Garcia, Cayden

Cgarcia174@stmarytx.edu

SLackr system manual

A comprehensive Valorant stat tracking system with a comedic twist, combining advanced API integration, statistical analysis, and user-friendly design.

Contents

[Overview 1](#_Toc184048591)

[Background on Valorant 1](#_Toc184048592)

[Valorant Trackers 1](#_Toc184048593)

[Description of SLackR 1](#_Toc184048594)

[Purpose of this Document 1](#_Toc184048595)

[Installation 2](#_Toc184048596)

[Steps to install and Run the system 2](#_Toc184048597)

[Source Files Needed 2](#_Toc184048598)

[Directory Structure 3](#_Toc184048599)

[Where to get the files 3](#_Toc184048600)

[Requirements 4](#_Toc184048601)

[Elements Needed to Run the System 4](#_Toc184048602)

[User Knowledge Requirements 4](#_Toc184048603)

[Hardware Requirements 4](#_Toc184048604)

[Software Requirements 4](#_Toc184048605)

[Functional Requirements 5](#_Toc184048606)

[Design 5](#_Toc184048607)

[How SlackR is put together 5](#_Toc184048608)

[Flow 6](#_Toc184048609)

[Major Modules 6](#_Toc184048610)

[Major interfaces 7](#_Toc184048611)

[Data/Data Structures 10](#_Toc184048612)

[Source Code 13](#_Toc184048613)

[The Files 13](#_Toc184048614)

[Conventions 14](#_Toc184048615)

[Modifications 15](#_Toc184048616)

[Instructions for making changes 15](#_Toc184048617)

[Potential Enhancements with More Time 15](#_Toc184048618)

[References 17](#_Toc184048619)

[Web scraping and API integration 17](#_Toc184048620)

[UI Design and Modular Programming 17](#_Toc184048621)

[Statistical Analysis and Visualization 17](#_Toc184048622)

[Appendix 18](#_Toc184048623)

# Overview

## Background on Valorant

Valorant, developed by Riot Games, is a tactical first-person shooter (FPS) that blends precise gunplay with unique agent abilities. Released in 2020, the game has quickly grown into one of the most competitive esports titles, boasting millions of players worldwide. Matches feature two teams of five players battling across various maps, requiring strategic coordination, individual skill, and adaptability. Players choose from a roster of agents, each with specialized abilities that influence gameplay.

## Valorant Trackers

In the competitive realm, understanding performance metrics is crucial. Valorant trackers provide statistical insights into gameplay, enabling players to improve by identifying strengths and weaknesses. These tools track metrics such as kills, assists, deaths, headshot percentages, and map win rates. However, many trackers focus solely on raw data, lacking personalized insights or creative interpretations of player stats.

## Description of SLackR

SlackR is a standalone Valorant stat tracker designed to offer a unique, comedic twist on the tracking experience. While delivering standard metrics like agent usage and win rates, SlackR differentiates itself by diving into less conventional statistics, such as knife kills, 149 Damage Done, and Multi-Kills. It further adds humor by assigning users a rank and archetype based on their performance.

## Purpose of this Document

This document serves as the **System Programmer's Manual** for SlackR. It provides developers with the necessary guidance to understand, install, modify, and maintain the system. From its architecture and design principles to specific source code details, the manual aims to ensure smooth development and collaboration. Whether for bug fixes, feature updates, or future enhancements, this manual is the definitive resource for SlackR's development lifecycle.

# Installation

The following instructions explain how to set up, compile, install, and run the SlackR system. This section includes a list of required source files, where to obtain them, and their correct placement in the system directory

## Steps to install and Run the system

1. **Clone the SlackR Repository**
   * Obtain the source code by cloning the SlackR repository. Use the following command in your terminal

*git clone* [*https://github.com/CayG22/SlackR.git*](https://github.com/CayG22/SlackR.git)

* + Alternatively, download the repository as a ZIP file and extract it to your desired location

1. **Install required Libraries**
   * Using the README.txt, install the required python libraries/extensions needed to run SlackR
2. **Run the System**
   * To execute SlackR, use the following command in the terminal

*python main.py*

* Follow any on-screen prompts or instructions to begin data retrieval

## Source Files Needed

The following files and directories are necessary for the system to function

* **Main program Files**
  + *main.py:* Entry point for the SlackR system
* **Modules**
  + *class\_file.py:* Handles all objects in the SlackR system
  + *stats\_file.*py: Handles all non-object function calls within the SlackR system
  + *utils.*py: Handles opening of json data
  + *rank\_config.*py: Handles rank creation for player object
* **Resource Files**
  + *game.json:* Holds game data for a Valorant match
  + *characters.json:* Holds character data for a player
  + *player.json:* Holds a specific player’s data
  + *recentMatches*.*json:* Holds recent match data for a specific player
  + *weapons.json:* Holds weapon data for a specific player
  + RankImages: Holds all rank images used within the SlackR system
  + *README.md:* Documentation for end-users

## Directory Structure

Ensure the SlackR repository is structured as follows:

-SlackRSystem

-Python files

*-main.py*

*-rank\_config.py*

*-class\_file.py*

*-stats\_file.py*

*-utils.py*

-rankImages

-*“rank”.png*

*-characters.json\**

*-game.json\**

*-player.json\**

*-recentMatches.json\**

*-weapons.json\**

\*These files will not be created until AFTER the first time you run the system. They will also change each time a new player is created within the system.

## Where to get the files

All files are available in the SlackR GitHub repository:

<https://github.com/CayG22/SlackR>

# Requirements

To successfully run and utilize SlackR, the following requirements must be met. This section outlines the necessary elements, user prerequisites, hardware and software specifications, and the functional capabilities of the system.

## Elements Needed to Run the System

1. **Access to Internet**
   * Internet access is required to access the Strats.gg API endpoint
2. **Python Environment**
   * A properly configured Python runtime environment
3. **SlackR Codebase**
   * Access to the SlackR repository and necessary files

## User Knowledge Requirements

Users running or modifying the system should have:

1. Basic knowledge of Python programming
2. Familiarity with making and parsing API requests using the *requests* library
3. Understanding of JSON data structures and how to manipulate them
4. (Optional) Awareness of how to use data visualization libraries like *Matplotlib* or *Pandas*
5. Knowledge of Valorant gameplay and statistical concepts for effective use

## Hardware Requirements

1. **Processor:** Minimum dual-core processor, recommended quad-core or higher
2. **RAM:** At least 4GB (8GB recommended)
3. **Storage:** At least 5 GB for the codebase and additional storage for data logs or cached data
4. **Network:** Stable internet connection for API requests

## Software Requirements

1. **Operating System**
   * Windows 10
2. **Python Version**
   * Python 3.9 or higher
3. **Required Libraries**
   * *Requests*
   * *PySimpleGUI*
   * *numpy*
   * *itertools*
   * *json*
   * *statistics*
   * *os*
   * *pandas*
4. **API Request Generator**
   * Use any, this allows for ease of access in modifying GET requests
   * In my case I used Insomnia
5. **API Configuration**
   * Ensure that the Strats.gg website is available and that your API requests are going through

## Functional Requirements

For programmers attempting to use or modify SlackR:

1. **Understand API Configuration**
   * Be able to set up and test API endpoints for retrieving data
   * Update API keys or endpoint URL’s in the configuration files as needed
2. **Navigate the Codebase**
   * Understand the modular structure of SlackR to locate specific functionalities like data retrieval, transformation, and visualizationFlo
3. **Extend the System**
   * Add new APIs or modify existing endpoints
   * Incorporate new metrics or features as needed
4. **Debug and Handle Errors**
   * Use logging and error-handling mechanisms to identify and resolve issues in API calls or data processing
5. **Integrate New Libraries**
   * Add or replace third-party libraries while ensuring compatibility with existing components
6. **Test Functionality**
   * Use built-in or custom test cases to verify system integrity after modifications

# Design

The design of SlackR focuses on creating a system that is both modular and extensible, ensuring ease of maintenance and future scalability. Each component has a clearly defined purpose, contributing to a seamless flow from data retrieval to actionable insights. This section provides an overview of the system’s construction, including its flow, major modules, interfaces, and data structures, offering a comprehensive understanding of how SlackR operates under the hood.

## How SlackR is put together

SlackR employs a modular design that separates concerns into distinct components, each responsible for a specific aspect of the system. By isolating key functionalities such as data retrieval, processing, and visualization, the system ensures clarity and ease of maintenance.

## Flow

A black board with white writing on it

Description automatically generated

# Major Modules

* **Home page:**
  + The first thing users see when they open SlackR is the home page. The home page allows users to input their Riot name and ID. If the player exists, then they are taken to that player’s Overview page
* **Stats.gg API:**
  + If the input information matches an existing player, then an API call to the Stats.gg API happens
  + Returned back to the system are four json files:
    - *characters.json:* Contains all character data for the input player
    - *player.json:* Contains all data for the input player
    - *recentMatches*.*json:* Contains recent match data for the input player
    - *weapons.json:* Contains weapon data for the input player
* **SlackR Overview Page:**
  + Contains all overview statistics for the input player
* **Recent Games Page**
  + Contains the links for the last five games the input user has played
  + The user has the ability to click on any of these games to pull up the that specific game’s data
  + Upon this user input, another API call is made to Stats.gg and a json file is returned to the system:
    - *game.json:* Contains all game data for a game
* **Game Page**
  + Contains all game statistics for the player, their teammates, and their opponents
  + Upon the user clicking on a teammate or opponent a API call is made to Stats.gg and a new *player.json* file is created, this time containing the player that was clicked on data
* **Teammate/Opponent page**
  + This page contains an overview of the specific player/opponent that was clicked on from the **Game Page**
* **Economy Page**
  + Contains the economy breakdown of a game

## Major interfaces

#### Home Page (Base)

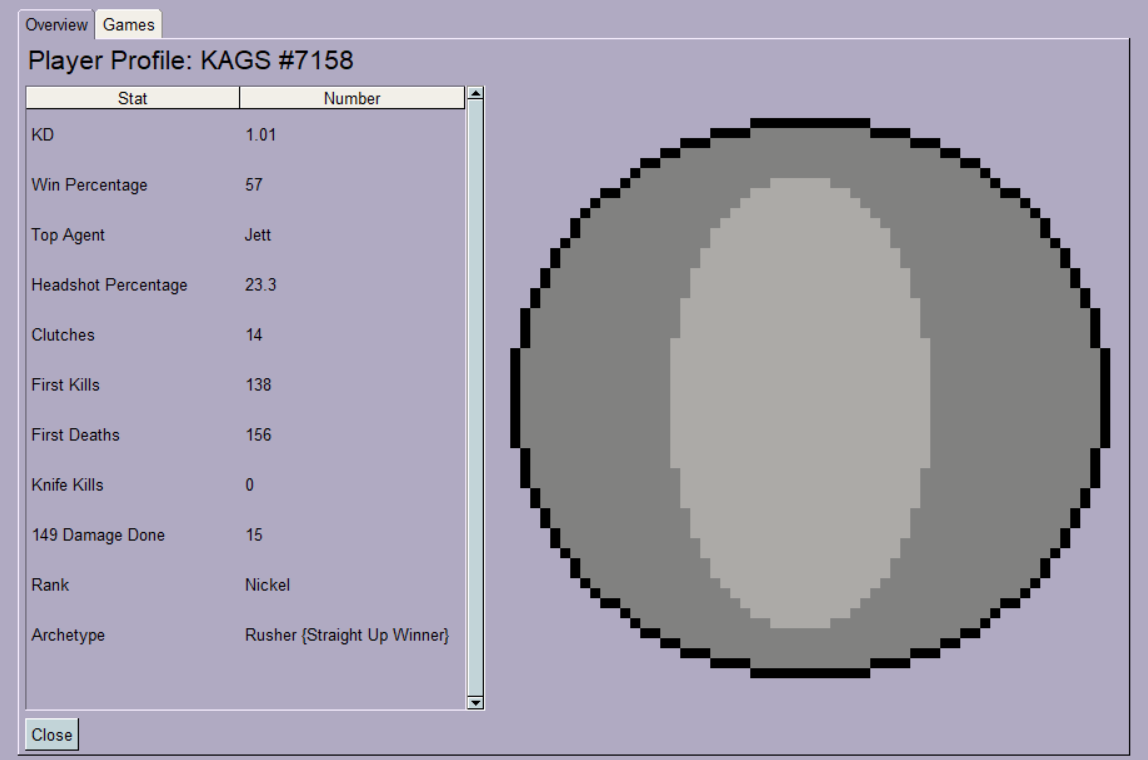
SlackR Home page. Allows for user to input a Valorant player.


#### Home Page (With Player)

A screenshot of a computer

Description automatically generated

#### Overview Page



#### Game Page

A screenshot of a computer

Description automatically generated

#### Game Page (Specific Game)

A screenshot of a computer

Description automatically generated

#### Economy Page

A screenshot of a computer

Description automatically generated

#### Teammate/Opponent Page (Private Profile)

A screenshot of a computer

Description automatically generated

## Data/Data Structures

##### player object

Created every time a new player is input into the system

* **Instance Variables**
  + *name:* Player name<string>
  + *kd:* Player KD<double>
  + *winp:* Player’s win percentage<double>
  + *top\_agent:* Player’s top agent played<string>
  + *headshot\_percentage:* Player’s headshot percentage<double>
  + *clutches:* Player’s clutches<int>
  + *first\_deaths:* Player’s first death in a round<int>
  + *first\_kills:* Player’s first kills in a round<int>
  + *one\_four\_nine\_damage\_done:* Amount of times player has hit an enemy for 149 damage (1 damage less needed to get a kill)<int>
  + *rank:* Player’s assigned rank (SlackR Rank)<string>
  + *archetype:* Player’s assigned archetype (SlackR Archetype)<string>
* **Get/Set Methods**

All the following methods both get and set the instance variables for the player object

* + *getKD*
  + *getWinPercentage*
  + *getTopAgent*
  + *getHeadshotPercentage*
  + *getClutches*
  + *getFirstDeaths*
  + *getFirstBloods*
  + *getKnifeKills*
  + *getOneFourNineDamageDone*
* **Assignment Methods**

The following methods assign both a rank and archetype to the player that was input into the SlackR system

* *calculateRank*
  + - Calculates rank based on player’s kd, win rate, clutches, first bloods, and one four nine damage done numbers
    - Each stat is assigned a weight and a base score is created
    - This base score is then compared to rank assignments and a rank is assigned to the player
  + *assignArchetype*
    - Calculates archetype based clutches, first deaths, headshot percentage, first kills, first deaths, and knife kills
    - Assigns an archetype to the player based on how high or low these stats are
* **Functional Methods**
  + *exportToExcel*
    - Gets ALL player instance variables and exports them to an excel spreadsheet
    - Every time a new player is entered a row in the spreadsheet is populated

##### Teammate object

The teammate object is created every time a new game is loaded into the system. The current player also gets a teammate object created for them.

* **Instance Variables**
  + *name:* Teammates name<string>
  + *kills:* Amount of kills teammate got in a game<int>
  + *deaths:* Amount of deaths teammate got in a game<int>
  + *assists:* Amount of assists teammate got in a game<int>
  + *hs\_perc:* Teammates headshot percentage for a game<double>
  + *team:* Teammates team in a game<list>
  + *agent:* Teammates agent played in a game<string>
  + *multi\_kills:* Amount of times teammate gets a multi kill in a game (Classified as getting 3 or more kills in a single round)<int>
* **Get/Set Methods**

The following methods both get and set the instance variables for a teammate object

* + *getKills*
  + *getDeaths*
  + *getAssists*
  + *getHSPercentage*
  + *getTeam*
  + *getAgent*
  + *getMultiKills*

##### Game object

A game object is created each time the user clicks on a specific game on the game page

* **Instance Variables**
  + *players:* All players in a game<List>
  + *red\_team:* Players on the red team<List>
  + *blue\_team:* Players on the blue team<List>
  + *map\_name:* Map name for current game<String>
* **Get/Set Methods**

The following methods both get and set the instance variables for the game object

* + *getPlayers*
  + *createRedTeam*
  + *createBlueTeam*
  + *getMapName*

##### Match object

A match object is created every time a new player is entered into the system. This object is specifically used to get the match history of a player

* **Instance Variables**
  + *names:* Map names for last 5 matches<list>
  + *current\_rank:* Player ranks for last 5 matches<list>
  + *agents\_played:* Agents played for last 5 matches<list>
  + *gameAPILink:* API links for last 5 matches<list>
    - Uses match ID to create link
* **Get/Set Methods**

The following methods both get and set the instance variables for the match object

* + *getNames*
  + *getAgents*
  + *getCurrentRank*
  + *getGameID*

##### Economy object

An economy object is created each time a game object is created. This object holds each teams economy for a specific game

* **Instance Variables**
  + *red\_team:* Players on the red team<list>
  + *blue\_team:* Players on the blue team<list>
  + *red\_economy:* Blue teams economy<list>
  + *blue\_economy:* Blue teams economy<list>
* **Get/Set Methods**

The following two methods both get and set each teams economy for a game. The economy list is round by round based. Each round 200 credits are added for each kill a team gets, the winning team of the round gets 3000 credits, whereas the losing team of a round gets 1900 credits. These lists are used within the economy page, using them to both graph each teams economy round by round. As well as find the minimum, maximum, and average economy for each team

* + *getBlueEconomy*
  + *getRedEconomy*

## Source Code

The SlackR source code is designed into three particular sections. The *main* file, containing the set up of the UI and creating all objects. The *class* file, containing all object code within the SlackR system. And the stats\_file, which contains the source code for creating and formatting API calls, creating and formatting json files, and creating and formatting API links. This source code section outlines how the code is organized, and conventions being followed within the SlackR system

### The Files

##### Main

* **Purpose:** The main file serves as the entry point for the SlackR system
* **Responsibilities:**
  + Sets up the UI
  + Instantiates and ititializes all objects required for the system to function
  + Manages interaction between the user and the backend components
* **Name:** *main.py*

##### Stats file

* **Purpose:** The stats file handles things related to API calls
* **Responsibilities:**
  + Loads specific games to get JSON file
  + Loads player profile to get JSON file
  + Load player profile to get character JSON file
  + Load player profile to get weapon JSON file
  + Load player profile to get recent matches JSON file
  + Creates the game links to load into the Strats.gg API request
  + Creates specific game link to load into the Strats.gg API request
  + Creates player links for Strats.gg API call , using the outline based on player name that was inserted into the system
  + Creates weapon link for Strats.gg API call
  + Creates character link for Strats.gg API call
  + Creates matches link for Strats.gg API call
* **Name:** *stats\_file.py*

##### Class file

* **Purpose:** Creation of all objects within the SLackR system
* **Responsibilities:**
  + Creation of Player object
  + Creation of Teammate object
  + Creation of Game object
  + Creation of Matches object
  + Creation of Economy object
* **Name:** *class\_file.py*

##### Utils file

* **Purpose:** Opens JSON files
* **Name:** *utils.py*

##### Rank configuration file

* **Purpose:** Creates ranks and archetypes for a player object
* **Responsibilities:**
  + Creates the standards for player ranks and archetypes
  + Both based off stats pulled from the Strats.gg API
* **Name:** *rank\_config.py*

## Conventions

##### Naming conventions

* **Functions:** Denoted by camelCase convention
* **Variables:** Denoted by underscore\_convention
* **Classes:** Denoted by Captial Convention

# Modifications

The **SlackR** codebase is designed for modularity, with each file handling a distinct set of responsibilities. Changes to the system should respect these modular boundaries to ensure maintainability and clarity. This section outlines how to implement software changes while adhering to the design and conventions established in the existing code. Additionally, it highlights potential improvements to the system if more time or resources were available.

## Instructions for making changes

* **Familiarize yourself with File Responsibilities**

Changes should be made within the file that aligns with the intended functionality below is a quick guide:

* + *main.py:* For UI-related updates or changes to object instantiation
  + *stats\_file.py:* For updates to API interactions, JSON formatting, or API link generation
  + *class\_file.py:* For modifications to existing objects or creation of new objects within the system
  + *utils.py:* For changes related to JSON file operations
  + *rank\_config.py:* For adjustments to player ranks and archetype standards
* **Follow Naming Conventions**
  + Functions: Use CamelCase
  + Variables: Use underscore\_convention
  + Classes: Use Capital Convention
* **Steps for Implementing Changes**
  + **Identify the Purpose of the Change:** Determine whether your change involves new functionality, a bug fix, or an enhancement to existing features
  + **Navigate to the Relevant File:** Locate the file most aligned with the change’s scope
  + **Preserve Existing Functionality:** Ensure changes do not disrupt unrelated features by testing relevant modules thoroughly
  + **Comment Extensively:** Provide inline comments explaining the purpose of the change. This is crucial for team collaboration and future revisions
  + **Test in *main.py:*** Once updates are complete, instantiate or reference the modified code in *main.py* to test integration with the UI and overall system
* **Ensure Modularity**

Any new feature or modification should remain modular. For instance:

* + Adding a new stat tracking feature? Create a new function in *stats\_file.py* to retrieve and format the required API data rather than overloading an existing function

## Potential Enhancements with More Time

* **Improved Error Handling**
  + Add robust error-checking mechanism for failed API calls in *stats\_file.py*
  + Provide more descriptive error messages to the user via *main.py*
* **Advanced Player Archetypes**
  + Expand *rank\_config.py* to incorporate more nuanced player archetypes based on additional metrics like clutch percentage or movement efficiency
* **Optimization of API Calls**
  + Implement asynchronous API calls in *stats\_file.py* to improve performance when retrieving data
* **Dynamic UI Enhancements**
  + Redesign the UI in *main.py* for better visualization of stats, including graphs and leaderboards
* **Logging and Debugging Tools**
  + Add a logging system in *utils.py* to track system activity and debug errors effectively
* **Integration of Additional APIs**
  + Incorporate stats from other platforms or data sources in *stats\_file.py* for a broader analysis of player performance

# References

## Web scraping and API integration

 Sweigart, Al. Automate the Boring Stuff with Python. No Starch Press, 2015.

 Mitchell, Ryan. Web Scraping with Python: Collecting Data from the Modern Web. O'Reilly Media, 2018.

 SeleniumHQ. Selenium Documentation. https://www.selenium.dev/documentation/

 Requests Library Documentation. https://docs.python-requests.org/

 Python JSON Library Documentation. <https://docs.python.org/3/library/json.html>

## UI Design and Modular Programming

 Fowler, Martin. Refactoring: Improving the Design of Existing Code. Addison-Wesley, 2018.

 Gamma, Erich, et al. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.

 Pilgrim, Mark. Dive Into Python 3. Apress, 2009.

 PySimpleGUI Documentation. <https://pysimplegui.readthedocs.io/>

 PEP 8 - Style Guide for Python Code. <https://peps.python.org/pep-0008/>

## Statistical Analysis and Visualization

* McKinney, Wes. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O'Reilly Media, 2017.
* VanderPlas, Jake. Python Data Science Handbook. O'Reilly Media, 2016.
* Matplotlib Documentation. https://matplotlib.org/stable/contents.html
* Waskom, Michael. Seaborn Documentation. https://seaborn.pydata.org/
* Strats.gg API. Strats.gg Documentation. <https://strats.gg/docs/>

# Appendix

**main.py**

*"""*

*main.py*

*Cayden Garcia*

*Fall 2024 - Advanced Software Engineering*

*Will call all main functions of SlackR*

*"""*

*from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg*

*import PySimpleGUI as sg*

*from stats\_file import \*  # Import all relevant functions*

*from class\_file import \*  # Import any classes used in stats*

*#test\_player = Player(player\_file,character\_file,weapon\_file)*

*def create\_input\_layout():*

*#Layout for input page(USER MUST HAVE "NAME #ID" i.e KAGS #7158)*

*layout = [*

*[sg.Text(text = "Welcome to SlackR! The slightly Comical and stress-reducing Valorant Stat Tracker!")],*

*[sg.Text(text = "Please enter your Riot Username and ID(Name,Space,ID)",background\_color=sg.theme\_background\_color()),sg.InputText()],*

*[sg.Button(button\_text = "Load Profile", button\_color=sg.theme\_background\_color()), sg.Button("Close", button\_color=sg.theme\_background\_color())]*

*]*

*return layout*

*def create\_home\_page\_layout(current\_player,recent\_matches):*

*#Data for current players overall stats*

*headers = ["Stat", "Number"]*

*data = [*

*["KD", current\_player.kd],*

*["Win Percentage", current\_player.winp],*

*["Top Agent", current\_player.top\_agent],*

*["Headshot Percentage", current\_player.headshot\_percentage],*

*["Clutches", current\_player.clutches],*

*["First Kills", current\_player.first\_kills],*

*["First Deaths", current\_player.first\_deaths],*

*["Knife Kills", current\_player.knife\_kills],*

*["149 Damage Done", current\_player.one\_four\_nine\_damage\_done],*

*["Rank", current\_player.rank],*

*["Archetype", current\_player.archetype],*

*]*

*#Data for last five games player has played*

*game\_headers = ["Rank", "Map Name", "Agent"]*

*game\_data = [*

*[recent\_matches.current\_rank[0], recent\_matches.names[0], recent\_matches.agents\_played[0]],*

*[recent\_matches.current\_rank[1], recent\_matches.names[1], recent\_matches.agents\_played[1]],*

*[recent\_matches.current\_rank[2], recent\_matches.names[2], recent\_matches.agents\_played[2]],*

*[recent\_matches.current\_rank[3], recent\_matches.names[3], recent\_matches.agents\_played[3]],*

*[recent\_matches.current\_rank[4], recent\_matches.names[4], recent\_matches.agents\_played[4]]*

*]*

*#Layout for home page*

*tab\_layout\_1 = [*

*[sg.Text(f"Player Profile: {current\_player.name}", font=("Helvetica", 16), justification="center")],*

*[sg.Table(values=data, expand\_y = True, row\_height = 40, headings=headers, auto\_size\_columns=True, justification="left", key="-TABLE-"), sg.Image(source = f"rankImages\{current\_player.rank}.png", size = (500,500))],*

*[sg.Button("Close")]*

*]*

*tab\_layout\_2 = [*

*[sg.Text(text = "Choose a game", justification="center")],*

*[sg.Table(values = game\_data, row\_height= 50, headings=game\_headers, auto\_size\_columns= True,expand\_x= True,justification="center",key="GAME\_TABLE", enable\_events=True)]*

*]*

*layout = [*

*[sg.TabGroup([[sg.Tab("Overview",tab\_layout\_1), sg.Tab("Games", tab\_layout\_2)]])]*

*]*

*return layout*

*def create\_game\_page\_layout(red\_team,blue\_team):*

*#Red team stats*

*red\_data = [*

*[red\_team[0].agent,red\_team[0].name,red\_team[0].kills,red\_team[0].deaths,red\_team[0].assists,red\_team[0].hs\_perc,red\_team[0].multi\_kills],*

*[red\_team[1].agent,red\_team[1].name,red\_team[1].kills,red\_team[1].deaths,red\_team[1].assists,red\_team[1].hs\_perc,red\_team[1].multi\_kills],*

*[red\_team[2].agent,red\_team[2].name,red\_team[2].kills,red\_team[2].deaths,red\_team[2].assists,red\_team[2].hs\_perc,red\_team[2].multi\_kills],*

*[red\_team[3].agent,red\_team[3].name,red\_team[3].kills,red\_team[3].deaths,red\_team[3].assists,red\_team[3].hs\_perc,red\_team[3].multi\_kills],*

*[red\_team[4].agent,red\_team[4].name,red\_team[4].kills,red\_team[4].deaths,red\_team[4].assists,red\_team[4].hs\_perc,red\_team[4].multi\_kills]*

*]*

*#blue team stats*

*blue\_data = [*

*[blue\_team[0].agent,blue\_team[0].name,blue\_team[0].kills,blue\_team[0].deaths,blue\_team[0].assists,blue\_team[0].hs\_perc,blue\_team[0].multi\_kills],*

*[blue\_team[1].agent,blue\_team[1].name,blue\_team[1].kills,blue\_team[1].deaths,blue\_team[1].assists,blue\_team[1].hs\_perc,blue\_team[1].multi\_kills],*

*[blue\_team[2].agent,blue\_team[2].name,blue\_team[2].kills,blue\_team[2].deaths,blue\_team[2].assists,blue\_team[2].hs\_perc,blue\_team[2].multi\_kills],*

*[blue\_team[3].agent,blue\_team[3].name,blue\_team[3].kills,blue\_team[3].deaths,blue\_team[3].assists,blue\_team[3].hs\_perc,blue\_team[3].multi\_kills],*

*[blue\_team[4].agent,blue\_team[4].name,blue\_team[4].kills,blue\_team[4].deaths,blue\_team[4].assists,blue\_team[4].hs\_perc,blue\_team[4].multi\_kills]*

*]*

*headers = ['Agent','Name','Kills', 'Deaths', 'Assists', 'HS%',"Multi-Kills"]*

*layout = [*

*[sg.Table(values = red\_data, headings=headers, num\_rows=5,row\_height=30,auto\_size\_columns=True, key='RED\_TABLE',enable\_events=True, background\_color='#f28282')],*

*[sg.Table(values=blue\_data, headings = headers,num\_rows=5,row\_height=30,auto\_size\_columns=True,key = 'BLUE\_TABLE',enable\_events=True,background\_color='#82a7f2')],*

*[sg.Button(button\_text = "Breakdowns",button\_color=sg.theme\_background\_color())]*

*]*

*return layout*

*def create\_breakdown\_layout(red\_econ,blue\_econ,map\_name):*

*headers = ['Max Economy', 'Min Economy', 'Average Economy']*

*red\_max\_econ = max(red\_econ,key=lambda x:x[1])[1]*

*red\_min\_econ = min(red\_econ,key=lambda x:x[1])[1]*

*red\_avg\_econ = round(sum(x[1] for x in red\_econ)/(len(red\_econ)))*

*blue\_max\_econ = max(blue\_econ,key=lambda x:x[1])[1]*

*blue\_min\_econ = min(blue\_econ,key=lambda x:x[1])[1]*

*blue\_avg\_econ = round(sum(x[1] for x in blue\_econ)/(len(blue\_econ)))*

*red\_econ\_table\_data = [*

*[red\_max\_econ,red\_min\_econ,red\_avg\_econ]*

*]*

*blue\_econ\_table\_data = [*

*[blue\_max\_econ,blue\_min\_econ,blue\_avg\_econ]*

*]*

*layout = [*

*[sg.Table(values = red\_econ\_table\_data, headings = headers, num\_rows = 1, row\_height = 30, auto\_size\_columns= True,background\_color='#f28282'),sg.Table(values = blue\_econ\_table\_data, headings = headers, num\_rows = 1, row\_height = 30, auto\_size\_columns= True,background\_color='#82a7f2')],*

*[sg.Graph((500,200),(0,7000),(len(red\_econ),16500),background\_color="white", key='-GRAPH-',pad=(200,10))]*

*#[sg.Graph((500,200),(0,0),(5,100),background\_color='white', key = 'RED\_ACC\_GRAPH')],*

*#[sg.Graph((500,200),(0,0),(5,100),background\_color='white', key = 'BLUE\_ACC\_GRAPH')]*

*]*

*return layout*

*def create\_secondary\_player\_layout(current\_player):*

*data = [*

*["KD", current\_player.kd],*

*["Win Percentage", current\_player.winp],*

*["Top Agent", current\_player.top\_agent],*

*["Headshot Percentage", current\_player.headshot\_percentage],*

*["Clutches", current\_player.clutches],*

*["First Kills", current\_player.first\_kills],*

*["First Deaths", current\_player.first\_deaths],*

*["Knife Kills", current\_player.knife\_kills],*

*["149 Damage Done", current\_player.one\_four\_nine\_damage\_done],*

*["Rank", current\_player.rank],*

*["Archetype", current\_player.archetype],*

*]*

*#Layout*

*headers = ["Stat", "Number"]*

*layout = [*

*[sg.Text(f"Player Profile: {current\_player.name}", font=("Helvetica", 16), justification="center")],*

*[sg.Table(values=data, expand\_y = True, row\_height = 40, headings=headers, auto\_size\_columns=True, justification="left", key="-TABLE-"), sg.Image(source = f"rankImages\{current\_player.rank}.png", size = (500,500))],*

*[sg.Button("Close")]*

*]*

*return layout*

*def main():*

*layout = create\_input\_layout()*

*theme = sg.theme("LightPurple")*

*window = sg.Window("SlackR", layout)*

*while True:*

*event,values = window.read()*

*if event == sg.WIN\_CLOSED or event =="Close":*

*break*

*if event == "Load Profile":*

*player\_link = createAPIPlayerLink(values[0])*

*weapon\_link = createAPIWeaponLink(values[0])*

*character\_link = createAPICharacterLink(values[0])*

*recent\_matches\_link = createAPIMatchesLink(values[0])*

*player\_file = loadPlayerProfile(player\_link)*

*weapon\_file = loadWeaponStats(weapon\_link)*

*character\_file = loadCharacterStats(character\_link)*

*recent\_matches\_file = loadRecentMatches(recent\_matches\_link)*

*recent\_matches = Matches(recent\_matches\_file)*

*current\_player = Player(values[0],player\_file,character\_file,weapon\_file)*

*current\_player.export\_to\_excel()*

*window.close()*

*home\_layout = create\_home\_page\_layout(current\_player,recent\_matches)*

*home\_window = sg.Window("SlackR-Home Page", home\_layout)*

*while True:*

*home\_event,values = home\_window.read()*

*if home\_event == sg.WINDOW\_CLOSED or home\_event == "Close":*

*break*

*if home\_event == "GAME\_TABLE":*

*selected = values["GAME\_TABLE"] #Returns what number row was selected in pos 0*

*if selected[0] == 0:*

*game\_link = createAPIGameLink(current\_player.name, recent\_matches.gameAPILink[0])*

*elif selected[0] == 1:*

*game\_link = createAPIGameLink(current\_player.name, recent\_matches.gameAPILink[1])*

*elif selected[0] == 2:*

*game\_link = createAPIGameLink(current\_player.name, recent\_matches.gameAPILink[2])*

*elif selected[0] == 3:*

*game\_link = createAPIGameLink(current\_player.name, recent\_matches.gameAPILink[3])*

*elif selected[0] == 4:*

*game\_link = createAPIGameLink(current\_player.name, recent\_matches.gameAPILink[4])*

*else:*

*print("Something went wrong, please try again")*

*game\_file = loadGame(game\_link)*

*game = Game(game\_file)*

*red\_team = []*

*blue\_team = []*

*for i in range(10):*

*current\_teammate = Teamate(game\_file,game.players[i])*

*if current\_teammate.team == 'Red':*

*red\_team.append(current\_teammate)*

*else:*

*blue\_team.append(current\_teammate)*

*econ = Economy(game\_file,game.red\_team,game.blue\_team)*

*map\_name = game.map\_name*

*game\_layout = create\_game\_page\_layout(red\_team,blue\_team)*

*game\_window = sg.Window("Game Page",game\_layout)*

*while True:*

*game\_event,player\_values = game\_window.read()*

*if game\_event == sg.WINDOW\_CLOSED:*

*break*

*if game\_event == "Breakdowns":*

*breakdown\_layout = create\_breakdown\_layout(econ.red\_economy,econ.blue\_economy,map\_name)*

*breakdown\_window = sg.Window("Breakdown Page",layout = breakdown\_layout)*

*breakdown\_window.finalize()*

*graph = breakdown\_window['-GRAPH-']*

*graph.draw\_text(text = f"Econ(round based)\n",location = (3,7500))*

*graph.draw\_lines(points = econ.red\_economy,color = 'red', width = 3)*

*for i in econ.red\_economy:*

*graph.draw\_point(point = i, size = .3, color='red')*

*graph.draw\_lines(points = econ.blue\_economy, color = 'blue', width = 3)*

*for i in econ.blue\_economy:*

*graph.draw\_point(point = i, size = .3, color = 'blue')*

*while True:*

*event,values = breakdown\_window.read()*

*if event == sg.WINDOW\_CLOSED:*

*break*

*else:*

*if game\_event == "RED\_TABLE":*

*player\_selected = player\_values['RED\_TABLE']*

*if player\_selected[0] == 0:*

*new\_player = red\_team[0].name*

*elif player\_selected[0] == 1:*

*new\_player = red\_team[1].name*

*elif player\_selected[0] == 2:*

*new\_player = red\_team[2].name*

*elif player\_selected[0] == 3:*

*new\_player = red\_team[3].name*

*elif player\_selected[0] == 4:*

*new\_player = red\_team[4].name*

*elif game\_event == "BLUE\_TABLE":*

*player\_selected = player\_values['BLUE\_TABLE']*

*if player\_selected[0] == 0:*

*new\_player = blue\_team[0].name*

*elif player\_selected[0] == 1:*

*new\_player = blue\_team[1].name*

*elif player\_selected[0] == 2:*

*new\_player = blue\_team[2].name*

*elif player\_selected[0] == 3:*

*new\_player = blue\_team[3].name*

*elif player\_selected[0] == 4:*

*new\_player = blue\_team[4].name*

*new\_player = new\_player.replace("#"," #")*

*new\_player\_player\_link = createAPIPlayerLink(new\_player)*

*new\_player\_weapon\_link = createAPIWeaponLink(new\_player)*

*new\_player\_character\_link = createAPICharacterLink(new\_player)*

*new\_player\_player\_file = loadPlayerProfile(new\_player\_player\_link)*

*new\_player\_weapon\_file = loadWeaponStats(new\_player\_weapon\_link)*

*new\_player\_character\_file = loadCharacterStats(new\_player\_character\_link)*

*try:*

*secondary\_player = Player(new\_player,new\_player\_player\_file,new\_player\_character\_file,new\_player\_weapon\_file)*

*secondary\_player.export\_to\_excel()*

*secondary\_home\_layout = create\_secondary\_player\_layout(secondary\_player)*

*secondary\_home\_window = sg.Window(f"{secondary\_player.name} Overiview",secondary\_home\_layout)*

*while True:*

*secondary\_home\_event,\_ = secondary\_home\_window.read()*

*if secondary\_home\_event == sg.WINDOW\_CLOSED:*

*break*

*except Exception as e:*

*print(e)*

*sg.popup("Profile is private...")*

*game\_window.close()*

*home\_window.close()*

*break*

*if \_\_name\_\_ == "\_\_main\_\_":*

*main()*

**Stats File**

*import requests*

*import json*

*import itertools*

*import matplotlib.pyplot as plt*

*import numpy as np*

*from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg*

*import statistics*

*#from sklearn.preprocessing import MinMaxScaler*

*from collections import Counter*

*from class\_file import Player*

*from utils import openJsonFile*

*#from class\_file import Player*

*"""Functionality functions"""*

*def loadDriver(url): #Loads driver*

*"""Selenium"""*

*options = Options() #intialize option variable to class Options*

*options.add\_argument('--headless') #Take away window from being open on program run*

*options.add\_argument('--ignore-certificate-errors')*

*options.add\_argument('--allow-insecure-localhost')*

*driver = webdriver.Chrome(service = Service(ChromeDriverManager().install()), options = options) #Initializes driver var to chrome driver*

*driver.get(url) #URL for what html page I want*

*driver.implicitly\_wait(10) #Makes driver wait 10 ms before doing anything: Allows for everything to load before accessing HTML elements*

*print(f"Driver loaded: {driver} with link: {url}")*

*return driver*

*def loadGame(url): #Loads specific game to get JSON file from game*

*payload = ""*

*headers = {"User-Agent": "insomnia/10.0.0"}*

*response = requests.get(url,headers=headers)*

*game\_data = response.json()*

*game\_file = 'game.json'*

*with open(game\_file,'w') as json\_file:*

*json.dump(game\_data,json\_file, indent=4)*

*print(f"Data has been saved to {game\_file}")*

*return game\_file*

*def loadPlayerProfile(url):*

*querystring = {"playlist":"competitive","season\_id":"dcde7346-4085-de4f-c463-2489ed47983b"}*

*payload = ""*

*headers = {"User-Agent": "insomnia/10.0.0"}*

*response = requests.get(url, data=payload, headers=headers, params=querystring)*

*player\_data = response.json()*

*player\_file = 'player.json'*

*with open(player\_file,'w') as json\_file:*

*json.dump(player\_data,json\_file,indent=4)*

*print(f"Player data has been saved to {player\_file}")*

*return player\_file*

*def loadCharacterStats(url):*

*querystring = {"playlist":"competitive","season\_id":"dcde7346-4085-de4f-c463-2489ed47983b"}*

*payload = ""*

*headers = {"User-Agent": "insomnia/10.0.0"}*

*response = requests.get(url, data=payload, headers=headers, params=querystring)*

*character\_data = response.json()*

*character\_file = 'characters.json'*

*with open(character\_file,'w') as json\_file:*

*json.dump(character\_data,json\_file,indent=4)*

*print(f"Character data has been saved to {character\_file}")*

*return character\_file*

*def loadWeaponStats(url):*

*querystring = {"playlist":"competitive","season\_id":"dcde7346-4085-de4f-c463-2489ed47983b"}*

*payload = ""*

*headers = {"User-Agent": "insomnia/10.0.0"}*

*response = requests.get(url, data=payload, headers=headers, params=querystring)*

*character\_data = response.json()*

*weapons\_file = 'weapons.json'*

*with open(weapons\_file,'w') as json\_file:*

*json.dump(character\_data,json\_file,indent=4)*

*print(f"Weapon data has been saved to {weapons\_file}")*

*return weapons\_file*

*def loadRecentMatches(url):*

*querystring = {"":["",""],"playlist":"competitive","season\_id":"dcde7346-4085-de4f-c463-2489ed47983b","include\_queued":"true"}*

*payload = ""*

*headers = {"User-Agent": "insomnia/10.0.0"}*

*response = requests.get(url, data=payload, headers=headers, params=querystring)*

*character\_data = response.json()*

*recent\_matches\_file = 'recentMatches.json'*

*with open(recent\_matches\_file,'w') as json\_file:*

*json.dump(character\_data,json\_file,indent=4)*

*print(f"Recent matches data has been saved to {recent\_matches\_file}")*

*return recent\_matches\_file*

*def create\_players(game): #Creates Player class and adds the player name and team to the class*

*teams = assignTeam(game)*

*players = []*

*for team,names in teams.items():*

*for name in names:*

*player = Player(name,team)*

*players.append(player)*

*return players*

*def getGameLinksForStratsGG(link): #Gets LAST FIVE GAMES PLAYED links, uses strats.gg overview page*

*strats = loadDriver(link) #Load driver*

*game\_list = []*

*game\_links = strats.find\_elements(By.CSS\_SELECTOR, "a.match") #a.match: match is the CSS selector, a is the anchor tag*

*all\_links = [link.get\_attribute('href') for link in game\_links] #Just get the links*

*for href in all\_links:*

*game\_list.append(href) #Add links to list*

*strats.quit()*

*return game\_list[:4] #Return last give games played*

*def createAPIGameLink(player\_name, game\_id):*

*num\_of\_spaces = player\_name.count(" ")*

*if num\_of\_spaces == 2:*

*split = player\_name.split("#")*

*name = split[0]*

*id = split[1]*

*split\_name = name.split(" ")*

*name1 = split\_name[0]*

*name2 = split\_name[1]*

*url = f"https://api.strats.gg/internal/api/v1/games/valorant/accounts/riot/{name1}%20{name2}%23{id}/matches/{game\_id}"*

*print(url)*

*return url*

*else:*

*split = player\_name.split("#")*

*name = split[0]*

*name = name.replace(" ","")*

*id = split[1]*

*url = f"https://api.strats.gg/internal/api/v1/games/valorant/accounts/riot/{name}%23{id}/matches/{game\_id}"*

*print(url)*

*return url*

*"""Overview stats functions"""*

*def getOverallStats(link):  # Gets overall stats and adds them to one tuple*

*driver = loadDriver(link)*

*try:*

*# Wait up to 9 seconds for each element to load; move on if not found*

*kd = getKD(driver) or "N/A"  # Return "N/A" if KD is not found*

*winp = getWinPercentage(driver) or "N/A"*

*top\_agent = getTopAgent(driver) or "N/A"*

*headshot\_percentage = getHeadShotPercentage(driver) or "N/A"*

*except TimeoutException:*

*print("Some elements took too long to load, moving on with available data.")*

*# Handle the case where some of the elements take too long to load or aren't found.*

*kd, winp, top\_agent, headshot\_percentage = "N/A", "N/A", "N/A", "N/A"*

*finally:*

*driver.quit()*

*return kd, winp, top\_agent, headshot\_percentage*

*def getKD(player): #Gets kd for a player, uses API call to get json player file for player*

*data = openJsonFile(player)*

*stats = data['stats']*

*kd = stats['kd\_ratio']*

*#print(kd)*

*return kd*

*def getWinPercentage(player): #Gets win percentage for a player, uses API call to get json player file for player*

*data = openJsonFile(player)*

*try:*

*stats = data['stats']*

*wins = stats['matches\_won']*

*played = stats['matches\_played']*

*winp = round((wins/played) \* 100)*

*except:*

*winp = 50*

*#print(winp)*

*return winp*

*def getTopAgent(player): #Gets top agent for a player, uses API call to get json character file for player*

*data = openJsonFile(player)*

*agent\_list = data['characters']*

*top\_agent = agent\_list[0]*

*top\_agent\_info = top\_agent['character']*

*top\_agent\_name = top\_agent\_info['name']*

*print(top\_agent\_name)*

*return top\_agent\_name*

*def getHeadShotPercentage(player): #Gets headshot percentage for a player, uses API call to get json player file for player*

*data = openJsonFile(player)*

*stats = data['stats']*

*hs\_perc = stats['headshots\_percent']*

*print(hs\_perc)*

*return hs\_perc*

*def getKnifeKills(player): #Gets knife kills for a player, uses API call to get json weapon file for player*

*data = openJsonFile(player)*

*knife\_kills = None*

*weapons\_list = data['weapons']*

*for weapon in weapons\_list: #Iterates through each weapon*

*name = weapon['metadata']['name']*

*kills = weapon['stats']['kills']*

*if "Melee" in name: #Limits to only knife kills*

*knife\_kills = kills*

*if knife\_kills == None:*

*knife\_kills = 0*

*#print(knife\_kills)*

*return knife\_kills*

*def get149DamageDone(url): #Calculate the amount of times 149 damage is done*

*driver = loadDriver(url) #load driver*

*word = "Phantom" #Looks for phantom*

*get\_table = driver.find\_elements(By.CSS\_SELECTOR,'tr') #Gets table*

*for row in get\_table: #Get individual rows*

*row = row.text*

*if word in row: #Only store row with phantom in it*

*phantom\_stats = row.split('\n') #Get rid of new line characters*

*hits\_with\_a\_kill = float(phantom\_stats[3]) #Get kill conversion, change it float*

*head\_shot\_percentage = float(phantom\_stats[6].replace('%', '')) #Get rid of % sign in hs%, change it to float*

*head\_shot\_percentage = head\_shot\_percentage/100 #Change hs% into decimal value*

*#Will need to simplify and correct these naming conventions later*

*total\_hits = phantom\_stats[7:] #Limit to just hits stats*

*total\_hits = [item.split('(')[1].split(')')[0] for item in total\_hits] #Splits at both parentheses and grabs just the number of hits*

*float\_hits = [float(item) for item in total\_hits] #Floatifiies number of hits*

*sum\_of\_hits = sum(float\_hits) #Sums all hit values*

*#Calculating 149 Damage Done*

*hits\_without\_a\_kill = 1.0 - hits\_with\_a\_kill #Get hits without a kill*

*head\_shots\_without\_a\_kill = head\_shot\_percentage \* hits\_without\_a\_kill #Compute head shots withotu a kill*

*one\_four\_nine\_damage\_done = int(sum\_of\_hits \* head\_shots\_without\_a\_kill) #Get amount of times 149 damage is done*

*driver.quit() #Quit driver*

*return one\_four\_nine\_damage\_done*

*"""Single game stats"""*

*def getStatsForOneGame(link): #Gets stats for one SINGLE GAME, uses strats.gg game page*

*game = loadDriver(link) #Load driver with strats.gg link*

*stat\_list = []*

*stats = game.find\_elements(By.CLASS\_NAME, 'compare-table\_\_row') #Find stats*

*for stat in stats:*

*stat = stat.text #Get text*

*stat = '\n'.join(stat.split('\n')[:2]) #Split at \n, only take first two positions, add them back together*

*stat = stat.replace("\n"," ") #Replace new line character with space*

*stat\_list.append(stat) #Add stat to list*

*game.quit() #close driver*

*return stat\_list #Return game stats*

*def findEconomyAverage(driver):*

*all\_players\_economy = []*

*while True:*

*try:*

*economy\_column = driver.find\_elements(By.CLASS\_NAME, 'col-economy')*

*for econ in economy\_column:*

*if econ.text == "Econ":*

*continue*

*all\_players\_economy.append(econ.text)*

*break  # Break if everything is fine*

*except StaleElementReferenceException:*

*continue  # Retry fetching elements if stale*

*player\_team\_econ = all\_players\_economy[0:5]*

*enemy\_team\_econ = all\_players\_economy[5:]*

*int\_player\_econ = [int(item) for item in player\_team\_econ]*

*avg\_player\_econ = sum(int\_player\_econ) / len(player\_team\_econ)*

*int\_enemy\_econ = [int(item) for item in enemy\_team\_econ]*

*avg\_enemy\_econ = sum(int\_enemy\_econ) / len(enemy\_team\_econ)*

*return avg\_player\_econ, avg\_enemy\_econ*

*def findWinOrLoss(driver):*

*#driver = loadDriver(url)*

*win\_or\_loss = []*

*find\_win\_or\_loss = driver.find\_elements(By.CSS\_SELECTOR,'span.title.type-subtitle--bold')*

*for game in find\_win\_or\_loss:*

*win\_or\_loss.append(game.text)*

*last\_5\_games = win\_or\_loss[:5]*

*return last\_5\_games*

*def getGameLinksForBlitzGG(driver):*

*#driver = loadDriver(url)*

*game\_list = []*

*game\_links = driver.find\_elements(By.CSS\_SELECTOR,'a.match-link')*

*all\_links = [link.get\_attribute('href') for link in game\_links]*

*for href in all\_links:*

*game\_list.append(href)*

*last\_five\_game\_links = game\_list[:5]*

*return last\_five\_game\_links*

*def calculateAntiThrifties(url):*

*driver = loadDriver(url)*

*game\_links = getGameLinksForBlitzGG(driver)*

*game\_outcomes = findWinOrLoss(driver)*

*results = {}*

*anti\_thrifted = 0*

*thrifties = 0*

*for index, link in enumerate(game\_links):*

*game = loadDriver(link)*

*econ\_averages = findEconomyAverage(game)*

*#Add results to dictionary with index as key*

*results[f"Game {index + 1}"] = {*

*"Outcome": game\_outcomes[index],*

*"Average Player Econ": econ\_averages[0],*

*"Average Enemy Econ": econ\_averages[1]*

*}*

*for game,data in results.items():*

*if data["Outcome"] == 'Defeat' and data["Average Player Econ"] > data["Average Enemy Econ"]:*

*anti\_thrifted += 1*

*elif data["Outcome"] == 'Victory' and data["Average Player Econ"] < data["Average Enemy Econ"]:*

*thrifties += 1*

*else:*

*print("No links between econ and outcome of game")*

*print(anti\_thrifted)*

*print(thrifties)*

*driver.quit()*

*"""Following functions work all together to create round by roud Win% Algo"""*

*def calculateMoneyPercentage(money\_list):*

*value\_count = {}  # List to store the occurrences of each value*

*flat\_money\_list = list(itertools.chain(\*money\_list))  # Combine the list of lists into one big list*

*flat\_money\_list.sort()  # Sort in ascending order*

*value\_count = Counter(flat\_money\_list)  # Get the count for each value*

*total\_values = len(flat\_money\_list)  # Get the length of the list*

*value\_percentage\_list = []  # List to store both the value and the percentage as a tuple*

*for value, count in value\_count.items():  # For each value calculate the percentage and add it to the new list*

*percentage = (count / total\_values) \* 100*

*value\_percentage\_list.append((value, percentage))  # Each tuple will contain the value in [0] and percentage [1]*

*# Extract percentages for normalization*

*percentages = np.array([perc for \_, perc in value\_percentage\_list]).reshape(-1, 1)  # Reshape for scaler*

*# Initialize the MinMaxScaler*

*#scaler = MinMaxScaler()*

*# Fit and transform the percentages*

*#normalized\_percentages = scaler.fit\_transform(percentages)*

*# Combine the normalized percentages with the original values*

*#normalized\_value\_percentage\_list = [(value, norm\_perc[0]) for (value, \_), norm\_perc in zip(value\_percentage\_list, normalized\_percentages)]*

*#return normalized\_value\_percentage\_list  # Returns list of tuples with normalized percentages*

*def calculatePlayerRoundWinPercentage(player, money\_list, round\_money, round\_outcomes):*

*name = player.name*

*team = player.team*

*money\_percentage\_list = []*

*round\_percentage\_list = []*

*total\_percentage\_list = []*

*# Assuming money\_list contains tuples of (money, percentage)*

*money\_percs = calculateMoneyPercentage(money\_list)*

*# Calculate the median from the money values*

*money\_values = [money for money, perc in money\_percs]*

*median\_value = statistics.median(money\_values)  # Properly calculate median*

*""" Now, bring in the normalized percentages and adjust based on median\_value """*

*for value in round\_money:*

*for num, perc in money\_percs:*

*if value == num:*

*# Adjust percentage based on comparison with the median*

*if value < median\_value:*

*perc = perc \* -.5*

*elif value > median\_value:*

*perc = perc \* 1.5*

*else:*

*perc = 0  # If equal to the median, make it neutral*

*money\_percentage\_list.append(float(perc) / 10)*

*# Round percentage based on whether player's team won or lost the round*

*for round\_num, winning\_team in round\_outcomes.items():*

*if team == winning\_team:*

*round\_perc = .01*

*else:*

*round\_perc = -.01*

*round\_percentage\_list.append(round\_perc)*

*# Combine both percentage lists*

*total\_percentage\_list = [a + b for a, b in zip(money\_percentage\_list, round\_percentage\_list)]*

*return total\_percentage\_list*

*def getPlayersInGame(game): #Gets the names of players in a game, intakes a JSON game file*

*game = openJsonFile(game)*

*player\_list = []*

*blue\_team = []*

*red\_team = []*

*players = game['match']['players']*

*for player in players:*

*team = player['metadata']['team\_id']*

*platform\_info = player['platform\_info']*

*name = platform\_info['platform\_user\_nick']*

*if team == "Blue":*

*blue\_team.append(name)*

*else:*

*red\_team.append(name)*

*player\_list = blue\_team + red\_team*

*return player\_list*

*def getAvgTeamWinPercentage(players): #Takes in a list of players from a game, gets average winp for each team*

*#Lists for teams*

*all\_players = []*

*for player in players: #goes through players, creates link, loads that link to get JSON file, uses JSON file to get win percentage, adds winp to list*

*link = createAPIPlayerLink(player)*

*player\_data = loadPlayerProfile(link)*

*winp = getWinPercentage(player\_data)*

*winp = float(winp)*

*all\_players.append(winp)*

*#Specify which team is which*

*blue\_team = all\_players[:5]*

*red\_team = all\_players[5:]*

*#Creates the average winp for each team*

*red\_team\_winp = round(sum(red\_team)/len(red\_team))*

*blue\_team\_winp = round(sum(blue\_team)/len(blue\_team))*

*#print(team\_1\_winp)*

*#print(team\_2\_winp)*

*return blue\_team\_winp,red\_team\_winp*

*"""The following three functions have been moved to the player class"""*

*def findRoundOutcome(game): #Finds what team won each round for a game, returns dict*

*with open(game,'r') as json\_file:*

*data = json.load(json\_file)*

*round\_outcome = {}*

*rounds = data['match']['rounds'] #limit to rounds sub-cat for match*

*for round\_data in rounds: #gets round num and the team that won for each round, adds to dictionary*

*round\_num = round\_data['round\_num']*

*winning\_team = round\_data['winning\_team']*

*round\_outcome[round\_num] = winning\_team*

*#print(round\_outcome)*

*return round\_outcome*

*def assignTeam(game):*

*with open(game, 'r') as json\_file:  # Load game file*

*data = json.load(json\_file)*

*teams\_dict = {"Red": [], "Blue": []}  # Initialize dictionary with two keys: Red and Blue*

*players = data['match']['players']  # Limit to match -> players*

*for player in players:  # For each player, get name and team\_id*

*platform\_info = player['platform\_info']*

*name = platform\_info['platform\_user\_nick']*

*team\_name = player['metadata']['team\_id']  # match -> players -> metadata: team*

*if "Red" in team\_name:*

*teams\_dict["Red"].append(name)*

*elif "Blue" in team\_name:*

*teams\_dict["Blue"].append(name)*

*return teams\_dict  # Return the dictionary with the teams*

*def getKillsPerRound(game): #Gets kills for each player for each round, using strats gg API and json file, returns dict*

*with open(game,'r') as json\_file:*

*data = json.load(json\_file) #Load json\_file for game*

*round\_dict = {} #Create dictionary*

*players = data['match']['players'] #Limit to match -> Players*

*for player in players: #For each player*

*platform\_info = player['platform\_info'] #limit to match -> Players -> platform\_info*

*name = platform\_info['platform\_user\_nick'] #gets name of player match -> Players -> platform\_info: name*

*if name not in round\_dict:*

*round\_dict[name] = {} #if the name is not in the dicitonary, put it ther*

*rounds = player['round\_results'] #Limit to round results, match -> players -> round\_results*

*for round\_data in rounds: #iterate through each round to get kills and round number*

*round\_num = round\_data['round\_num'] #match -> players -> round\_results: round\_num*

*kills = round\_data['kills'] #match -> players -> round\_results: kills*

*round\_dict[name][round\_num] = kills #Store into dictionary with name as the key*

*#print(round\_dict)*

*return round\_dict*

*"""Python Functionality Functions"""*

*def createPlayerLink(x): #Gives a strats.gg outline based on the player name that was inserted*

*num\_of\_spaces = x.count(" ")*

*if num\_of\_spaces == 1:*

*split = x.split(" ")*

*p\_name = split[0]*

*p\_id = split[1]*

*link = f"https://www.strats.gg/valorant/stats/{p\_name}%23{p\_id}/overview"*

*print(f"Link created {link}")*

*return link*

*else:*

*split = x.split(" ")*

*p\_name1 = split[0]*

*p\_name2 = split[1]*

*p\_id = split[2]*

*link = f"https://www.strats.gg/valorant/stats/{p\_name1}%20{p\_name2}%23{p\_id}/overview"*

*print(f"Link created {link}")*

*return link*

*def createAPIPlayerLink(player): #Takes in player name,Creates outline for API link, returns link created*

*num\_of\_spaces = player.count(" ")*

*if num\_of\_spaces == 2:*

*split = player.split("#")*

*name = split[0]*

*id = split[1]*

*split\_name = name.split(" ")*

*name1 = split\_name[0]*

*name2 = split\_name[1]*

*url = f"https://api.strats.gg/internal/api/v1/games/valorant/accounts/riot/{name1}%20{name2}%23{id}/sections/season"*

*print("Player Profile API link Created!")*

*return url*

*else:*

*split = player.split('#')*

*name = split[0]*

*name = name.replace(" ","")*

*id = split[1]*

*url = f"https://api.strats.gg/internal/api/v1/games/valorant/accounts/riot/{name}%23{id}/sections/season"*

*print("Player Profile API link Created!")*

*return url*

*def createAPIWeaponLink(player): #Takes in player name,Creates outline for API link, returns link created*

*num\_of\_spaces = player.count(" ")*

*if num\_of\_spaces == 2:*

*split = player.split("#")*

*name = split[0]*

*id = split[1]*

*split\_name = name.split(" ")*

*name1 = split\_name[0]*

*name2 = split\_name[1]*

*url = f"https://api.strats.gg/internal/api/v1/games/valorant/accounts/riot/{name1}%20{name2}%23{id}/sections/weapons"*

*return url*

*else:*

*split = player.split('#')*

*name = split[0]*

*name = name.replace(" ","")*

*id = split[1]*

*url = f"https://api.strats.gg/internal/api/v1/games/valorant/accounts/riot/{name}%23{id}/sections/weapons"*

*return url*

*def createAPICharacterLink(player): #Takes in player name,Creates outline for API link, returns link created*

*num\_of\_spaces = player.count(" ")*

*if num\_of\_spaces == 2:*

*split = player.split("#")*

*name = split[0]*

*id = split[1]*

*split\_name = name.split(" ")*

*name1 = split\_name[0]*

*name2 = split\_name[1]*

*url = f"https://api.strats.gg/internal/api/v1/games/valorant/accounts/riot/{name1}%20{name2}%23{id}/sections/characters"*

*return url*

*else:*

*split = player.split('#')*

*name = split[0]*

*name = name.replace(" ", "")*

*id = split[1]*

*url = f"https://api.strats.gg/internal/api/v1/games/valorant/accounts/riot/{name}%23{id}/sections/characters"*

*return url*

*def createAPIMatchesLink(player): #Takes in player name,Creates outline for API link, returns link created*

*num\_of\_spaces = player.count(" ")*

*if num\_of\_spaces == 2:*

*split = player.split("#")*

*name = split[0]*

*id = split[1]*

*split\_name = name.split(" ")*

*name1 = split\_name[0]*

*name2 = split\_name[1]*

*url = f"https://api.strats.gg/internal/api/v1/games/valorant/accounts/riot/{name1}%20{name2}%23{id}/matches"*

*return url*

*else:*

*split = player.split('#')*

*name = split[0]*

*name = name.replace(" ", "")*

*id = split[1]*

*url = f"https://api.strats.gg/internal/api/v1/games/valorant/accounts/riot/{name}%23{id}/matches"*

*return url*

**Class File**

*from stats\_file import \**

*from utils import openJsonFile*

*import pandas as pd*

*from rank\_config import ARCHETYPES, RANKS #For rank and archetype assignment*

*import os*

*class Player: #For current player that is looking at his/her stats*

*def \_\_init\_\_(self,name,player\_file,character\_file,weapons\_file):*

*self.name = name*

*self.kd = self.getKD(player\_file)*

*self.winp = self.getWinPercentage(player\_file)*

*self.top\_agent = self.getTopAgent(character\_file)*

*self.headshot\_percentage = self.getHeadshotPercentage(player\_file)*

*self.clutches = self.getClutches(player\_file)*

*self.first\_deaths = self.getFirstDeaths(player\_file)*

*self.first\_kills = self.getFirstBloods(player\_file)*

*self.knife\_kills = self.getKnifeKills(weapons\_file)*

*self.one\_four\_nine\_damage\_done = self.getOneFourNineDamageDone(weapons\_file)*

*self.rank = self.calculateRank()*

*self.archetype = self.assignArchetype()*

*def getKD(self,player\_file):*

*data = openJsonFile(player\_file)*

*stats = data['stats']*

*return stats['kd\_ratio']*

*def getWinPercentage(self,player\_file):*

*try:*

*data = openJsonFile(player\_file)*

*stats = data['stats']*

*wins = stats['matches\_won']*

*played = stats['matches\_played']*

*return round((wins/played) \* 100)*

*except:*

*return 50*

*def getTopAgent(self,character\_file):*

*data = openJsonFile(character\_file)*

*agent\_list = data['characters']*

*top\_agent = agent\_list[0]*

*top\_agent\_info = top\_agent['character']*

*return top\_agent\_info['name']*

*def getHeadshotPercentage(self,player\_file):*

*data = openJsonFile(player\_file)*

*stats = data['stats']*

*return stats['headshots\_percent']*

*def getClutches(self,player\_file):*

*data = openJsonFile(player\_file)*

*stats = data['stats']*

*return stats['clutches']*

*def getFirstDeaths(self,player\_file):*

*data = openJsonFile(player\_file)*

*stats = data['stats']*

*return stats['first\_deaths']*

*def getFirstBloods(self,player\_file):*

*data = openJsonFile(player\_file)*

*stats = data['stats']*

*return stats['first\_bloods']*

*def getKnifeKills(self,weapon\_file):*

*data = openJsonFile(weapon\_file)*

*weapons\_list = data['weapons']*

*for i,weapon in enumerate(weapons\_list):*

*metadata = weapon['metadata']*

*weapon\_name = metadata['name']*

*try:*

*if "Melee" in weapon\_name:*

*melee\_data = weapons\_list[i]*

*stats = melee\_data['stats']*

*return stats['kills']*

*except:*

*print("No knife kills :(")*

*return 0*

*return 0*

*def getOneFourNineDamageDone(self,weapon\_file):*

*data = openJsonFile(weapon\_file)*

*weapons\_list = data['weapons']*

*for i,weapon in enumerate(weapons\_list):*

*metadata = weapon['metadata']*

*weapon\_name = metadata['name']*

*try:*

*if "Phantom" in weapon\_name:*

*phantom\_data = weapons\_list[i]*

*stats = phantom\_data['stats']*

*hits\_with\_a\_kill = stats['kill\_conversion']*

*accuracy = stats['accuracy']*

*headshot\_percent = accuracy['headshots\_percent']*

*head\_shots = accuracy['headshots']*

*body\_shots = accuracy['bodyshots']*

*leg\_shots = accuracy['legshots']*

*total\_hits = head\_shots + body\_shots + leg\_shots*

*hits\_without\_a\_kill = 1.0 - hits\_with\_a\_kill*

*head\_shots\_without\_a\_kill = (headshot\_percent/100) \* hits\_without\_a\_kill*

*return round(total\_hits \* head\_shots\_without\_a\_kill)*

*except:*

*print("No Phantom Kills :(")*

*return 0*

*def calculateRank(self):*

*#Score Assignments*

*kd\_score = .3*

*win\_rate\_score = .2*

*clutch\_score = .2*

*first\_blood\_score = .1*

*knife\_kills = .1*

*one\_four\_nine\_score = .1*

*#Score Calculations*

*base\_score = (*

*(self.kd \* kd\_score) + (self.winp \* win\_rate\_score) + (self.clutches \* clutch\_score) +*

*(self.first\_kills \* first\_blood\_score) + (self.one\_four\_nine\_damage\_done \* one\_four\_nine\_score)*

*)/100*

*for rank, threshold in RANKS.items():*

*if base\_score >= threshold:*

*return rank*

*return "Nickel"*

*def assignArchetype(self):*

*stats = {*

*'Clutches' : self.clutches,*

*'first\_deaths' : self.first\_deaths,*

*'hs\_perc' : self.headshot\_percentage,*

*'first\_kills' : self.first\_kills,*

*'knife\_kills' : self.knife\_kills,*

*'winp' : self.winp*

*}*

*archetype\_list = []*

*for archetype, condition in ARCHETYPES.items():*

*if condition(stats):*

*archetype\_list.append(archetype)*

*if not archetype\_list:*

*archetype\_list.append("Basic")*

*return archetype\_list*

*def export\_to\_excel(self, file\_name="player\_stats.xlsx"):*

*player\_data = {*

*"Name": self.name,*

*"KD": self.kd,*

*"Win percentage": self.winp,*

*"Top Agent": self.top\_agent,*

*"Headshot Percentage": self.headshot\_percentage,*

*"Clutches": self.clutches,*

*"First Kills": self.first\_kills,*

*"First Deaths": self.first\_deaths,*

*"Knife Kills": self.knife\_kills,*

*"149 Damage Done": self.one\_four\_nine\_damage\_done,*

*"Rank": self.rank,*

*"Archetype": self.archetype,*

*}*

*# Create dataframe with a single row*

*df = pd.DataFrame([player\_data])*

*if os.path.isfile(file\_name):*

*# If the file exists, append data to it*

*with pd.ExcelWriter(file\_name, engine="openpyxl", mode="a", if\_sheet\_exists="overlay") as writer:*

*df.to\_excel(writer, index=False, header=False, startrow=writer.sheets["Sheet1"].max\_row)*

*else:*

*# If the file does not exist, create it with headers*

*df.to\_excel(file\_name, index=False, engine="openpyxl")*

*print(f"{self.name}'s stats successfully exported to Excel sheet '{file\_name}'.")*

*class Teamate: #Object for current players team mates, changes each a new game is loaded. Current player also gets a object for this*

*def \_\_init\_\_(self,game\_file,player\_name):*

*self.name = player\_name*

*self.kills = self.getKills(game\_file,player\_name)*

*self.deaths = self.getDeaths(game\_file,player\_name)*

*self.assists = self.getAssists(game\_file,player\_name)*

*self.hs\_perc = self.getHSPercentage(game\_file,player\_name)*

*self.team = self.getTeam(game\_file,player\_name)*

*self.agent = self.getAgent(game\_file,player\_name)*

*self.multi\_kills = self.getMultiKills(game\_file,player\_name)*

*def getKills(self,game\_file,current\_player\_name):*

*data = openJsonFile(game\_file)*

*match = data['match']*

*players = match['players']*

*for player in players:*

*platform\_info = player['platform\_info']*

*player\_name = platform\_info['platform\_user\_nick']*

*if player\_name in current\_player\_name:*

*stats = player['stats']*

*kills = stats['kills']*

*return kills*

*else:*

*continue*

*def getDeaths(self,game\_file,current\_player\_name):*

*data = openJsonFile(game\_file)*

*match = data['match']*

*players = match['players']*

*for player in players:*

*platform\_info = player['platform\_info']*

*player\_name = platform\_info['platform\_user\_nick']*

*if player\_name in current\_player\_name:*

*stats = player['stats']*

*deaths = stats['deaths']*

*return deaths*

*else:*

*continue*

*def getAssists(self,game\_file,current\_player\_name):*

*data = openJsonFile(game\_file)*

*match = data['match']*

*players = match['players']*

*for player in players:*

*platform\_info = player['platform\_info']*

*player\_name = platform\_info['platform\_user\_nick']*

*if player\_name in current\_player\_name:*

*stats = player['stats']*

*assists = stats['assists']*

*return assists*

*else:*

*continue*

*def getHSPercentage(self,game\_file,current\_player\_name):*

*data = openJsonFile(game\_file)*

*match = data['match']*

*players = match['players']*

*for player in players:*

*platform\_info = player['platform\_info']*

*player\_name = platform\_info['platform\_user\_nick']*

*if player\_name in current\_player\_name:*

*stats = player['stats']*

*accuracy = stats['accuracy']*

*hs\_perc = accuracy['headshots\_percent']*

*return hs\_perc*

*else:*

*continue*

*def getTeam(self,game\_file,current\_player\_name):*

*data = openJsonFile(game\_file)*

*match = data['match']*

*players = match['players']*

*for player in players:*

*platform\_info = player['platform\_info']*

*player\_name = platform\_info['platform\_user\_nick']*

*if player\_name in current\_player\_name:*

*metadata = player['metadata']*

*team = metadata['team\_id']*

*return team*

*else:*

*continue*

*def getAgent(self,game\_file,current\_player\_name):*

*data = openJsonFile(game\_file)*

*match = data['match']*

*players = match['players']*

*for player in players:*

*platform\_info = player['platform\_info']*

*player\_name = platform\_info['platform\_user\_nick']*

*if player\_name in current\_player\_name:*

*metadata = player['metadata']*

*character = metadata['character']*

*agent = character['name']*

*return agent*

*else:*

*continue*

*def getMultiKills(self,game\_file,current\_player\_name):*

*multi\_kills = 0*

*data = openJsonFile(game\_file)*

*match = data['match']*

*players = match['players']*

*for player in players:*

*round\_results = player['round\_results']*

*platform\_info = player['platform\_info']*

*player\_name = platform\_info['platform\_user\_nick']*

*if player\_name in current\_player\_name:*

*for round in round\_results:*

*kills = round['kills']*

*if kills >= 3:*

*multi\_kills += 1*

*return multi\_kills*

*else:*

*continue*

*class Game: #Game specific information*

*def \_\_init\_\_(self,game\_file):*

*self.players = self.getPlayers(game\_file)*

*self.red\_team = self.createRedTeam(game\_file)*

*self.blue\_team = self.createBlueTeam(game\_file)*

*self.map\_name = self.getMapName(game\_file)*

*def getPlayers(self,game\_file):*

*try:*

*players\_list = []*

*data = openJsonFile(game\_file)*

*players = data['match']["players"]*

*for player in players:*

*platform\_info = player['platform\_info']*

*player\_name = platform\_info['platform\_user\_nick']*

*players\_list.append(player\_name)*

*return players\_list*

*except Exception as e:*

*print(e)*

*def createRedTeam(self,game\_file):*

*try:*

*red\_team = []*

*data = openJsonFile(game\_file)*

*players = data['match']["players"]*

*for player in players:*

*platform\_info = player['platform\_info']*

*player\_name = platform\_info['platform\_user\_nick']*

*metadata = player['metadata']*

*team = metadata['team\_id']*

*if "Red" in team:*

*red\_team.append(player\_name)*

*return red\_team*

*except Exception as e:*

*print(e)*

*def createBlueTeam(self,game\_file):*

*try:*

*blue\_team = []*

*data = openJsonFile(game\_file)*

*players = data['match']["players"]*

*for player in players:*

*platform\_info = player['platform\_info']*

*player\_name = platform\_info['platform\_user\_nick']*

*metadata = player['metadata']*

*team = metadata['team\_id']*

*if "Blue" in team:*

*blue\_team.append(player\_name)*

*return blue\_team*

*except Exception as e:*

*print(e)*

*def getMapName(self,game\_file):*

*try:*

*data = openJsonFile(game\_file)*

*\_map = data['match']['map']*

*return \_map['name']*

*except Exception as e:*

*print(e)*

*class Matches: #Class for match history*

*def \_\_init\_\_(self,matches\_file):*

*self.names = self.getNames(matches\_file)*

*self.current\_rank = self.getCurrentRank(matches\_file)*

*self.agents\_played = self.getAgents(matches\_file)*

*self.gameAPILink = self.getGameID(matches\_file)*

*def getNames(self,matches\_file):*

*try:*

*match\_names = []*

*data = openJsonFile(matches\_file)*

*match\_list = data['matches']*

*for match in match\_list:*

*metadata = match['metadata']*

*\_map = metadata['map']*

*name = \_map["name"]*

*match\_names.append(name)*

*return match\_names*

*except Exception as e:*

*print(e)*

*def getAgents(self,matches\_file):*

*try:*

*agent\_names = []*

*data = openJsonFile(matches\_file)*

*match\_list = data['matches']*

*for match in match\_list:*

*metadata = match['metadata']*

*character = metadata['character']*

*name = character['name']*

*agent\_names.append(name)*

*return agent\_names*

*except Exception as e:*

*print(e)*

*def getCurrentRank(self,matches\_file):*

*try:*

*current\_rank = []*

*data = openJsonFile(matches\_file)*

*match\_list = data['matches']*

*for match in match\_list:*

*stats = match['stats']*

*rank = stats['rank\_name']*

*current\_rank.append(rank)*

*return current\_rank*

*except Exception as e:*

*print(e)*

*def getGameID(self,matches\_file):*

*try:*

*game\_id = []*

*data = openJsonFile(matches\_file)*

*match\_list = data['matches']*

*for match in match\_list:*

*metadata = match['metadata']*

*id = metadata['id']*

*game\_id.append(id)*

*return game\_id*

*except Exception as e:*

*print(e)*

*class Economy:*

*def \_\_init\_\_(self,game\_file,red\_team,blue\_team):*

*self.red\_team = red\_team*

*self.red\_economy = self.getRedEconomy(game\_file,red\_team)*

*self.blue\_team = blue\_team*

*self.blue\_economy = self.getBlueEconomy(game\_file,blue\_team)*

*def getBlueEconomy(self,game\_file,blue\_team):*

*data = openJsonFile(game\_file)*

*match = data['match']*

*players = match['players']*

*round\_econ = [0] \* len(match['rounds'])*

*win\_econ = [0] \* len(match['rounds'])*

*kill\_econ = [0] \* len(match['rounds'])*

*for player in players:*

*platform\_info = player['platform\_info']*

*player\_name = platform\_info['platform\_user\_nick']*

*if player\_name in blue\_team:*

*rounds = match['rounds']*

*for i,round in enumerate(rounds):*

*winning\_team = round['winning\_team']*

*if "Blue" in winning\_team:*

*win\_econ[i] += 3000*

*else:*

*win\_econ[i] += 1900*

*round\_results = player['round\_results']*

*for i,result in enumerate(round\_results):*

*kills = result['kills']*

*kill\_bonus = kills \* 200*

*kill\_econ[i] += kill\_bonus*

*econ\_data = []*

*for i,total in enumerate(round\_econ):*

*total\_econ = kill\_econ[i]+win\_econ[i]*

*econ\_data.append((i,total\_econ))*

*print(econ\_data)*

*return econ\_data*

*def getRedEconomy(self,game\_file,red\_team):*

*data = openJsonFile(game\_file)*

*match = data['match']*

*players = match['players']*

*round\_econ = [0] \* len(match['rounds'])*

*win\_econ = [0] \* len(match['rounds'])*

*kill\_econ = [0] \* len(match['rounds'])*

*for player in players:*

*platform\_info = player['platform\_info']*

*player\_name = platform\_info['platform\_user\_nick']*

*if player\_name in red\_team:*

*rounds = match['rounds']*

*for i,round in enumerate(rounds):*

*winning\_team = round['winning\_team']*

*if "Red" in winning\_team:*

*win\_econ[i] += 3000*

*else:*

*win\_econ[i] += 1900*

*round\_results = player['round\_results']*

*for i,result in enumerate(round\_results):*

*kills = result['kills']*

*kill\_bonus = kills \* 200*

*kill\_econ[i] += kill\_bonus*

*econ\_data = []*

*for i,total in enumerate(round\_econ):*

*total\_econ = kill\_econ[i]+win\_econ[i]*

*econ\_data.append((i,total\_econ))*

*print(econ\_data)*

*return econ\_data*