**Capstone Project Report**

Discord Tournament Administration

Kennesaw State University

IT4983-W01 Capstone

Lily Lama-Solet - Shane Wild - Brandynn Daugherty - Grant Richardson - Jonathan Pass

Project Website: https://github.com/llamasol/DiscordTournamentBot

March 24, 2024

# **Executive Summary**

With Kennesaw State University’s Esports division, there still remains a great deal of manual labor needed to initiate and manage in-house tournaments. Because of this, tournaments take an unnecessary amount of time to get started. To facilitate in-house tournaments, KSU Esports admins utilize Discord for communication with players. This provides a way for players to check-in to tournaments and be placed into teams by admins based on their in-game rank.

The goal of our team is to develop a tournament bot on Discord that the admins and players can use. With a Discord bot, players can check in faster and be matched up in teams more efficiently. Admins will also be able to use the bot for moderation of the matches and push or pull player statistics from a spreadsheet instantly.

# **Table of Contents**

[**Executive Summary 3**](#_9qw9fbo1583l)

[**Table of Contents 4**](#_xnqihw6wc777)

[**Background 5**](#_251jqpcallh8)

[**Project Outcomes and Achievements Summary 6**](#_gi7i992jdnk)

[**Project Planning and Management Summary 7**](#_kgukn8a6lbkx)

[**Team Reflection on Project Experience 8**](#_28wie7d3bb4w)

[**Appendix 9**](#_yydie8pe4cuf)

[**Bibliography 10**](#_txn80enjogci)

# **Background**

Kennesaw State University runs an eSports program that regularly hosts events and tournaments for games like League of Legends and Super Smash Bros. As of right now, tournament administration tasks like team formation and keeping track of player statistics is all done manually, specifically using spreadsheets. Because of this, the overall tournament administration process can take more time than it should, which leads to players waiting an excessive amount of time to be put on teams and can delay the tournament start time. The current system also creates the possibility of human error when tracking player data and doing other tasks, since everything is done manually.

The goal of this project is to automate the tournament administration process. In order to achieve this, we will create a Discord bot that will be able to handle tournament administration tasks. For example, the bot will be able to assign players into teams automatically based on relevant player information and determine which teams will go against each other for each match. Matchmaking algorithms will be implemented into the bot’s code in order to achieve this. Other tournament administration tasks like keeping track of and storing player statistics will also be handled automatically by the bot. Also, this will be a bot that not only tournament administrators can use, but it will be something that players will be able to use as well. Players will be able to check-in by simply interacting with the bot and using a check-in command that will allow them to leave and check-in.

In terms of the more technical side of the background around this project, we used the programming language Python for the code for the bot, since it is very user friendly and easy to use when compared to other languages. We also made use of Discord’s Python API library, and it allowed us to add functionality to the bot. The Discord API library lets the bot be interacted with, and allows it perform a variety of actions, like responding to certain text prompts and performing an action after a command is issued, using Python code (Welcome to discord.py). We also made use of Google Sheet’s API library, which we needed in order to allow the bot to read and write to Google Sheets spreadsheets. The Google Sheets API allows users to do many things like create Google Sheets spreadsheets, read from and write to spreadsheets, and update or edit cells within spreadsheets using code (Google). The overall intent of this entire project is to automate various tournament administration tasks by creating a Discord bot that can perform these tasks with ease.

# **Project Outcomes and Achievements Summary**

The first task was to create the Discord bot. This was done by creating a new Discord Application in Discord’s Application page. The bot was given the appropriate permissions based on what features the admins will use. Once it was created, a URL was generated by ticking the necessary boxes for the bot’s scope and permissions in the OAuth2/URL Generator tab. With this URL, the bot was able to be invited into the KSU Esports test server. (Creating a bot account)

The next step was to develop several admin commands. The commands are ‘/checkin’, ‘/volunteer’, ‘/players’, ‘/matchmake’, ‘/points’, ‘/wins’, ‘/remove’, ‘/toxicity’, and ‘/votemvp’. The ‘/checkin’ and ‘/volunteer’ commands will provide buttons for users to click to check into or leave tournaments, volunteer out or rejoin a tournament, respectively. The ‘/votemvp’ command will let players use a slash command to choose who they think was the best in the match, which will then send a message to the ‘admin’ channel showing who voted and the player they chose. The ‘/toxicity’ command is explicitly used by an admin for moderation purposes for when a player is acting in a negative manner.

The ‘/players’ command uses two separate for loops, one for players and one for volunteers and adds all users in the player and volunteer roles to two separate lists. It will then display three separate embeds. The first will show how many more players or volunteers are needed (if the number of players is not divisible by 10) or that at least one lobby is ready to matchmake. The second embed will display all of the users in the players role and show the number of players at the bottom; the last embed does the same, but for users in the volunteer role.

The ‘/matchmake’ command reads from the spreadsheet containing each player’s tier and preferred in-game roles. With this data, it will then generate up to 50 teams (can generate exponentially more, but throttled to prevent a massive amount of permutations) and first sorts players by their tier to balance teams, then assigns everyone based on their preferred role. Depending on who signs up for matches, players could potentially not be placed in their preferred role; however, we felt this was the best way to ensure players were not placed against opponents of greater skill level.

The ‘/points’ command checks which users are in the player and volunteer roles, then uses the ‘update\_points’ method to add one point to the appropriate values for those players and volunteers. If a user is a player, then they receive one point in participation and games played. If a user is a volunteer, then they receive one point in participation. The ‘/remove’ command is used to take every user that is in the player or volunteer role and remove them from those roles to make it easier for admins to initiate the next match.

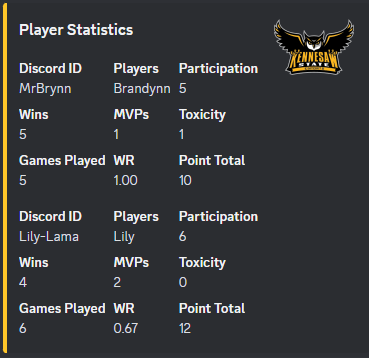
The ‘/wins’ command is rudimentary in how it can add points to how players who won a match are given a point in their win score on the Google Sheets spreadsheet. This command takes in 5 strings, which will be the 5 winner’s Discord ID usernames. It will then search the spreadsheet for those names and if one cannot be found (e.g., due to a misspelled username), it will not update those win scores, causing an admin to redo the command; therefore, retyping the 5 names.

For the buttons that appear from the check-in and volunteercheck commands, we had to implement them to respond directly to each individual player privately so as to not flood the text channel with messages. We also implemented a class in the code to create the buttons and their functionality. The buttons were given a timeout so it would be unusable after 15 minutes. Each one was also given a different color and label to help users differentiate between them. We initially wanted a button to be disabled for a user when they clicked on it; however, if you set a button to be disabled, then after the first click it is disabled for everyone. To work around this, when a button is clicked the first time, it will send the initial message based on the button; then if a user tries to click on a button a second time, it will let them know they have already done so. (API docs for bots and developers; lykn)



In order to connect the Discord bot to the Google Sheets API, there were several prerequisites that needed to be done in order to establish a connection between them. For starters, you must create a project in Google Cloud Console. From there you have to enable the Google Sheets API. The next step was to configure consent in the OAuth screen and add a gmail account as a test user allowing it to have permission to access the Google Sheet from the Discord bot. The third step in setting up the API environment was to create credentials that will allow test users authorization. The credentials can be created by going to the credentials tab (left side) and selecting ‘Create Credentials’ at the top, then select OAuth client ID and fill out the necessary information. Once created, you must download it as a JSON file and store it as “credentials.json” and can be stored wherever you need it to. Lastly, Google’s client library must be installed using the command “pip install --upgrade google-api-python-client google-auth-httplib2 google-auth-oauthlib”. (Python quickstart)

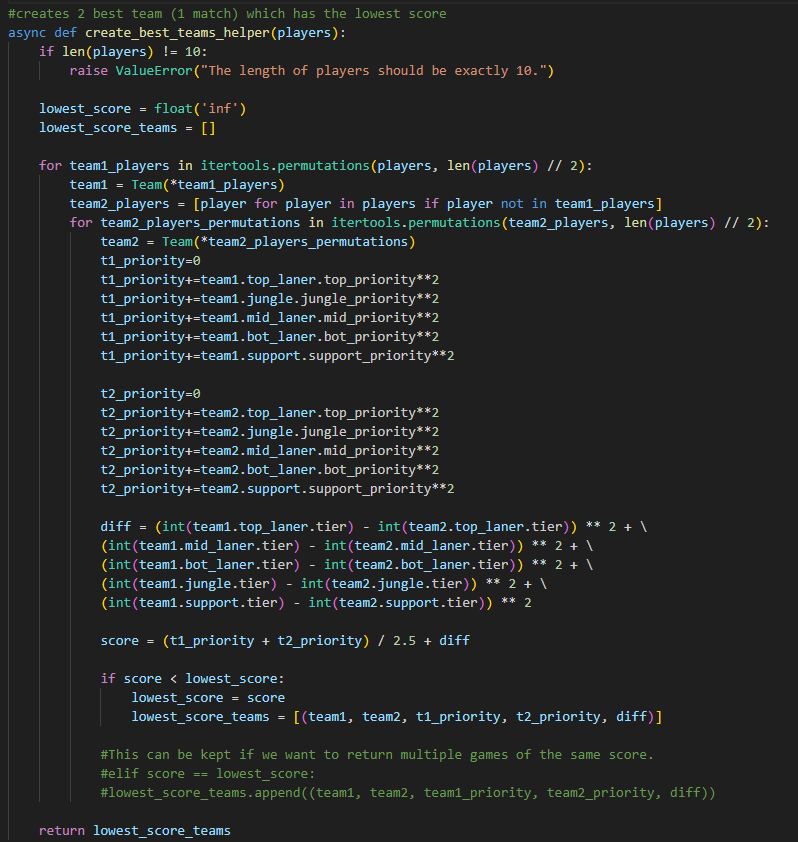
Once the environment has been set up, the code can now be implemented. Google has provided several helpful methods to get started on reading and writing to Google Sheets using multiple coding languages. The downside to these methods is they are only quickstart ones, so it gets us connected to the API but getting it to read and write how we need it requires extra code. (Python quickstart)

To get the bot to display an output that was readable and visually appealing, the data that is read from Google Sheets is displayed using Discords embeds. The slash commands ‘/matchmake’ and ‘/players’ displays helpful information for admins. The ‘/matchmake’ command will first display the users who volunteered to sit out of a match and each subsequent embed will display all of the lobbies showing the teams 1 and 2 and in-game roles with the respective players assigned to those roles on both teams. (API reference)

In regards to the matchmaking method, players will get a tier from 1-6. This matchmaking method shown below will take 10 players which consist of name, tier, as well as their preferred roles (top, jungle, middle, bottom, support) on a scale from 1 to 5. The algorithm loops through all possible 3,628,800 player permutations, splitting them into two different teams. First it will organize players by their tier, then assign them based on their preferred role. Each match-up will assign the game a score, which is calculated as:

score=(t1top laner​ − t2 top laner)2 + (t1 mid laner ​− t2 mid laner)2 + (t1bot laner​ − t2 bot laner)2 +

(t1 jungle​− t2 jungle​)2 + (t1 support ​− t2 support​)2

This score is added to the squared values of all the players preference of roles based on the roles they get. So if the top laner has a 3 for their priority on top, it will add 9 to the score of the game. The current implementation takes the score of the difference of tiers between players plus the players priority divided by 2.5..

**Project Planning and Management Summary**

Our project was broken up into 3 milestones. The first milestone was divided into 3 parts. The first part was to research information needed to create the bot, develop the commands and buttons, and prepare all documentations for submission.

To get us started on the research, we decided on using Python for the bot’s code. With this, we could get started on researching the documentation on how to develop the commands and buttons using Python. Discord.py provided a lot of helpful documentation on developing the bot because it explained the purpose of the keywords and how to use them (Welcome to discord.py). After doing research on Discord’s API library, we moved on to developing the bot. We created two commands; a check-in command and a re-check/volunteer command. The check-in command allows players to both check-in and leave games and the re-check/volunteer command allows players to volunteer to sit out of games and re-join games. Our main goal for milestone 1 was to have the bot be able to be interacted with by using commands and buttons.

For milestone 2, we started off doing some additional research, particularly on Riot’s API and Google Sheets API, which are two API’s that we needed to utilize for this stage of the project. For the Riot API, there is a developer portal that has a lot of useful information, like how to properly use the API, information on API keys and how to receive one, error codes, and rate limits (Riot Developer Portal).

Google also has a developer portal with an overview on the API and how to use it. There’s a lot of good information on it that was very useful for us like creating spreadsheets, reading and writing to spreadsheets, and updating spreadsheets (Google). After that, we went right into working on creating the code that will allow the bot to read and write to spreadsheets. We also decided to get started on the matchmaking algorithm, which is something that is meant to be a requirement for milestone 3, but we decided to get started on it during milestone 2. Having a demo for the matchmaking proved useful, as it allowed our sponsor to offer suggestions which went into milestone 3. While we were mostly successful when it came to milestone 2, we did run into some challenges, particularly when it came to having the bot read and write to spreadsheets. While we were overall successful with getting the bot to read to Google Sheets spreadsheets, we ran into some trouble with getting the bot to write to spreadsheets and couldn’t get that method fully working before the milestone 2 due date. Also, Discord’s embeds, which is what we are using to display teams after matchmaking, is also pretty limiting since the field limit is lower than we’d like it to be. Overall, even with the setbacks, milestone 2 was mostly successful.

Milestone 3 mainly revolved around finishing up the matchmaking algorithm and the Google Sheets read/write function and integrating them into the main code for the bot. Up until the end of milestone 3, the Google Sheets read/write function and the matchmaking algorithm were made separately outside of the main code for the bot, since we found it easier to make and test them separately before integrating them into the bot. After the Google Sheets read/write function and the matchmaking algorithm were finished and integrated into the bot, we spent a solid amount of time debugging the code. While we finished everything that was required of us in the end, not everything went smoothly. One unfortunate thing that happened was that the Riot API ended up getting unused since it was difficult to use it to pull players and their data and put them into the Google Sheets spreadsheets through the bot. However, this was something that was extra and not required of us but would’ve been nice for our sponsor to have. In place of this, we implemented the matchmaking algorithm to go off each player's tiers, which is given by an admin, to get a similar outcome. We also found Python to be a bit more limiting than we’d like, particularly when it comes to its package functionality and how it handles Google API requests. Lastly, one thing that we wanted to do but couldn’t was expand the bot and allow it to run and administer tournaments for other competitive games outside of League of Legends, but this wasn’t a minimum requirement for us. Milestone 3 was overall successful and we finished the project with all requirements and objectives met.

# **Team Reflection on Project Experience**

While there were a few setbacks during the project, we were overall very successful. Some of the biggest factors that led to our success were effective and constant communication and the skill and technical knowledge of our team members. This project required a good amount of technical knowledge and skills, like knowing how to work with API’s and programming, and our team members were skilled and capable of facing the challenges that the project gave us. Also, if some or all of our group members lacked certain knowledge that was needed or would be helpful, then we effectively did our research and gained that knowledge.

For communication and collaboration, we mainly used Discord. We met on Discord every weekday at 8:30 p.m., and while it wasn’t as necessary when the project first started since the workload was smaller back then, it became much more necessary as the project went on. Discord was a very good platform to use for communication since the bot we were making was for Discord and it allowed us to communicate within the server that we did our testing in. We also used Microsoft Teams for communication, however we used it solely to conduct our weekly meetings with our sponsor. Our team had a strong preference for Discord, since we found it much easier to use than Microsoft Teams, so we used Discord most of the time for communication.

While our group was very successful, we did run into some setbacks. One of our biggest setbacks was during milestone 2, which was when we could only get the bot to read from Google Sheets spreadsheets but couldn’t get it to write to spreadsheets before the due date. One of our requirements for milestone 2 was being able to have the bot both read from and write to spreadsheets, so this setback did cost us. Another challenge we ran into involved Discord’s embeds, which is what we were using to display teams after the matchmaking process. Discord’s embeds have both a character and a field limit, and while the character limit was no problem, the field limit was lower than we would’ve liked. Another big setback that we ran into during milestone 3 was having to ditch the Riot API, since we found it difficult to use it to pull players and their data and put them into Google Sheets spreadsheets using the bot. The good thing though is that this was not a minimum requirement for us, so it wasn’t detrimental that it was left out, but it would’ve been nice to have implemented. We also weren’t able to look into expanding the bot’s functionality and allow it to be used for competitive games other than League of Legends, but this was also not a minimum requirement for us. One last challenge was that we found Python to be a bit more of a limiting programming language than we would’ve liked, particularly when it comes to how it handles Google API requests and its package functionality.

# **Appendix**

# **Bibliography**

*API docs for bots and developers. Discord Developer Portal. (n.d.). https://discord.com/developers/docs/interactions/application-commands*

*API reference¶. API Reference. (n.d.). https://discordpy.readthedocs.io/en/latest/api.html?highlight=embed#embed*

*Creating a bot account*. Creating a Bot Account. (n.d.). https://discordpy.readthedocs.io/en/stable/discord.html

Google. (n.d.). *Google Sheets API Overview*. Google for Developers. https://developers.google.com/sheets/api/guides/concepts

Google. (n.d.). *Python quickstart | google sheets | google for developers*. Google. https://developers.google.com/sheets/api/quickstart/python

lykn. (n.d.). *A Gist which shows/tells you how to make buttons using discord.py v2(version 2.0.0a)*. Gist. https://gist.github.com/lykn/bac99b06d45ff8eed34c2220d86b6bf4

*Product registration*. Riot Developer Portal. (n.d.). https://developer.riotgames.com/docs/portal

Welcome to discord.py. (n.d.). https://discordpy.readthedocs.io/en/latest/