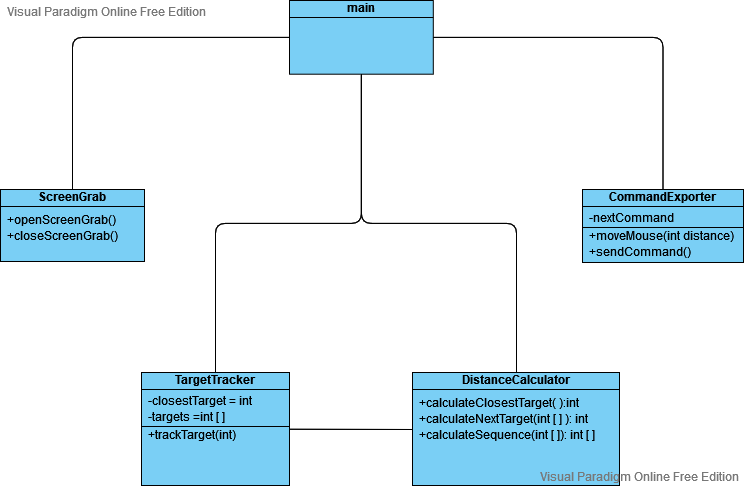
# Software Architecture & Design

## Architecture & Design of Computer Vision Machine Learning Code

### Activity Diagram

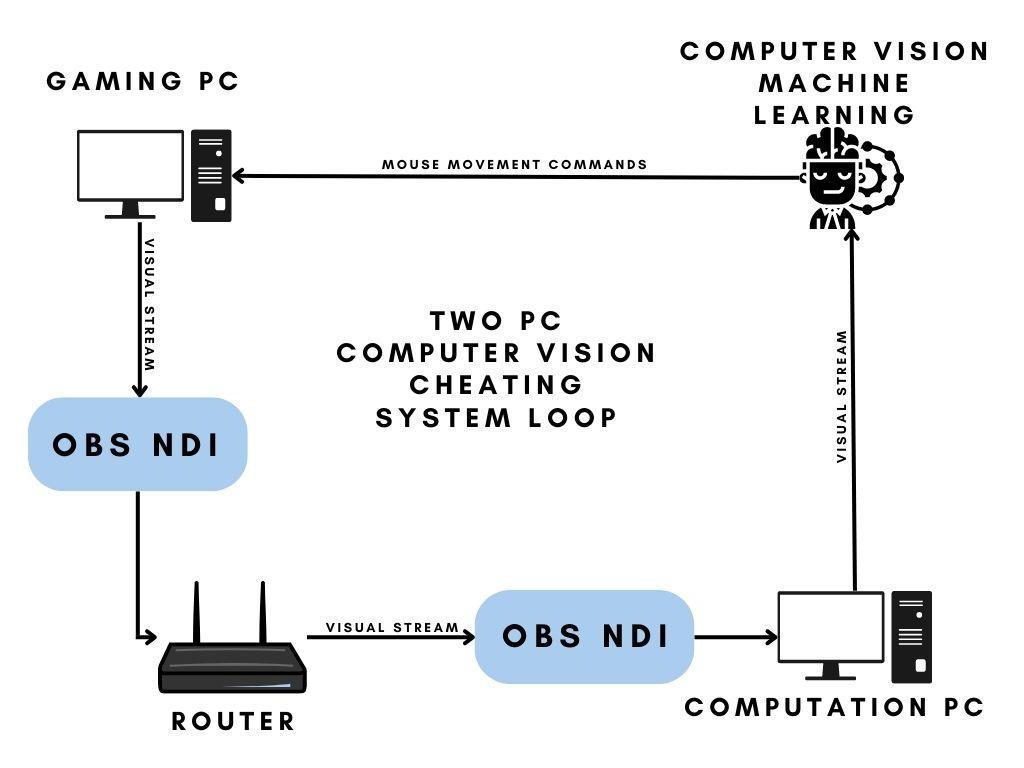
## 

### UML Code Level Diagram

****

## Components needed

### Component Implementation Diagram



### Existing Technology

1. OBS
2. OBS NDI Plugin
3. Router
4. Two Pc’s with Windows OS Installed
5. Python

### Python libraries

1. PIL
2. NumPy
3. cv2

### Developed Technology

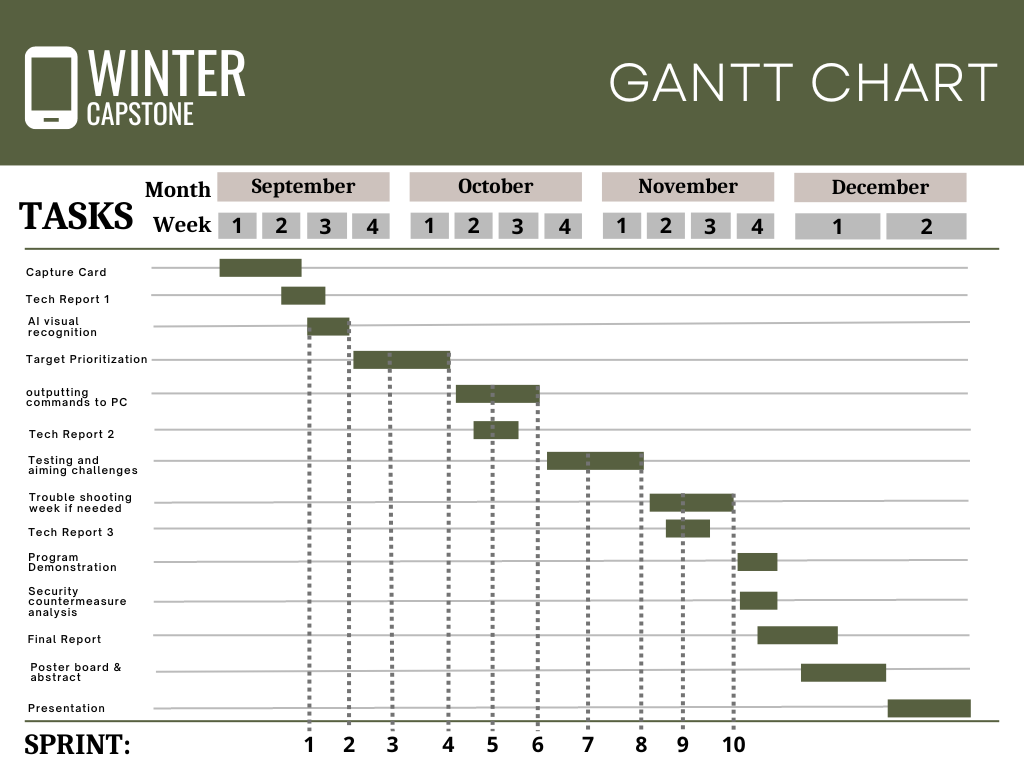
1. Computer Vision Machine Learning Python code
   1. Python Computer vision capture
      1. Captures content brought in over OBS and feeds it into python
   2. Python Target Tracking code
      1. Analyzes the video stream and tracks targets on the stream
   3. Python Target prioritization code
      1. Prioritizes the closest target if multiple targets exist
   4. Python Aiming Command code
      1. calculates based on crosshair and distance to target the commands that need to be sent back to the gaming PC to place crosshair over the target and then fire

# Timeline

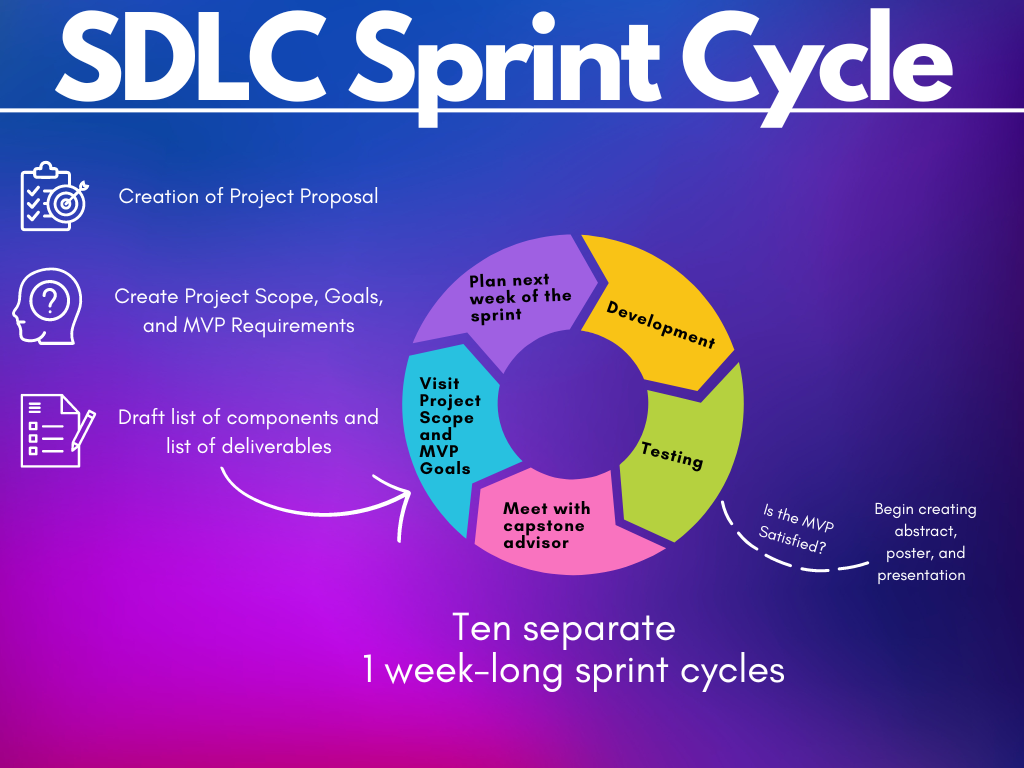
## Deliverables

* Security countermeasures analysis
* Final Report
* Capstone Posterboard
* Abstract
* Capstone Presentation
  + Demo of Program functionality

## Gantt Chart



## Software Development Life Cycle



# Implementation & Execution

## Visual Stream

### Remote Video Stream

a software-based solution for sharing content between two PCs. At first, I tried using OBS and streaming over a private UDP address but the receiving feature was only available on Linux machines so I gave up on that strategy.

I then thought about a strategy that would be accessible to most people which would be to point a webcam at the screen and just analyze the feed but again the quality of the stream would be poor and could cause issues for the AI.

The best solution I found was to use a plugin for OBS (a streaming program) called OBS NDI. This plugin allows me to stream the content over IP from the gaming machine through the router and then the router searches for the other machine on the network with OBS software that is listening on a specific port. The second machine is then able to receive the content from the first machine with about a half-second delay which I believe is sufficient for this project's proof of concept. The delay can also be brought down by experimenting with encoding standards for the video stream which is something I plan to do if I have the time.

### Window Capture

Using the Numpy and windows32 python libraries I created a way for python to capture the screen of my computer system or a program window. The system takes screenshots from the video capture and does data value management and conversions to make the format readable for pythons OpenCV library.

## Matching and thresholding

### Matching

Using the python library cv2 I make use of the matchTemplate function to find matches of my input image from a screenshot of the window capture. The function tends to overlap multiple matches close together because it finds multiple similar pixels and doesn't truly understand the object it is looking for just the general pixels. The function finds matches that have x, and y values close together and group them into click points.

### Thresholding

The Thresholding value assigns a confidence parameter to each match. Basically how confident or how much of the image matched up with the provided target file. The threshold can be set to where only matches that are above a certain confidence percentage value are displayed.

## Overlapping Positive Matches

### Grouping Matches

### 

### Click points and Rectangles

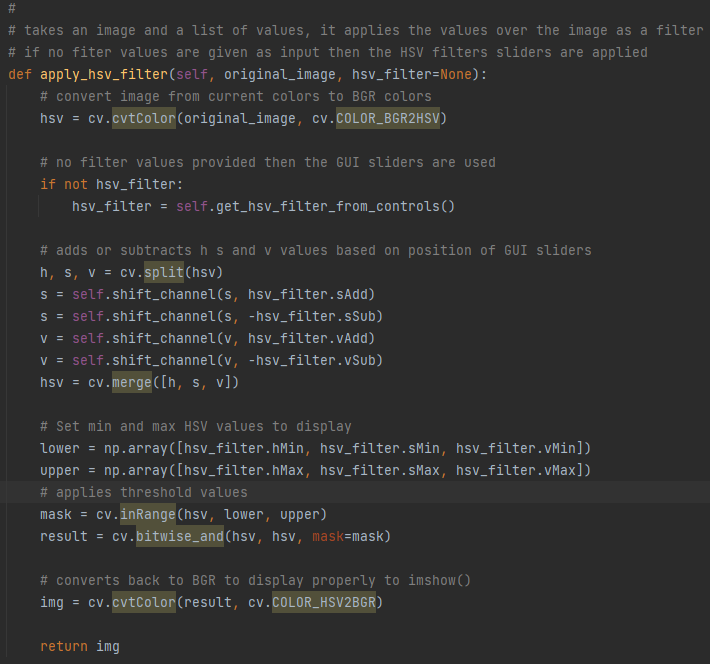




## HSV Thresholding

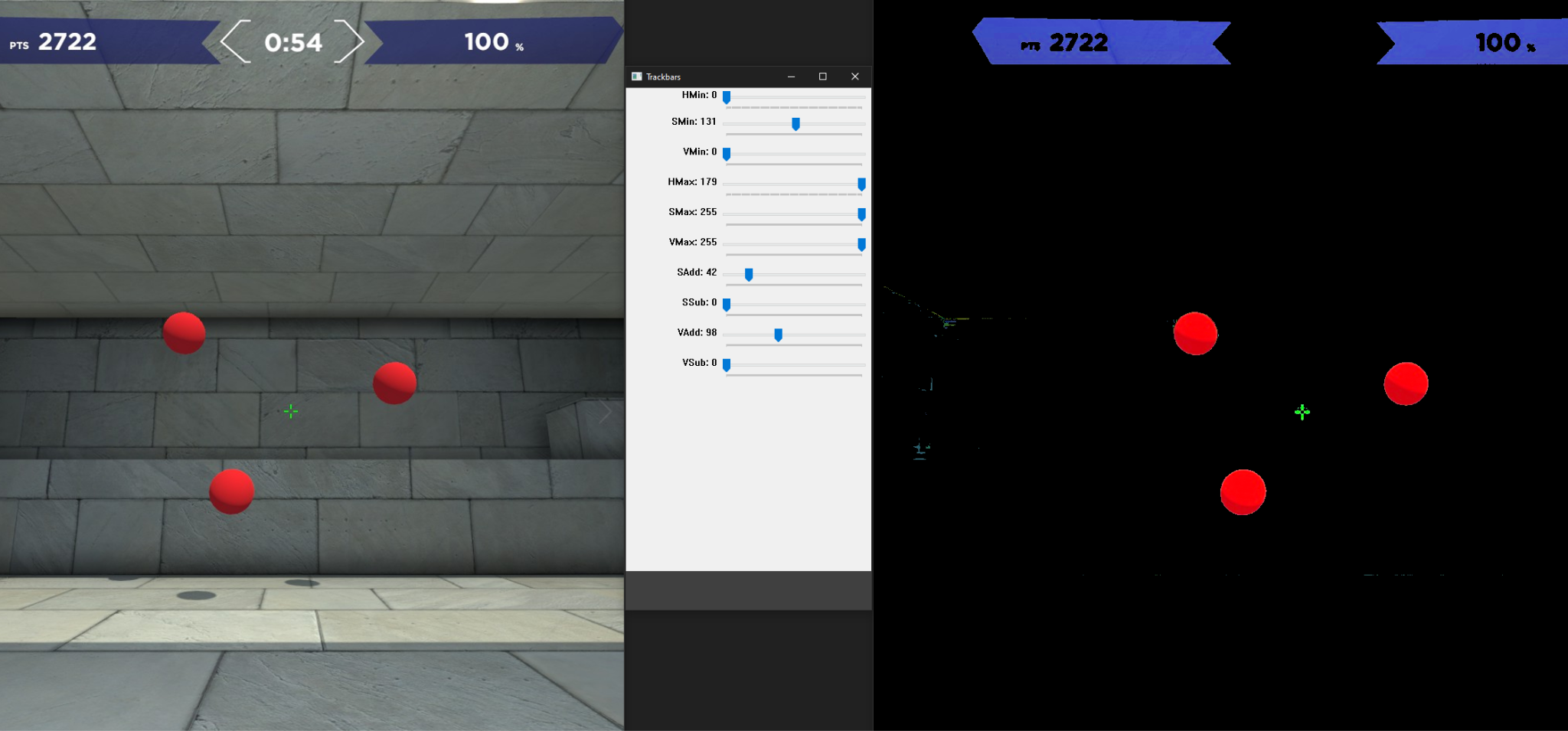
### Color Conversion

The original image is converted from the BGR range to the HSV range. By manipulating the HSV sliders we can remove certain color ranges from the image and remove shadows so the initial image we are attempting to detect is simplified. This speeds up the detection process by simplifying the initial image.

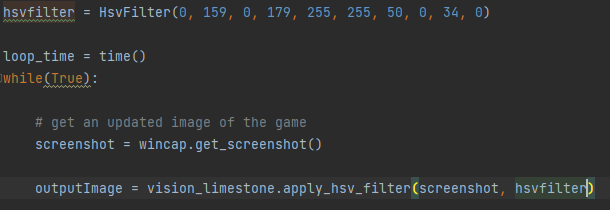


### Sliders

By manipulating sliding values we are able to remove certain color ranges from the image. On the left is the input image and on the right is the processed image that is used for searching. It becomes much easier to discern the targets and crosshair against the black background

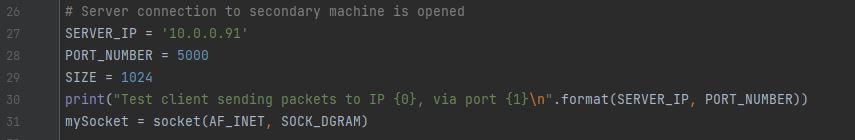


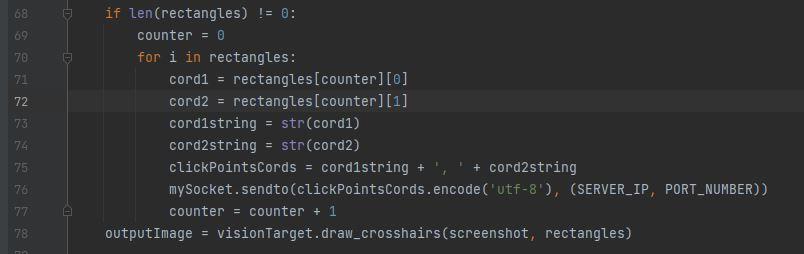
Furthermore, I can manipulate the values of the original image till I get the processed image to the desired state. Then I save a small image of the target in the processed state and that becomes our new match image. I can also save all the values to a variable and simply input it whenever I call the apply\_hsv\_filter function. That way I only have to deal with the sliders once.



## Remote Botting

The remote botting segment was achieved by opening a socket communication between the two computers via python.





# Conclusion

My project did not entirely bypass any Kernel 0 anti-cheat systems mainly due to the lack of ability to test. I do believe that given the ability to test this system against a Kernel 0 anti-cheat I could find a way to bypass such systems either with this solution or a similar one. There are already technologies such as alien boxes which modify the controller that is plugged into the computer/console and used to play the game.

I don’t believe the future of anti-cheat software lies in escalated permissions and invasive software. I believe systems that monitor user gameplay with machine learning to analyze and detect machine-like movements in players’ gameplay will be a favorable and less intrusive alternative. Furthermore, an anti-cheat based around user gameplay could be tailored to the skill level of the players being monitored. Meaning you can use silver-rank players' gameplay to make a silver-level anti-cheat. This would not only detect irregular movements and performance from players but would also reduce the number of false flags against high-level players and would help prevent players from playing below their skill level.

# Learnings

My key learnings and skills developed from this challenging project are my proficiency in the python programming language. I have also built on my learning from CSS 382 Intro to AI by learning more about Machine learning and its applications of computer visions through the OpenCV python package.

# Future Work

I would love to be able to test this system against an actual kernel 0-level anti-cheat system and see what changes would need to be made to my software to bypass them with this strategy. For example, I do know through research that the command I sent over the python socket is blocked by most anti-cheats so I would find an alternative way to move the mouse. I would also like to program this in a more efficient language to increase the reaction time of my aim bot and see if I could get this software to run on a less powerful computer than mine or perhaps even a smaller device. I used python due to my familiarity with using python in an AI context. Python is a great language but not always the best answer when it comes to efficiency. Finally, I would like to explore ways to counteract the type of cheating I am attempting to perform. I would love to explore the idea of machine learning-based anti-cheat systems rather than Kernel 0 anti-cheat systems.