

CS 4850 – Section 2 – Spring 2025

02-T3 Tournament Bot

Jordan Danh, Charles Hyun, Alex Vuong

Sharon Perry

April 20, 2025

Website Link: <https://jordandanhksu.github.io/>

GitHub Link: <https://github.com/jordandanhKSU/TournamentBotKSU>

STATS AND STATUS		
LOC	3617	
Components/Tools	5: GitHub, Excel, .db file, Discord, Riot Games API	
Hours Estimate	388	
Hours Actual	400	

Contents

Introduction	3
Requirements	3
Project Goals	3
Definitions and Acronyms	3
Assumptions.....	4
Design Constraints	4
Environment.....	4
User Characteristics	4
Functional Requirements	4
Non-Functional Requirements	6
External Interface Requirements.....	6
Analysis & Design	9
Assumptions and Dependencies	9
Architectural Strategies.....	9
System Design Overview.....	9
Detailed System Design	10
Development	11
Database Connection	15
Test Plan & Report.....	15
Version Control	16
Conclusion	16
Appendix A: User Ability Guide.....	18
Appendix B: Gantt Chart	19

Introduction

The Tournament Bot project is owned and sponsored by KSU eSports, and they are seeking some help in developing a new tournament bot to help the organizers run their in-house League of Legends tournaments. These tournaments are currently being run manually using a google sheet to track attendance, seed brackets and track results and player data.

Currently, manual tournament administration has caused these tournaments to have major wait times for players, bias in bracket formation, and room for error in data tracking.

This project has been a continuation project since Spring 2024 and is seeking improvements in automation to avoid tedious manual work for tournament organizers.

Currently, our job is to implement functionalities that handle admin duties (team placement, score keeping, tracking player performance), implement functional algorithm-based matchmaking, create and update a database for player statistics, implement user-friendly UI using Discord embeds and commands, add proper integration with game developer APIs to pull player data, and create testing plan and testing execution status.

Requirements

Project Goals

- An effective matchmaking algorithm that accounts for player ranks, statistics, etc.
- User-friendly UI using Discord embeds and commands
- Increase efficiency in tournament admin duties
- Create and update a database for player statistics
- Pull player data from relevant game developer's API

Definitions and Acronyms

- Discord – Communication platform that allows developers to host bots to add user specific features. This is where we will be hosting our tournament bot
- League of Legends – Online 5 versus 5 competitive games. This is the main game that this project focuses on getting the tournament bot to function for.
- Riot Games – Company that creates league of legends. They also give access to their database by giving developers API access.
- Tournament Admins – project's main stakeholders
- In-Houses – Tournament ran weekly by the KSU eSports League of Legends team
- Matchmaking – The process that matches players together which is meant to make multiple balanced teams

Assumptions

- It is assumed that Discord will allow free access to create Discord bots for the developers.
- It is assumed that Riot Games will allow developers to utilize their API key if we give them a proper reason for using it.
- It is assumed that the clients are familiar with using and interacting with Discord bots. The users can use the /help command to guide them on how to use the bot.
- It is assumed that Google allows their API to be used when working with a Google spreadsheet.

Design Constraints

Environment

The Tournament Bot will be developed through one of Discord's functionalities which allow users to develop their own Discord bot. This Discord bot will be an effective tool for hosting KSU League of Legends in-houses. The Discord bot will also update a google spreadsheet of player statistics from these in-houses. The Discord bot will also provide players with user-friendly UI for easy interactions when registering for tournaments.

User Characteristics

Stakeholders / Tournament Admins (Top Priority)

- Expected to be knowledgeable about the game League of Legends
- Expected to be knowledgeable about the KSU eSports Community
- Expected to have basic knowledge of using the discord platform

These users are the top priority for this project because they will be the main users of the Tournament Bot. The bot's main purpose is to help with administrative tasks, which is valuable only to the Tournament Admins.

Players

- Expected to know how to play the game League of Legends
- Expected to have basic knowledge of using the discord platform

Players have a smaller role when it comes to using this Tournament Bot. There will be features available to players such as looking up your player data like that of win rate or preferred player role, but the main purpose of Players in this system is to apply to play a game in the in house and then play said game.

Functional Requirements

- 3.1 Starting a Tournament
 - Link Riot ID
 - Players must use /link [riot_id]

- Takes data from Riot API and puts information into a spreadsheet
 - Assign role preferences
 - Orders the player's role preferences
 - Unlink Riot ID
 - Players can change their Riot ID by unlinking their current ID
 - Volunteer
 - Gives players an option to sit out but gives them a participation point
 - Check In
 - Players can participate by checking in into a list of players
 - Needs at 10 players. Groups of 10 for multiple games
 - Leave
 - Players can leave check in if they choose to leave for reasons
- 3.2 Matchmaking
 - Assigns possible teams through an algorithm
 - Considers player statistics and ranks
 - Could lead to potential imbalances
 - Same teams could be created so it needs deviation
 - Swap functionality
 - Allows admins to swap players around
 - Account for factors that the matchmaking cannot predict
 - Players not getting along or players want to play with friends
 - Post Matchmade teams
 - After teams are created through the matchmaking algorithm and have been finalized, the teams will be posted in the channel where check-in started
- 3.3 Statistic Tracking
 - Tracks player performance, win/loss
 - Admins can set what team wins (red or blue)
 - Tracks participation, wins, toxicity, and MVP
 - MVP is given through a voting system by the winning team
 - Admins can allow players to vote for an MVP
 - Admins can skip this process
 - Toxicity is given by admins through reports
 - Statistics are recorded in a spreadsheet
 - May need to give admins permission to edit the spreadsheet so they can manually edit statistics
 - Statistics are exported to an excel sheet and database
 - Player stats, player matches
- 3.4 Post Game Interactions
 - Admins are given a choice to start the check-in process again or stop the games

Non-Functional Requirements

- Security
 - An important aspect of this project is security. This is because we are gathering data on players, and we should be responsible for that data we collect. This means it is imperative to follow proper procedures when handling tokens, keys, and links or anything of that nature that may expose our databases to malicious users.
- Usability
 - One of the main requirements of this project is to develop a product that allows users to be able to use the bot with its user-friendly interface. By using Discord's embed and command features, we can ensure that the product will allow the users to perform tasks effectively.
- Maintainability
 - The project is a continuation of past projects, and it is important to build off the previous projects and be able to create a product that can be improved on for future projects. Currently, the stakeholders want the developers to produce a product that uses a complex matchmaking algorithm as the main priority.
- Scalability
 - The final goal of this project is to be able to use this bot to host larger tournaments, so it is important to create a product that can handle large amounts of players while managing a large amount of data.

External Interface Requirements

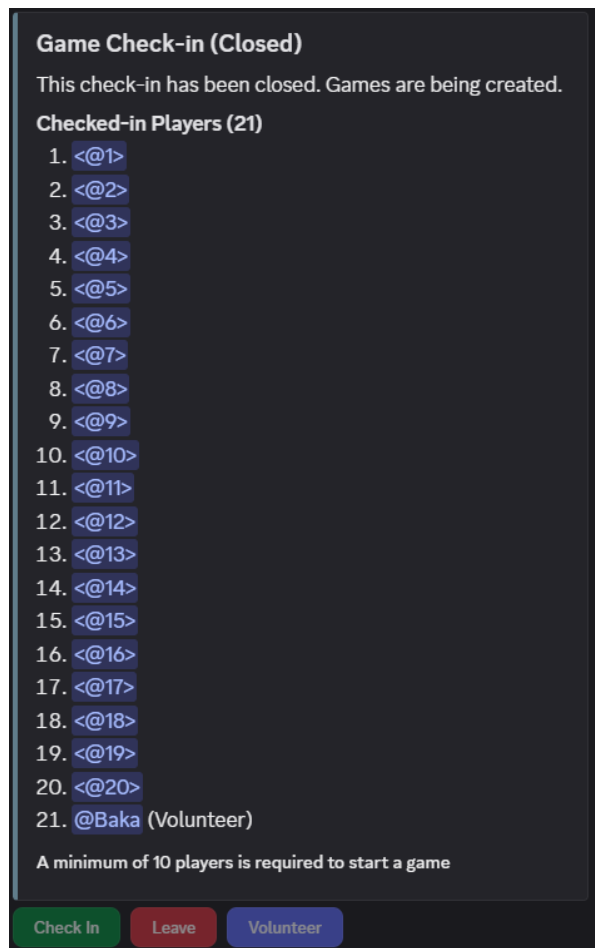
User Interface Requirements

- The users will be prompted to link their Riot ID with their Discord account. Users can also list their role preferences to help with the matchmaking algorithm.


































The screenshot shows a Discord interface with a dark theme. At the top, there is a header bar with the text "Select your role preferences (1 (high) to 5 (never))" in a light blue font. Below this, a line of text displays "Current preferences - Top: 4, Jng: 4, Mid: 4, ADC: 4, Sup: 4". Underneath, there are five vertical dropdown menus, each with a light blue label and a downward arrow icon. The labels are "Top preference..", "Jng preference..", "Mid preference..", "ADC preference..", and "Sup preference..".

- This will allow the users to be able to interact with the tournament bot. Users can check-in by clicking on the “Check-In” button, and they can cancel if they don’t want to participate.



- When a tournament starts, users can display the players and volunteers through a simple command.

Game 1		
Blue Team	Tier and Rank	Role Preference
 Top: Baka	Tier: 7 Rank:	 Top,  Jun
 Jun: P19_DIA_B	UNRANKED	 Bot,  Sup
 Mid: P14_PLA_B	Tier: 3 Rank: DIAMOND	 Bot,  Sup
 Bot: P13_PLA_M	Tier: 4 Rank: PLATINUM	 Mid,  Sup
 Sup: P10_GOL_S	Tier: 4 Rank: PLATINUM	 Bot,  Sup
	Tier: 5 Rank: GOLD	
Red Team	Tier and Rank	Role Preference
 Top: P18_EME_M	Tier: 3 Rank: EMERALD	 Mid,  Sup
 Jun: P17_EME_J	Tier: 3 Rank: EMERALD	 Jun,  Sup
 Mid: P15_PLA_S	Tier: 4 Rank: PLATINUM	 Bot,  Sup
 Bot: P16_EME_T	Tier: 3 Rank: EMERALD	 Top,  Sup
 Sup: P11_GOL_T	Tier: 5 Rank: GOLD	 Top,  Sup
Result		
 Blue Team Wins!		

- Once there are enough players, the matchmaking algorithm will show potential teams for the admins to pick from to start the games.
- After a game ends, the users will be prompted to give the game's results, and the winning team will be able to vote for their MVP. The results will record player participation, wins, MVP, and toxicity on to a Google Sheet.

Software Interface Requirements

- The project will be mainly on Discord as the tournament bot is being operated on there.
- The project will be coded on GitHub and will be used as version control.
- The project will take data from Riot Games's API to track player statistics.
- The project will use Google Sheets to record player statistics from the games played.

Communication Interface Requirements

- The project will need to be able to communicate with Discord and Riot Games as being able to use their API and bot functionalities to make this project work.

Analysis & Design

Assumptions and Dependencies

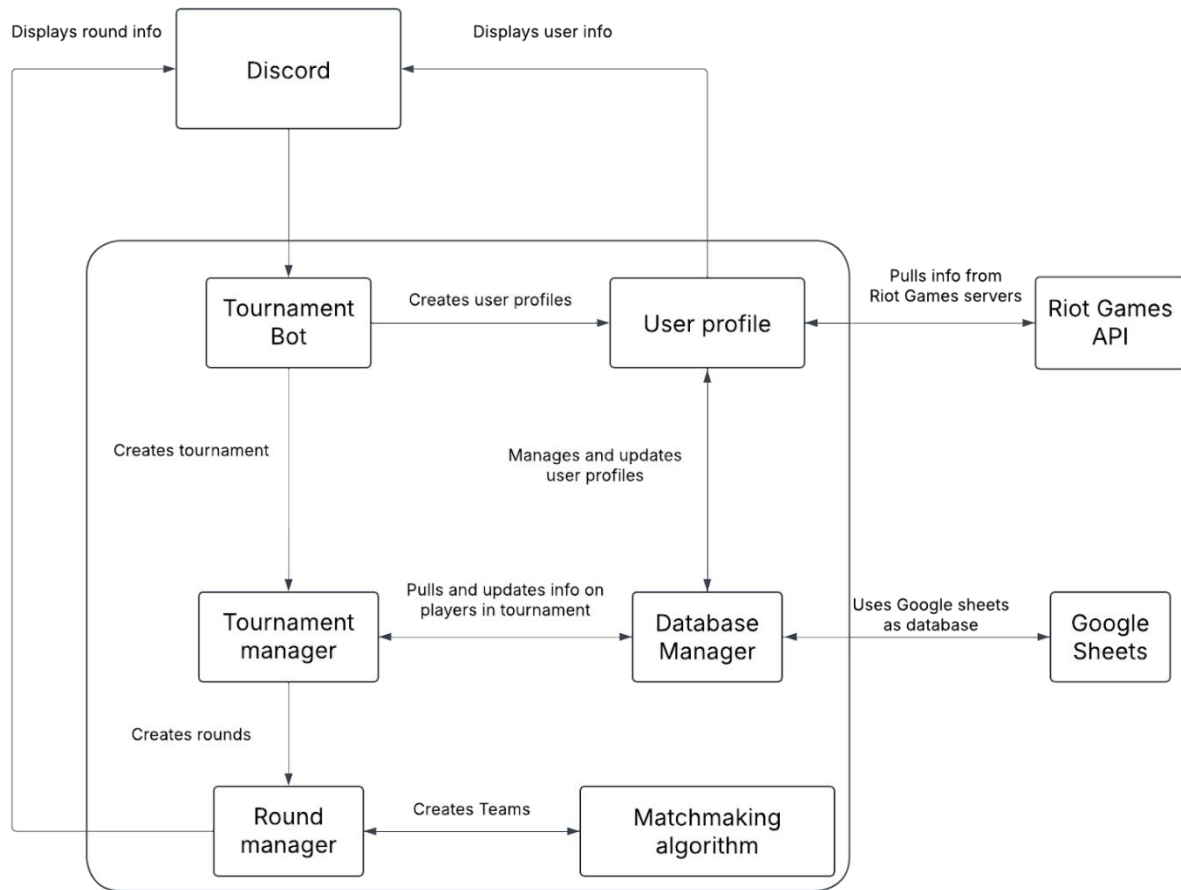
The bot relies on multiple APIs to function, including the Riot Games API, the Discord API, and the Google Sheets API. It is also assumed the bot will be hosted on a server, whether an online service or one owned by KSU, that will be able to run Python. Possible future changes would include future support for more games that would need different API keys to pull player data.

Architectural Strategies

The Discord bot will be programmed in Python because the previous bots we are expanding upon using Python, and the libraries provided to develop Discord bots like Discord.py. Python also has libraries to support asynchronous completion of tasks, which would improve the speed of creating multiple teams through the matchmaking algorithm. Google Sheets was chosen as the database for ease of use and simplicity, allowing admins who are not tech-knowledgeable to edit and update player information manually. Discord embeds will be implemented for ease of use of players, so many users can interact with the bot without needing to memorize the text commands. The program will be modular enough to easily allow more games to be added and use existing code to create teams and matchmake for future development.

System Design Overview

Players and admins will interact with the bot through Discord, either through embeds or commands. The main component is the Tournament Bot, which will register users and start tournaments. Players will create a user profile by linking their Riot account to their Discord account, which is then stored in a Google Sheets by the database manager. Once a tournament is created by the tournament manager, it will pull the information of checked in players from the database and create rounds of different teams based on the matchmaking algorithm. Both the information of the round and of user profiles can be displayed back to Discord.



Detailed System Design

The Tournament Bot will have multiple commands to enable players to register and admins to host and manage the tournament. Players will first register their account by linking their Riot Games' user ID, which will pull their current rank through the Riot Games API. Players will also be able to set their preferred roles. The information is then sent to the database manager that uses Google Sheets to track information such as rank, in-house tier, tournament win rate, points, role preference, and other stats. Admins can also manually add in or edit player information, such as penalizing players with a toxicity point. Users

Admins and tournament organizers can start tournaments through the bot by first starting a check-in period for players. The check-in will create a button for players to press to either sign up for the tournament or volunteer if there are not enough players. If there is an eligible number of players to start the tournament (multiple of 10), the tournament manager will send the player information to the round manager, which will split the players into groups of 10 based on their rank and tier.

After every player has been split into their respective groups, the matchmaking algorithm will then attempt to create balanced teams of the 10 players to face off. The algorithm would take in

different weights, such as rank, preferred role, win rate, and other factors to determine fair and balanced teams that have similar chances of winning. The algorithm would also be able to prioritize different weights based on the tournament organizer's discretion. The algorithm will then give a selection of three options of teams that can be chosen and edited by admins. Once the teams have been decided, the round information will be displayed, and the game can start. After the game, admins can select which team won, award or deduct points, and start the next set of rounds.

Development

- Matchmaking
 - Player List
 - First will take in a player list of x size, which will then sort each player in descending order from highest rank. It will then split the list into groups of 10 based on those ranks.
 - Assign Teams
 - For each group of 10, the algorithm will first start off with a randomized team, where each player will be assigned to one of the two teams and assigned to a random role.
 - Matchmaking Algorithm
 - The overall "fitness" level will then be calculated for this set of teams.
 - First it will find the overall difference in "prowess" level between the two teams.
 - This is done by finding each player's prowess and comparing the difference between the two players in a certain role.
 - Then it will find how preferable the role assigned to each player is.
 - These two factors will give an overall fitness score that shows how balanced the teams are and how many players have a preferable role.
 - This will then be stored in a key to remember calculated teams
 - Team Creations
 - The algorithm will then create new teams by swapping the players in the team with other positions. These new teams will also be evaluated on a fitness level. The new team with the highest fitness (lower is better), will then be the new team that will be explored. The cycles continue until a team with a desirable fitness level is returned or it has reached a certain epoch.
 - Result
 - The final best determined team is then returned to be used in the tournament
- Database

- The database is needed to store player statistics and data. The matchmaking algorithm also utilizes data from the database like RolePreference.
- DiscordID, DiscordUsername, PlayerRiotID, Participation, Wins, MVPs, ToxicityPoints, GamesPlayed, WinRate, TotalPoints, PlayerTier, PlayerRank, RolePreference are recorded in the main_db.db file.
 - Functions that update the database:
 - Update_wins(winners)
 - Takes in a list of winners and updates a point in Wins and updates the player winrate. Also updates TotalPoints and GamesPlayed by adding 1
 - Update_win_rate(conn, discord_id)
 - Takes in the database connection and DiscordID and calculates the player's winrate
 - Update_points(member)
 - Takes in the player's DiscordID and updates the Participation point by adding 1
 - Updates according to the player's role: Volunteer or Player
 - Update_username(player)
 - Takes in the player's username and updates the DiscordUsername if it is outdated
 - Insert a new player if they do not exist and update accordingly
 - Link(member, riot_id: str)
 - Takes in the player's username and riotID. Updates PlayerRiotID
 - RiotID example: BobIAm#1234
 - Use the RiotGames API to search for the player's rank and update it to the database. This will also calculate the user's PlayerTier
 - Gives an error message if the bot could not find the riotID
 - Update_toxicity(member)
 - Takes in the player's name. Updates ToxicityPoints by 1 and updates TotalPoints by subtracting 1
 - Insert a new player if they do not exist and update accordingly
 - Remove_user(member)
 - Takes in the member's DiscordID and removes the player from the database
 - Add_mvp_point(member)
 - Takes in the member's DiscordID and adds an MVP point by 1

- Insert a new player if they do not exist and update accordingly
 - Set_role_preference(member, preference: str)
 - Takes in the member's DiscordID and a five-character string of the player's role preference
 - Insert a new player if they do not exist and update accordingly
 - Clear_database()
 - Resets the database
- Update_excel(discord_id, player_data)
 - The data in the database is saved in the main_db.db file and then updated to the excel sheet.
 - The function takes in the DiscordID and player data to see if the user is in the spreadsheet and adds them if they are not in the spreadsheet. Then, it updates to the spreadsheet
- Pre-Tournament User Interactions
 - /link
 - Asks the user what their RiotID is and uses the Riot Games API to search for the player's ID
 - Appropriate RiotID: Dude#1234
 - If found, saves in the database
 - If not found, give an error message
 - If the player is left unranked then, the admins can manually override the spreadsheet to list their last season's rank.
 - /unlink
 - Users can unlink their RiotID
 - /rolepreference
 - Ask the user what their role preferences are. Players can select through a drop-down menu. 1 being the most preferred and 5 being the least preferred
 - Role preference example:
 - "12345"
 - The role order is Top, Jungle, Mid, Bot, Support, so Top would be the most preferred and Support would be the least preferred
 - Takes in this string and adds it into the database
 - If the user wants to Fill, then the string will be set to "55555"
 - /createadminchannel
 - For admins to operate the bot, they must create an admin channel in their Discord server and use this command in that admin channel.

- /stats
 - Display player statistics
- Tournament Process Interactions
 - /checkin
 - Admins can start the tournament check-in process. The buttons will be active for 15 minutes
 - Users can click “Check In” or “Leave”
 - Admins can click “Start Game” once there are enough players
 - The matchmaking algorithm will be used to determine the teams from the pool of players
 - Takes in role preference and rank/skill tier
 - Gives a new embed in an admin channel to determine results
 - Swap functionality
 - Admins are given an option to swap players around if there are factors that the matchmaking algorithm cannot account for.
 - Example: Players prefer not to be on the same team as someone
 - The bot will have the ability to manage roles in the Discord Server.
 - /players
 - Will give the user the “player” role to determine that they will be playing in the current match
 - /volunteers
 - Will give the user the “volunteer” role to determine that they will be sitting out in the current match
 - /points
 - This command gives a participation point to the Discord members with the player and volunteers role
- Post-Tournament Game Interactions
 - MVP voting
 - Players on the winning team can vote for MVP
 - The selected MVP gets a point
 - Deciding results
 - In a separate channel, admins determine the result of the game played
 - They can click either the “Blue Team Win” or “Red Team Win” button
 - Gives a point in Wins to the winners
 - Another game
 - Admins can choose to play another game with different teams and players
 - Export
 - Admins can export data from the database to the excel sheet.
 - /toxicity

- Admins can give a player a toxicity point for bad behavior

Database Connection

For the database, it uses AioSQLite Database for real-time analytics, asynchronous support allowing for enhanced scalability, and works well with the nature of how Discord Bots use the async applications. The Microsoft Excel sheet will copy the values of the db file and update them. It will also check if the user is in the database and adds them if they are not. This allows for easy access to the player data.

Connection String for db file: “await aiosqlite.connect(DB_PATH)”

This code is used to establish an asynchronous connection to an SQLite database using the aiosqlite library.

Connection String for excel file: “workbook = load_workbook(SPREADSHEET_PATH)”

This line of code is used to access a Microsoft Excel file by using the openpyxl library.

Test Plan & Report

We will test our project by following the same process as how a normal in-house tournament game is done from the new player experience to the admin of the operations. We will all test the project as we need at least 11 people to test the project. Currently, we want to test the user experience by making the UI intuitive for users to go through, make sure that the tournament process aligns with how the current in-houses operate, and the database immediately records data after a match is finished. We will also test the capabilities of the matchmaking algorithm to see its effectiveness in a live scenario and edge cases. It will be tested on Discord through a Discord server.

Requirement	Pass	Fail	Severity
Players set role preference	O		
Players linking their Riot ID (Pulls from Riot Games API)	O		
Stores player information (Participation, Wins, MVP, Toxicity, Rank/Tier, Role Preference)	O		
Player Check-in	O		
Volunteer Button	O		
MVP voting	O		
Player Toxicity	O		

Admin permissions (open check-in, form teams, record match results, start next game)	O		
Matchmaking Algorithm	O		
Formed lane matchups minimized difference in tiers	O		
Players assigned preferred roles		X	Minor
Output formed teams to player chat	O		
Importing data to an excel sheet		X	Minor
Additional bot commands / button interactions (swap players, stats, etc.)	O		

Play testing will be conducted by Jordan and Alex as they want to make sure the UI is intuitive for users and the matchmaking algorithm is working properly. Exporting the data to the excel sheet has been troublesome as it does not properly update. In the matchmaking algorithm, it will tend to find optimized teams through non-preferred roles for players. This is an issue as prioritizing role preference is largely considered in the main KSU eSports League of Legends games.

Version Control

We utilized previous versions of the tournament bot given to us by our sponsors, which helped us have some sort of baseline when developing this software.

GitHub will be used for version control. This will ensure that the project owners can monitor our progress and can track changes from the developers. Developers can track changes to the code to ensure that it is well-documented and can be used in the main software.

Conclusion

For our first milestone deliverables, we wanted to develop our project plan and scope documentation, develop the initial version of the Discord bot with basic team and bracket formation, and a basic database setup for storing player statistics. To make these deliverables possible, we were given documentation from our sponsor about what they want in the project and previous iterations of the project that we can reuse. We also participated in the KSU eSports League of Legends games to try to match the game flow and see how our sponsor operates manually. Some things we learned in the first milestone were that the Discord library was limited in their GUI options, and the previous group's large codebase made it difficult to understand their code.

In our second milestone, we were tasked with updating the Discord bot with advanced matchmaking algorithms and AI, integrating the game developer API for real-time player data, and optimizing server resource management through asynchronous processing. We were successful in our matchmaking algorithm as it consistently matches players in the same tier against each other while minimizing tier difference if players are not in the same tier. We were also successful in integrating Riot Games's API to link player riot IDs so we can track their player rank for the matchmaking algorithm. We learned that Discord is naturally asynchronous due to its event driven architecture which helps with optimizing server resource management through asynchronous processing. Some challenges we faced were matchmaking unable to factor previous rounds of team formations, the matchmaking has trouble giving players their highest preferred role, and asynchronous processing can lead to code complexities like race conditions.

In our third milestone, we made sure that our Discord bot is fully functional and tested with easy-to-use UI, create testing reports with identified issues, and create the final project documentation. To accomplish this, we created a software test plan which details all the requirements and their severity score, created documentation in our GitHub discussing Discord bot installation and how to run the bot, and ran multiple play tests to ensure the bot runs smoothly.

Currently, our project progression has the correct implementation of Discord bot commands, the bot reads and writes data to a database, the bot uses the matchmaking algorithm to match teams based on their given weights which considers the player tier and role preference, the bot can be interacted with chat commands or buttons, and the bot displays information to chat using embeds.

However, we face problems with known bugs and potential issues. Some bugs we found are the matchmaking sometimes does not have a player's main role prioritized, the data from the .db file does not export to the Excel sheet correctly, the Games Played statistic is getting updated multiple times for some players, and the Check-In, Leave and Volunteer buttons sometimes does not get disabled when they are no longer needed. For potential issues, the matchmaking algorithm has issues balancing with the primary role preferences; for example, the algorithm has trouble prioritizing player role preferences and decides to be optimized for their secondary or tertiary roles. The Riot Games API key has problems as sometimes it cannot operate under KSU Wi-Fi, and the key needs to be regenerated every 24 hours.

Since this project is a continuation of previous iterations from past years, we want future implementations to finish implementation of Docker support, add support for other games like Valorant or Teamfight Tactics, and add a proper command to export data from the database to Microsoft Excel.

Appendix A: User Ability Guide

Admin Commands:

- `/createadminchannel`: Set a channel to an admin channel for game management
- `/checkin`: Start the check-in process for the tournament games
- `/toxicity [user_id]`: Gives a player a toxicity point

Admin Privileges:

- Start and Cancel button: allows admins to start or cancel games
- Swap button: allows the admins to swap players around
- Finalize Games button: allows the admins to confirm the games with the matchmade teams
- Start MVP Voting button: allows the admins to start MVP voting which players can pick who the MVP is in a game.
- Skip MVP button: allows admins to skip MVP voting
- Next Game / Re-Check-In button: restarts the tournament process

Player Commands:

- `/link`: Allow players to connect their Riot ID with their Discord account
- `/unlink`: Allow players to disconnect their Riot ID with their Discord account
- `/stats`: Displays player statistics with an embed.
- `/rolepreference`: Allow players to set their role preferences.

Player Privileges:

- Check In Button: Players can interact with this button to be added to a list of players.
- Leave Button: Players can interact with this button to leave the list.
- Volunteer Button: Players can interact with this button to volunteer to sit out.
- MVP voting: Players can vote who they think is the MVP of a game.

Appendix B: Gantt Chart

[illegible]