CS 30700 Design Document

TEAM NAME:
- Skip 5 Studios

Team 8 Members:
- Henry Wellman
- Andrew Thomae
- Wes Turnbull
- Nathan Simon
- Alex Bolinger

Index

Purpose:

Burnout, lethargy, and other such emotions have long since plagued people causing a general lack of productivity and feelings that could lead to depression. We seek to develop a strategic multiplayer sports game to combat these feelings and foster an experience that can be a release from the troubles that come with a person's day-to-day life. Our game will be a class-based multiplayer sports game based on the rules of handball providing a unique set of game mechanics and playstyles to provide a one-of-a-kind experience capable of eliminating feelings of burnout and general exhaustion. The players will be able to choose from a list of different animal classes like the lion class or the penguin class, which will have special abilities and different status attributes, changing how the player moves around the field and enabling different team builds depending on the preferred play style of the users.

Some similar projects include *Rocket League* and *Mario Super Strikers*. The success and popularity of these games inspired us to create something better. *Ball of the Wild* also pulls ideas from *Overwatch* and *Battlefield* with the class system. Basic gameplay rules stem from a mixture of handball and ultimate frisbee.

While similar projects have delivered excellent gameplay and mechanics, one of the major problems facing them is a lack of replayability. Game one of *Mario Super Strikers* does not feel that different from game one thousand of *Mario Super Strikers*. In addition to having different animal classes to choose from with special stats and abilities to give the player far more ways to play the game and strategize, we will also give the player access to change certain aspects of the game. Some examples of these changes would be ball size, field size, number of players, etc… This should help each game feel different from the ones that came before it, and help the player feel like there is still much to be done after game one thousand.

Functional Requirements:

1. Users Account

As a user,

- I want to be able to create an account.
- I want to be able to login to my account once I have created it.
- I want to be able to reset my password.
- I want to be able to delete my account.
- I want to be able to update my information.

2. Friends

As a player,

- I want to be able to add users as friends to my account.
- I want to be able to challenge my friends to private games.
- I want to be able to join the same lobby as my friends before a game.

3. Player Progression

As a player,

- I want an experience system with different levels to progress through.
- I want to gain experience from completing a game, with extra experience if I win.
- I want to gain experience from things like the amount of time I held the ball, or number of goals/assists scored.
- I want to be able to unlock achievements.
- I want a ranking system when I play online.
- I want a way for other players to view my ranking.
- I want to be able to unlock new characters and abilities as I get better at the game.
- (If time allows) I want to have character customization.
- (If time allows) I want to gain experience after completing a challenge in challenge mode.

4. Character Classes

As a player,

- I want different classes to have different stats and abilities
- I want to be able to play as multiple different player classes

As a developer,

- I want characters to be modular so they can be added, removed, or edited later.

5. Online Play

As a player,

- I want to be able to play online with strangers.
- I want a way to communicate with my teammates.
- (if time allows) I want my matches to be made based on player skill.

6. Match Settings

As a player,

- I want to be able to choose the time duration of my games.
- I want to be able to choose the size of the teams in my games.
- I want to be able to choose the size of the ball in my games.
- I want to be able to choose the size of the field in my games.

7. Match Features

As a player,

- I want an assist system to see who helped the player who scored a goal.
- I want an overtime match in the game if the timer runs out and the score is tied.
- I want to enjoy the soundtrack.

As a developer,

- I want a time limit on how long a player can hold the ball to encourage teamwork.
- I want shots with the ball to be more powerful if consecutive passes were recently made.
- I want there to be a limit on how often and how long a player can sprint.

8. Game Settings

As a player,

- I want to be able to change the resolution the game runs in.
- I want to be able to swap out key mappings.
- (if time allows) I want controller support.

As a developer,

- I want developer tools built into the game (frame rate, usage, etc) to monitor performance.

9. Intuitive UI Design

As a player,

- I want to be able to navigate the menus easily.
- I want to see who scored a goal in the game.
- I want to be able to easily see a scoreboard with a timer during gameplay.
- I want to be able to see my in-game abilities and see the cooldowns if I have already used them.

10. Additional Modes

As a player,

- I want to have a digestible tutorial mode to learn the rules of the game.
- I want customizable game modes with rules like when you miss a shot, you also lose a point.
- (if time allows) I want to play a game offline with AI bots.
- (if time allows) I want to have a challenge mode where I have to complete difficult tasks.
- (if time allows) I want a king of the hill mode, where a team needs to keep the ball within a limited area on the field for a determined amount of time to win.

Non-Functional Requirements:

1. Performance

As a developer,

- I want our game to run at 720p at 30 frames per second on low-performance hardware from as much as five years ago.
- I want to keep any delays between clients less than 200ms.

2. Usability

As a developer,

- I want the gameplay to be picked up easily.
- I want the menu system to be intuitive.
- I want the game to not be too overly computationally intensive.

3. Security

As a developer,

- I will use Epic Games for authentication.
- I will not allow any endpoint exceptions or exposures when connecting to our database.

4. Scalability

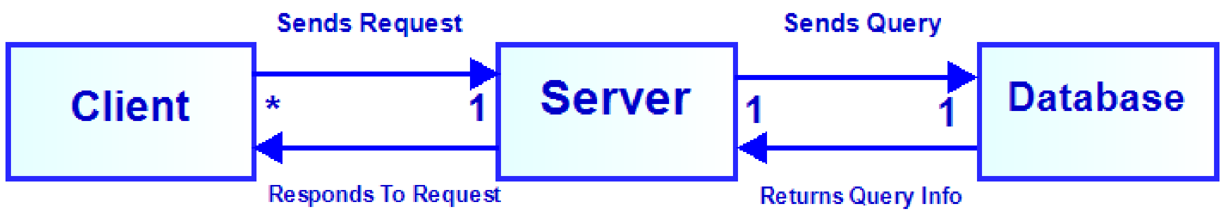
As a developer,

- I want to be able to add new characters into the game by making use of a modular class structure.
- I want to be able to find bugs easily and add more content without obfuscating the original files.

Design Outline:

High-Level Overview

This project will be a strategic, class-based, multiplayer sports video game that allows players to choose from several different game modes. The game utilizes the client-server model; the project server will be an AWS Lightsail instance handling many interactions from the different clients, which are the distributed copies of the game developed in Unreal Engine 5. The server will store the player's account information and send the data to the user when necessary, while also hosting different online sessions that occur for the multiplayer matches.



Client

- The client provides the interface for our game and is stored on the user's local machine.
- The client requests user information from the server like username and player level.
- The client receives responses from the server and uses this information to render out the game elements on the user's screen.

Server

- The server receives requests for user information from many different clients.
- The server communicates with the database to update, store, or retrieve the information requested from the client.
- The server then organizes the requested user information and sends a response back to the client.

Database

- The database contains all game information like user account information or custom types like the field or animal class.
- The database communicates with the server and returns any requested information back to the server.

Design Issues:

Functional Issues

What information do we need for signing up for an account?

1. Allow the creation of login IDs using Facebook, Google, etc.
2. No login ID creation
3. **Create a username and password that is unique to our service**

Decision: Option 3

Justification: Our reasoning behind this decision is security. We ruled out option 2 right away because without a login ID there is no way to keep track of player information like rank and achievements. We ultimately chose option 3 to keep our users' information safe. In the event of a security breach, the only information gained by the hacker will be a user's Ball of the Wild account statistics. This way we do not hold any valuable information about our users.

How do we allow players to communicate?

1. Typing into the chat box
2. **Fixed hotkey messages**
3. None

Decision: Option 2

Justification: We chose to go with option 2 because we believe that communication is crucial to collaboration among teammates in multiplayer games, but it is also very important to a player's enjoyment of their gaming experience. We do acknowledge that allowing users to communicate freely requires strict guidelines and moderation, so only allowing users to use fixed messages should help to prevent toxicity in our game.

How do we compute the rankings of players?

1. **Only take into account wins/losses**
2. Take into account goals scored and assists made
3. Player level

Decision: Option 1

Justification: Option 3, determining the ranking of a player by player level, is not an effective way to fully understand the skill a player may have. A user could play 1000 games and be worse than someone who has only played 10, yet still, be a higher rank than them. By definition, this wouldn't be a very effective ranking system. The issue with option 2, although it would be a valid way to compute a ranking, is that it doesn't weigh defensive statistics when computing. It is also much simpler in terms of the implementation of the game.

How are we matching players for online multiplayer?

1. **Have players create and choose lobbies manually.**
2. Matchmake players on a first come first served basis.
3. Create matches based on ranking or level.
4. Create matches based on the distance between players for optimal connection.

Decision: Option 1

Justification: Option 1 is the best way to have players join each other's lobbies. This is because, in terms of programming, it is the simplest solution to have a list of every game open and allow players to choose whatever game they want. Also, it makes it very easy for new players to understand because they can name a lobby and other players will be able to see it easily without having to deal with an invitation system. Option 2 would be suboptimal because that would result in a lot of different issues with skilled players being paired with worse ones. The same issue would arise if we went with option 4. Option 3 is the second best option for us, however, we are only going to be able to create the ranking system if we have time at the end of the project, which would make it problematic to join games before we finish that system.

Non-functional Issues

What database do we want to use?

1. MySQL
2. **PostgreSQL**
3. SQLite

Decision: Option 2

Justification: PostgreSQL is the best option for our game due to the ease of use of open-source software, coupled with a robust feature set that compliments our design. For example, in addition to the standard database data types, Postgres allows developers to design their own complex types. This way we can directly store types like the animal class a player is using during a match to help us align our database closer to how our game looks and feels to the players. Furthermore, the views and materialized views of Postgres abstract the original table structures for data that is frequently queried together like account username, and player level. This makes accessing and viewing relevant information that much easier.

What hosting service do we want to use?

1. **AWS**
2. Azure
3. Google Cloud

Decision: Option 1

Justification: We plan on hosting our game on an AWS Lightsail instance. This is due to several reasons. First, it is very quick and simple to get all our data imported into the server, and Amazon provides a solid free tier with 3 months of unlimited access with the same resource pool as any of their premium ec2 instances. The second reason is scalability. While we can choose any of Amazon’s server locations around the world to host our first instance, linking multiple servers together on AWS is also still very simple, with an excellent amount of documentation and an active community. This means that as our project grows, hosting and connection issues will not emerge at the same rates as they might with other virtual server providers, and any problems that do arise will be easier to pinpoint and resolve. The third is reliability. Amazon maintains all its servers for the developer and ensures that should an outage occur on the facility where your private server is stored, your information will be transferred over to a corresponding server in a different facility while AWS runs maintenance. The peace of mind knowing that the backend of our game will never go out for long periods of time is a small but very important component of why we chose AWS.

Who hosts games?

1. **Dedicated server**
2. Users host games

Decision: option 1

Justification: Our justification for this is pretty simple. We have already decided that we want to use AWS as our main hosting service. By hosting games on a dedicated server our game is less taxing on the individual user's machine. This way we can lower our recommended system requirements for running Ball of the Wild and make the game available to a wider audience.

What code repository do we use?

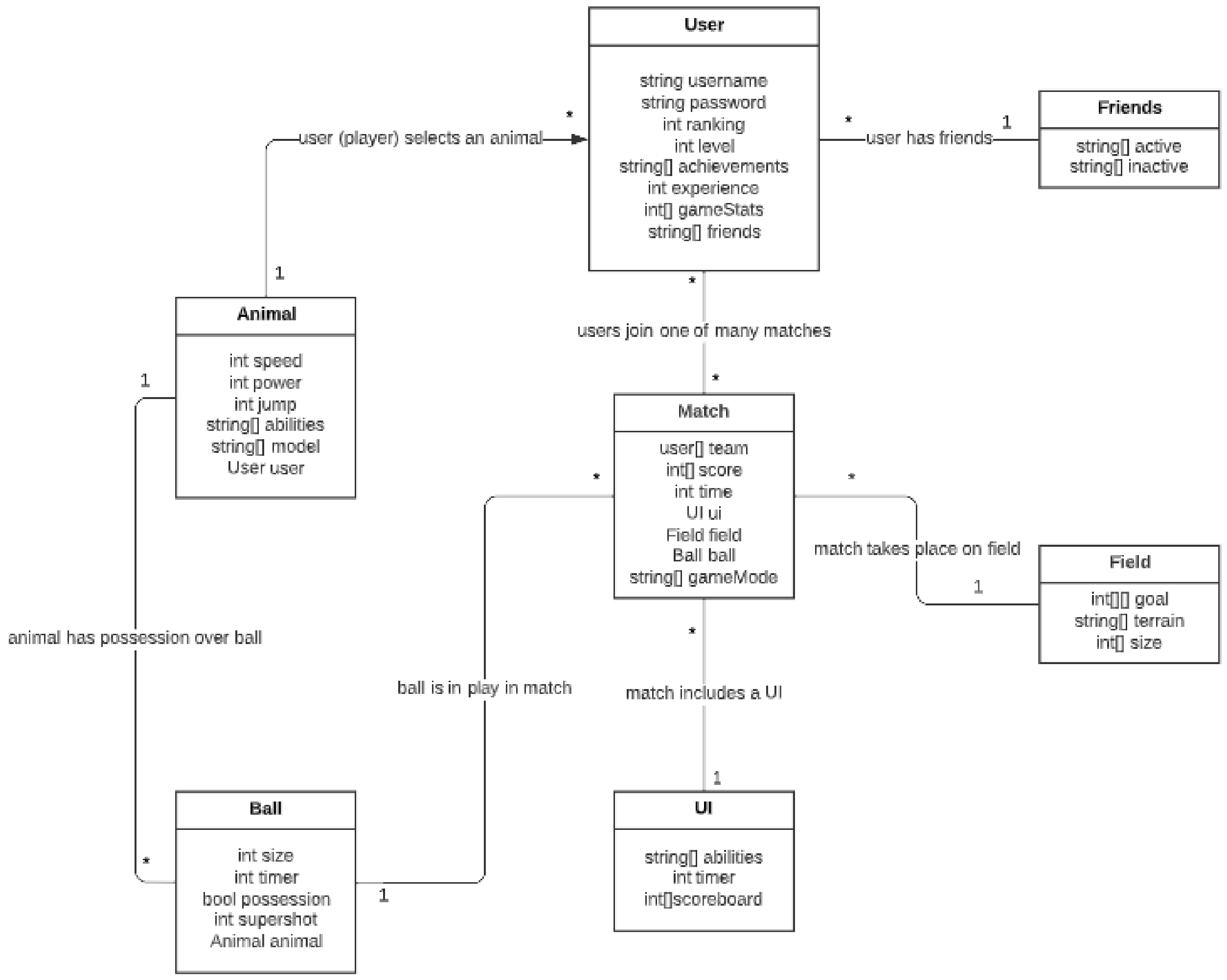
1. **GitHub**
2. Raw git server
3. Bitbucket

Decision: Option 1

Justification: Our justification for using GitHub was based on the fact that it is the repository that we have the most experience working with. GitHub is also very intuitive, widely known, widely used, and it is very effective for doing group work such as this. GitHub also supports Git LFS (Large File Storage), which is what is needed for collaboration on the unreal engine which generates large files during development. With a raw git server or bitbucket, it would require a lot more research done on the front end of things to just set up the code repository. It made the most sense to go with what we already knew.

Design Details:

Class Diagram



Class Design:

- **Animal**
  - The Animal is the character the user will control.
  - Animals can have different stats such as speed, throwing power, and jump height.
  - Animals will also have unique abilities in order to change the strategies players use when they play as different animals.
  - Each animal will have a different character model to visually change their appearance.

- **Field**
  - The goals are going to be on either end of the field, and that is where the detection for scoring will occur.
  - There will be different terrains to play on, which will affect several parts of the physics of the game. The ground could be more or less bouncy, which will directly affect the ball.
  - Field size will be scalable, which is going to be set in the menu and will allow for more players to be in a game at once without it becoming overcrowded.

- **Ball**
  - The size of the ball will have a default setting, however before the game begins there will be an option to edit this for a change in gameplay.
  - As a player grabs the ball, there will be a timer that will count down, and if the timer hits 0, the ball will jump out of the player's hands. This timer will also be changeable within a menu.
  - Possession will either be set to yes or no, depending on if a player is holding it at the time.
  - The ball can be Supershot, which will make it so that another player won't be able to catch it in the air.

- **UI**
  - The UI is a scoreboard that will appear on the top of a player's screen.
  - Shows the current score.
  - Shows a clock that shows the time remaining in the match.
  - Shows what abilities a player is able to use during the match.
  - Created at the beginning of the match.

- **User**
  - User objects are created and stored when a user creates an account.
  - Each user will have a username and password to be used for login. The username will also be used as a display name in-game.
  - Users will also have different statistics such as ranking, level, and achievements.
  - The user's ranking will be based on their win/loss record.
  - The player level will take into account games played, games won, and achievements earned.
  - Achievements will be predetermined objectives for players to complete in order to earn experience to level up.

- **Friends**

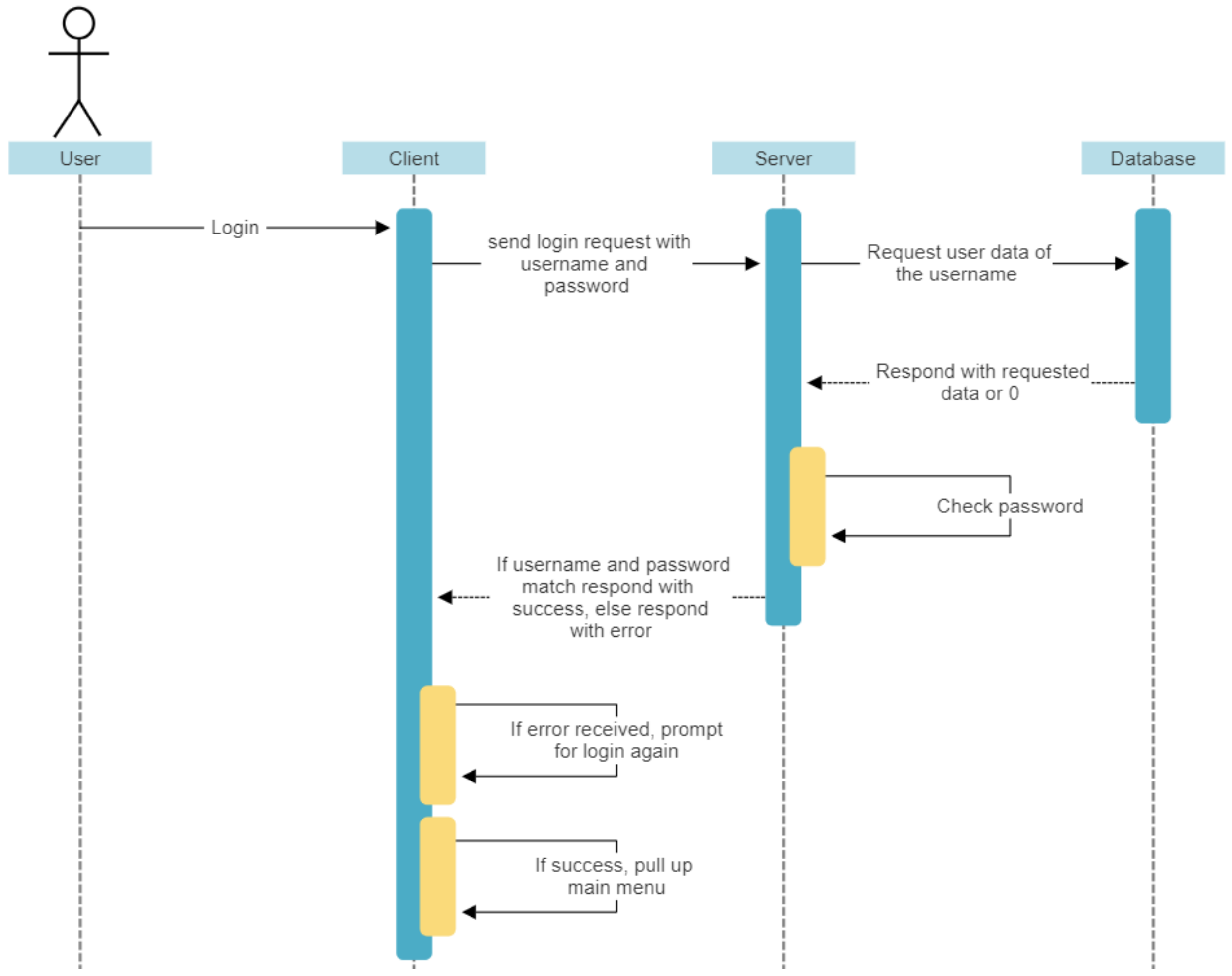
  - The friends class allows the user to grant permissions to other players, like directly challenging them to an online match from a menu.
  - If a friend is active, then they will appear available to challenge or message the user.
  - If a friend is inactive, their name will not appear in the available section and will be grayed out to show offline status. The ability to challenge or message a friend is disabled when they are offline.

- **Lobby**
  - The lobby is the screen shown to users while they are waiting to start or join a game.
  - The lobby is used to customize gameplay.
  - Users can change the length of time that the game lasts.
  - Users can pick which animal they would like to play as.
  - Users can change the size of the ball.
  - Users can change the size of the field.
  - Users can change the number of players in the match.

- **Menu**
  - The menu class is used to display relevant information to the user
  - Each menu contains tabs, which when clicked on point to a page that is loaded for the player to see.
  - Each page in a menu has different options and settings for the player to adjust.
  - Each menu will have an apply changes button if any options or settings were altered to adjust how the game looks or feels.
  - Each menu will have a cancel button which will return the player's display to the last screen that was being displayed/used.
  - Derivatives of the menu class will include the game menu, pause menu, and game matching menu.

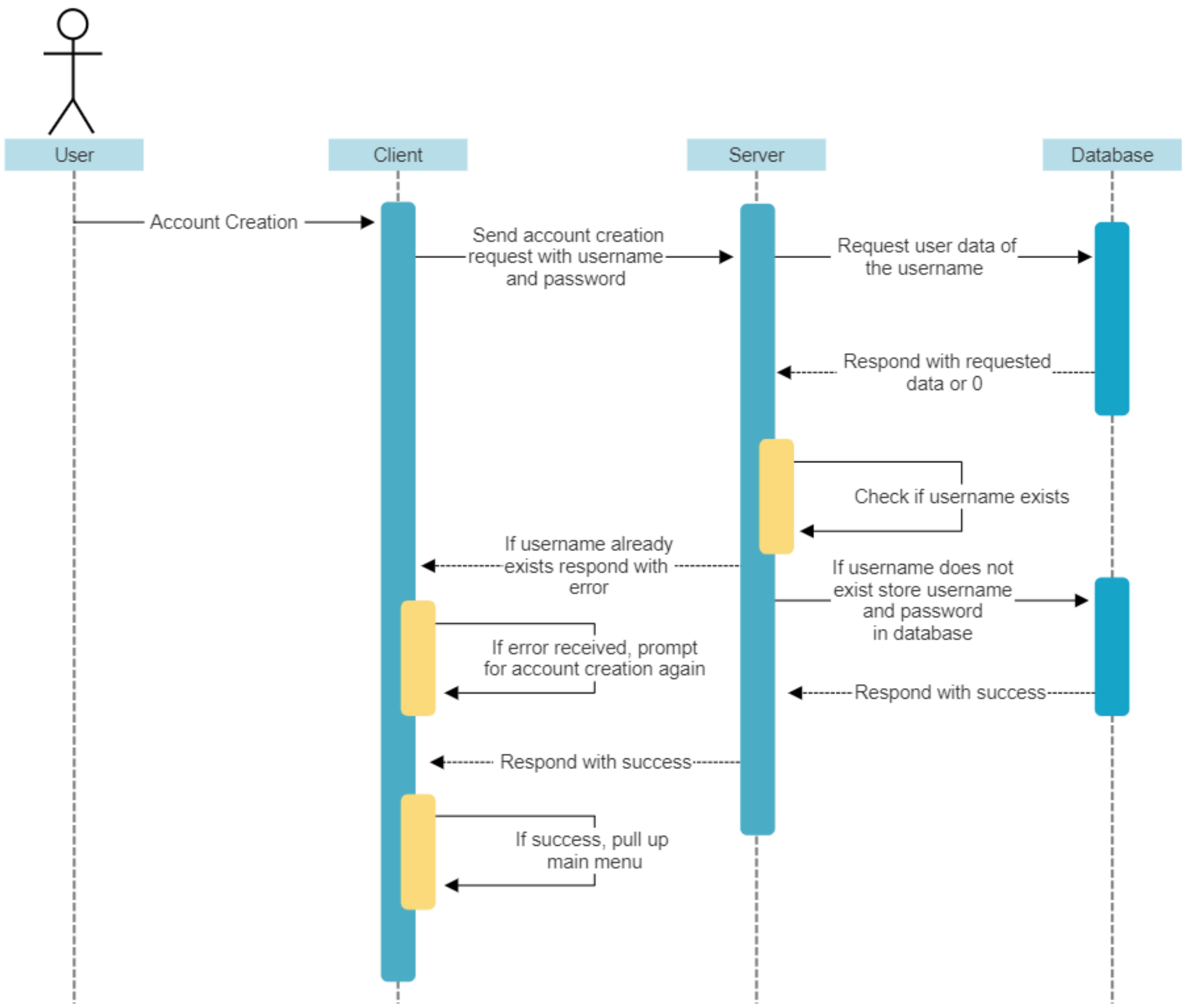
- **Match**
  - Within a match, there will be 2 user arrays that make up the teams that are competing.
  - The match will also hold information about the score and the time that will go within the UI to display to the users playing the game.
  - There are also several different game modes that could be played and adjusted within the menu to diversify a user's experience.

Sequence Diagrams:


User
Client
Server
Database
Login
send login request with username and password
Request user data of the username
Respond with requested data or 0
Check password
If username and password match respond with success, else respond with error
If error received, prompt for login again
If success, pull up main menu

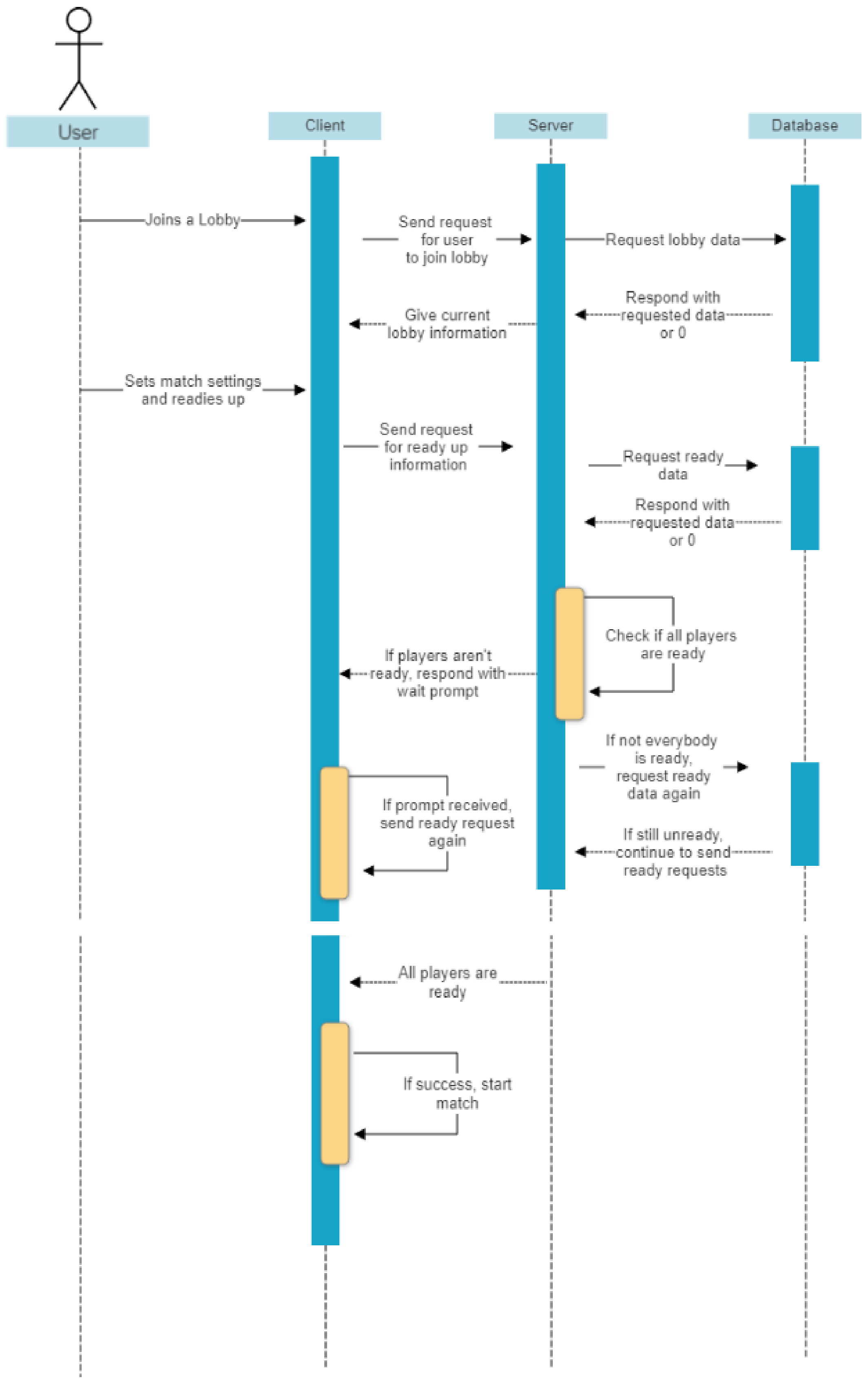
Login

When users go to login to their *Ball of the Wild* account, they will send a login request with their username and password. The server will communicate with the database to fetch the data associated with that username if it exists. If the username exists, the server will check the password against the one provided by the user for logging in. If the passwords match, then the user will be greeted with the main menu. If the username does not exist or the passwords do not match, the user will be prompted to enter their login information again.


User
Client
Server
Database
Account Creation
Send account creation request with username and password
Request user data of the username
Respond with requested data or 0
Check if username exists
If username already exists respond with error
If username does not exist store username and password in database
If error received, prompt for account creation again
Respond with success
Respond with success
If success, pull up main menu

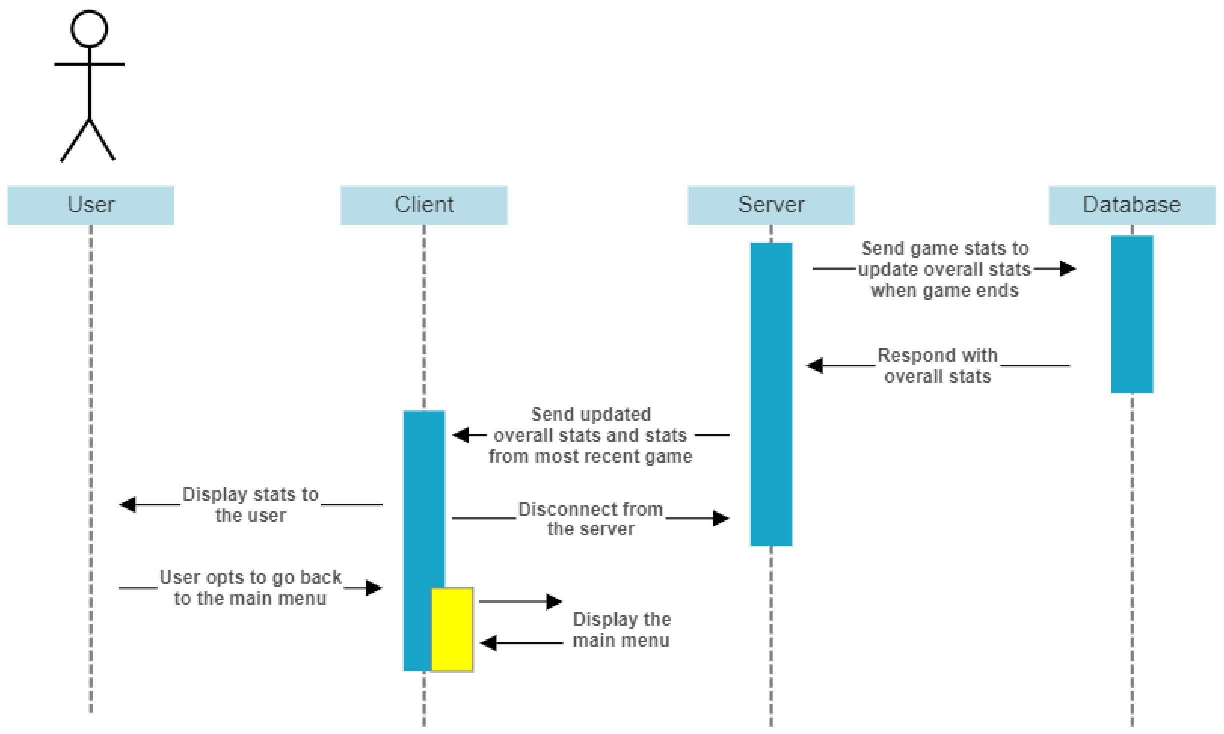
Account Creation

If a user does not already have a *Ball of the Wild* account, they will have to go through the account creation process. A user will send an account creation request with a username they pick and a password. The server will fetch data about that username from the database if it exists. If the username does exist, it will tell the user to choose a different username. If the username does not already exist, then the server will store the new username and password in the database and the user will be greeted with the main menu.


User
Client
Server
Database
Joins a Lobby
Send request for user to join lobby
Request lobby data
Respond with requested data or 0
Give current lobby information
Sets match settings and readies up
Send request for ready up information
Request ready data
Respond with requested data or 0
Check if all players are ready
If players aren't ready, respond with wait prompt
If not everybody is ready, request ready data again
If prompt received, send ready request again
If still unready, continue to send ready requests
All players are ready
If success, start match

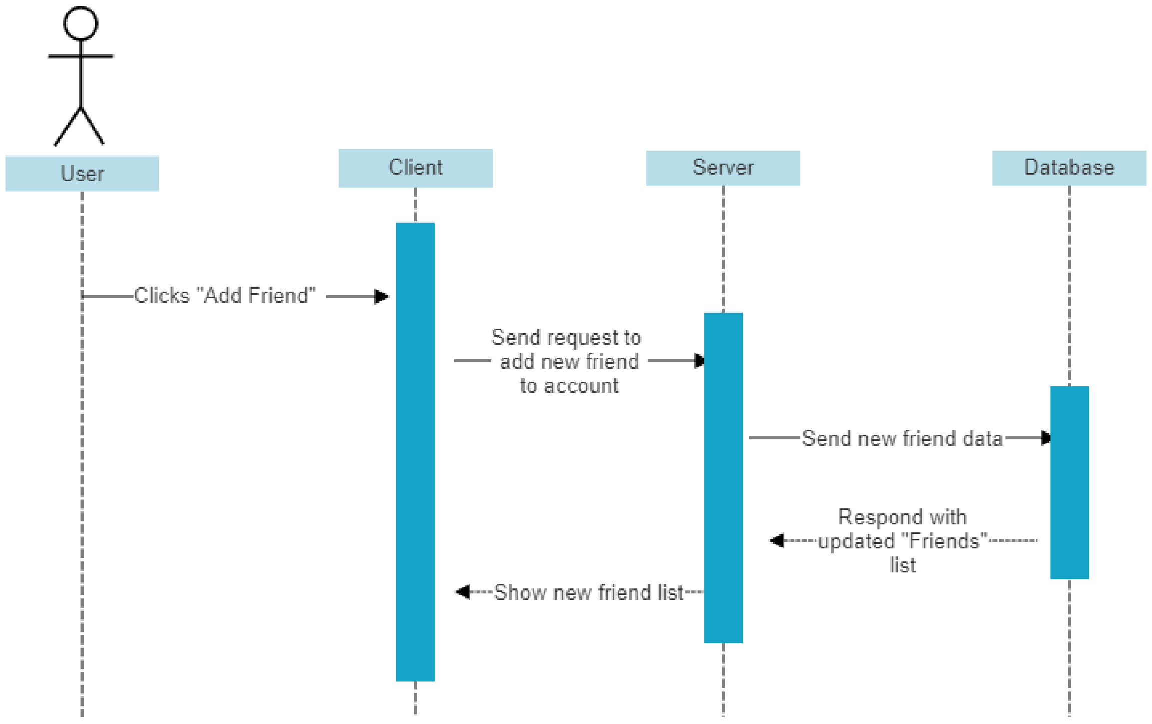
Starting A Match

The user first clicks to join a lobby, where the client sends a request to the server, which in turn, requests the lobby data from the database. If there are no errors, the database returns the lobby data, otherwise, the database will return a 0. Then, the server returns the lobby information to the client which uses this information to establish a session with the server. Within this session, the user selects their match settings, and once they have finished selecting the settings, sends a request to the server letting other players know the lobby is ready to be joined. The server then continually checks to see if all players have joined and are ready for the match to start. If not, all players are ready, the server keeps checking and the client displays the waiting prompt. Once all players are ready, the server sends a request to the database for game-ready data. When the server sends this data back to the client, the waiting prompt disappears, and the match session begins.


User
Client
Server
Database
Send game stats to update overall stats when game ends
Respond with overall stats
Send updated overall stats and stats from most recent game
Display stats to the user
Disconnect from the server
User opts to go back to the main menu
Display the main menu

Post-Game Sequence

When the match is over the server sends a request to update player information and game stats to the database. The server also requests this relevant game information from the database, so it can send this information back to the clients. The clients then display this information to the user and disconnect from the server. The user will then have the option to either continue viewing the post-game screen with their game stats or go back to the main menu. If the user chooses to go back, then the client will display the main menu screen.

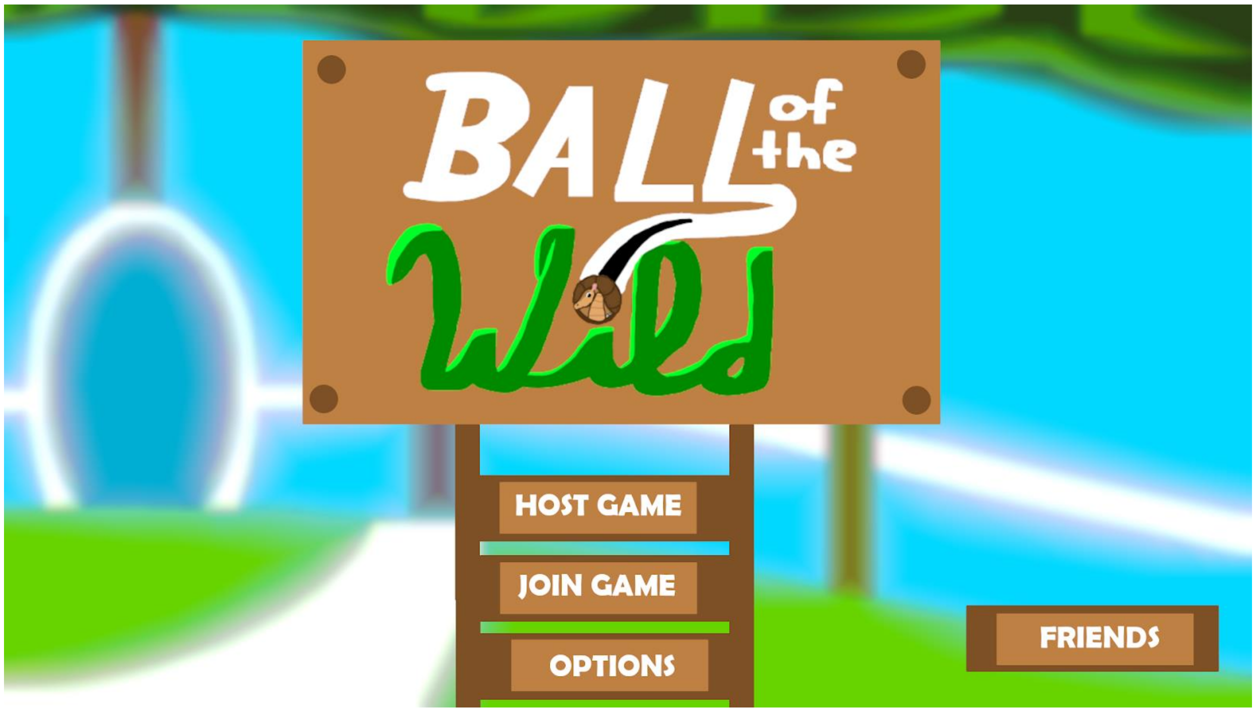


Accepting Friend Request

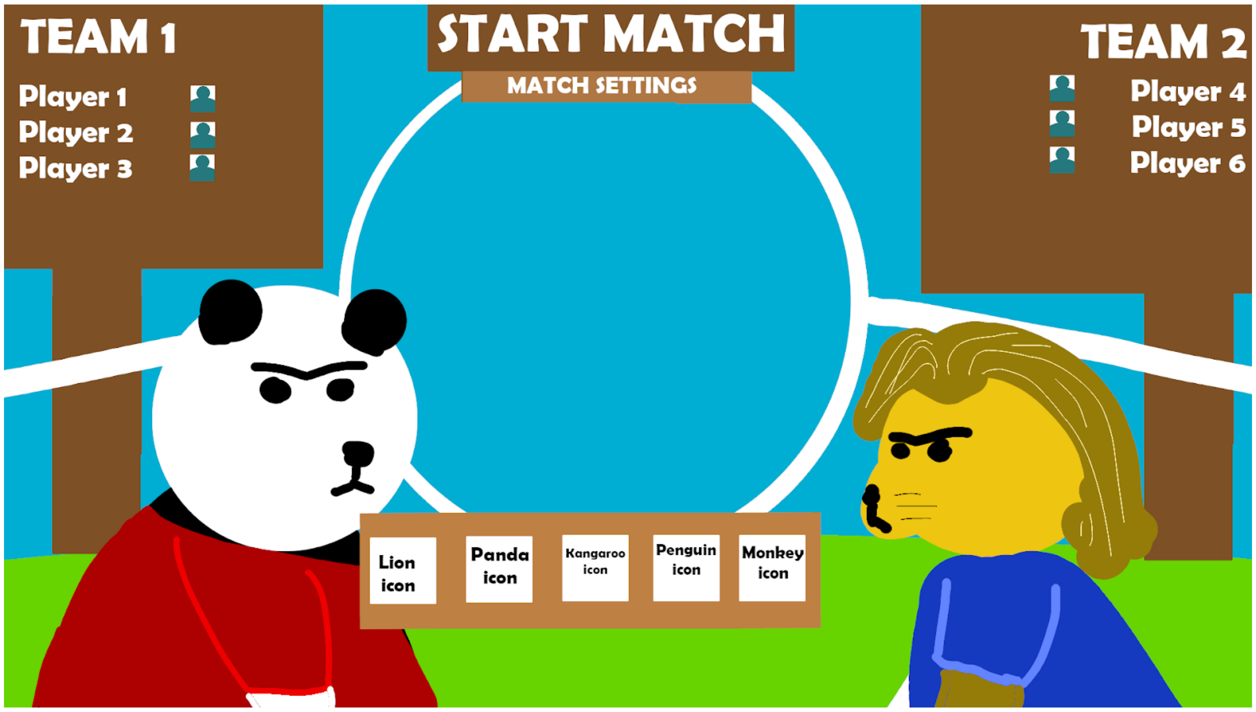
In the menu, if a player has sent you a friend request, there will be a notification on your “Friends” tab. If a user chooses to accept the request, the client will have to send the acceptance to the server, which will then update the database. The database will now include all the user’s previous friends, and also the new one. That is then sent back from the server to the client for the user to see.

UI MOCKUPS

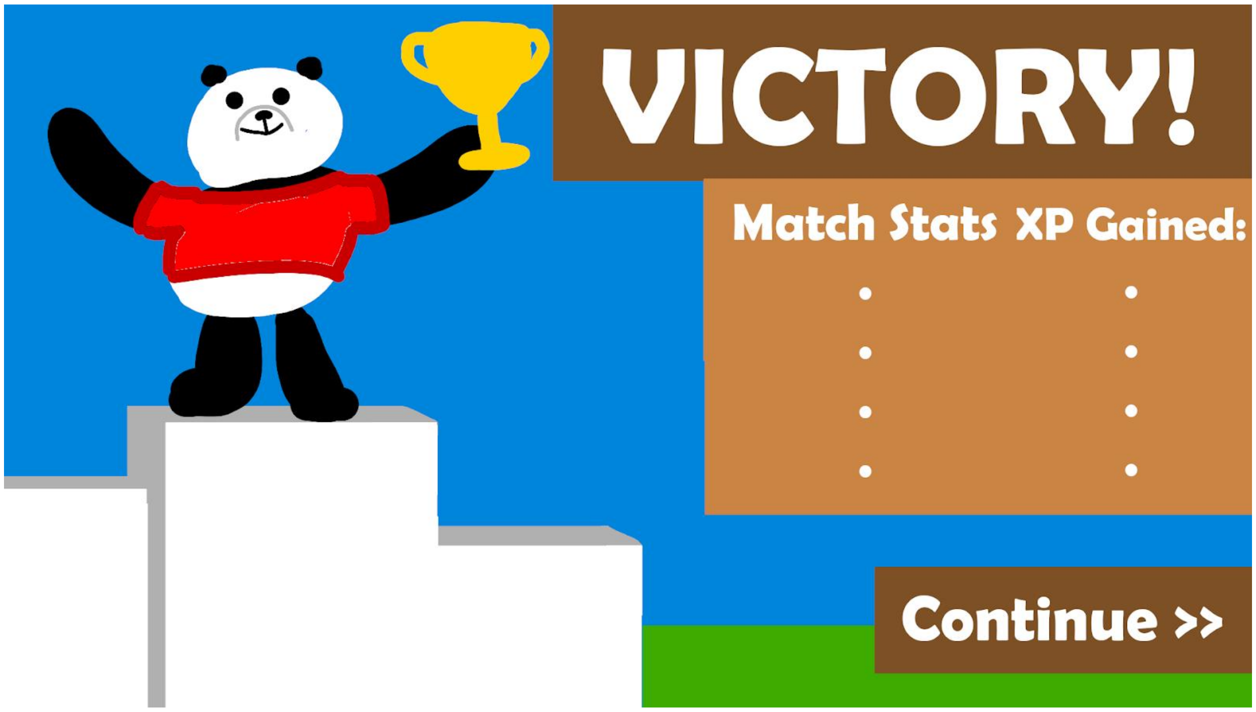
MAIN MENU



LOBBY/CHARACTER SELECTION



VICTORY SCREEN


VICTORY!
Match Stats
XP Gained:
Continue >>