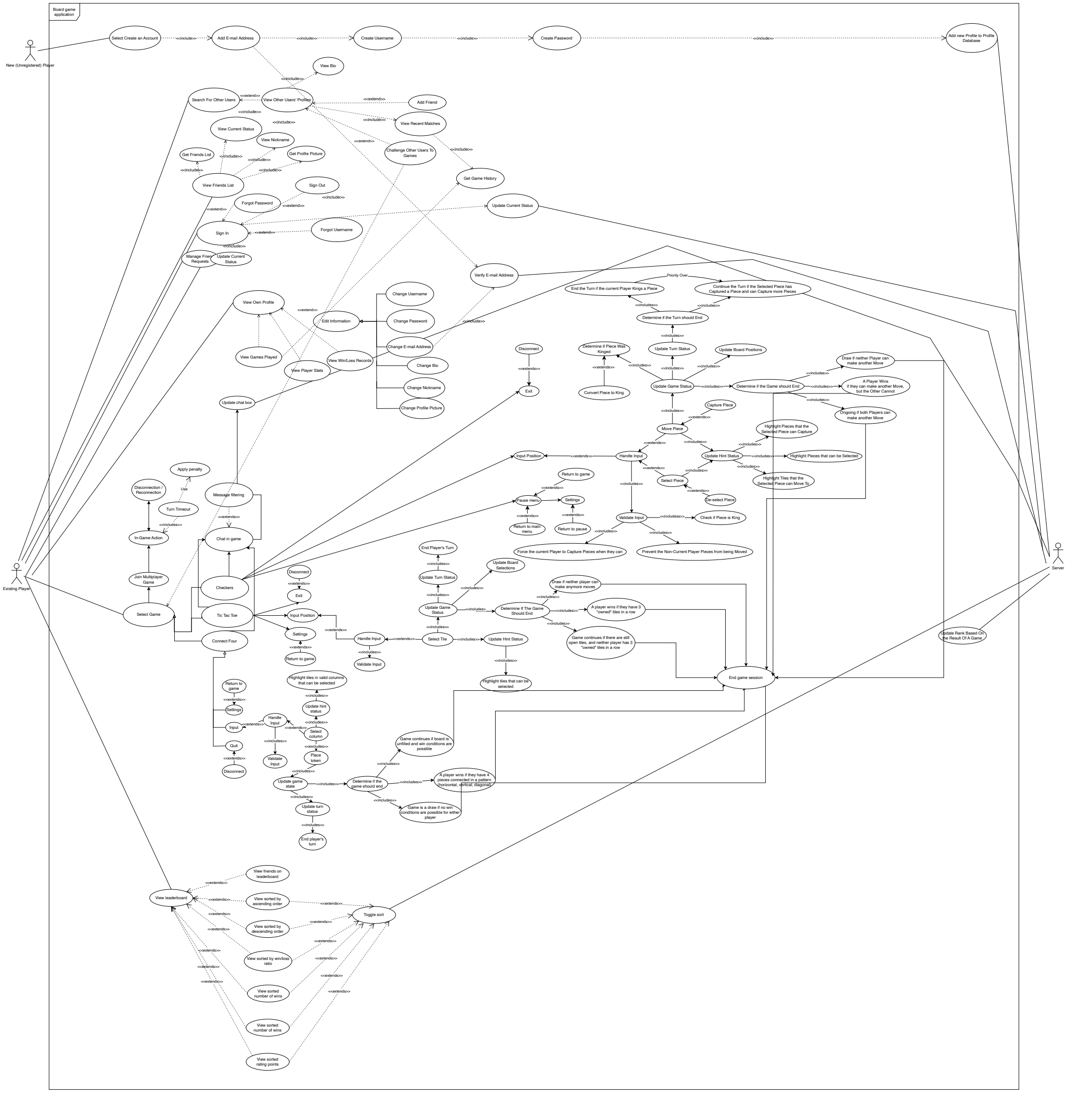
Board game application
New (Unregistered) Player
Existing Player
Server
Select Create an Account
Add E-mail Address
Create Username
Create Password
Add new Profile to Profile Database
View Bio
Search For Other Users
View Other Users' Profiles
Add Friend
View Recent Matches
Challenge Other Users To Games
Get Game History
View Current Status
View Nickname
Get Profile Picture
Get Friends List
View Friends List
Sign Out
Forgot Password
Forgot Username
Sign In
Update Current Status
Manage Friend Requests
Verify E-mail Address
View Own Profile
Edit Information
Change Username
Change Password
Change E-mail Address
Change Bio
Change Nickname
Change Profile Picture
View Games Played
View Win/Loss Records
View Player Stats
Update chat box
Disconnect
Exit
Determine if Piece Was Kinged
Convert Piece to King
Update Turn Status
Update Board Positions
Update Game Status
Determine if the Game should End
Draw if neither Player can make another Move
A Player Wins if they can make another Move, but the Other Cannot
Ongoing if both Players can make another Move
End the Turn if the current Player Kings a Piece
Priority Over
Continue the Turn if the Selected Piece has Captured a Piece and can Capture more Pieces
Determine if the Turn should End
Capture Piece
Move Piece
Highlight Pieces that the Selected Piece can Capture
Input Position
Handle Input
Update Hint Status
Highlight Pieces that can be Selected
Highlight Tiles that the Selected Piece can Move To
Select Piece
De-select Piece
Return to game
Pause menu
Settings
Return to main menu
Return to pause
Validate Input
Check if Piece is King
Force the current Player to Capture Pieces when they can
Prevent the Non-Current Player Pieces from being Moved
Apply penalty
Use
Disconnection / Reconnection
Turn Timeout
Message filtering
In-Game Action
Chat in game
Join Multiplayer Game
Select Game
Checkers
Tic Tac Toe
Connect Four
Disconnect
Exit
Input Position
Settings
Handle Input
Validate Input
Return to game
End Player's Turn
Update Turn Status
Update Board Selections
Update Game Status
Determine If The Game Should End
Draw if neither player can make anymore moves
A player wins if they have 3 "owned" tiles in a row
Game continues if there are still open tiles, and neither player has 3 "owned" tiles in a row
Select Tile
Update Hint Status
Highlight tiles that can be selected
End game session
Update Rank Based On the Result Of A Game
Highlight tiles in valid columns that can be selected
Update hint status
Select column
Place token
Update game state
Determine if the game should end
Game continues if board is unfilled and win conditions are possible
A player wins if they have 4 pieces connected in a pattern (horizontal, vertical, diagonal)
Game is a draw if no win conditions are possible for either player
Update turn status
End player's turn
Return to game
Settings
Input
Quit
Disconnect
Handle Input
Validate Input
Select column
Place token
View leaderboard
View friends on leaderboard
View sorted by ascending order
View sorted by descending order
View sorted by win/loss ratio
View sorted number of wins
View sorted number of wins
View sorted rating points
Toggle sort
<<include>>
<<extend>>
<<includes>>
<<extends>>
<<excludes>>

Use Case Descriptions

Authentication and Profile Team:

Use case: Create account

Iteration: 2

Primary Actor: New (Unregistered) Player

Goal in context: Begin the process of registering an account for a new profile to use Online Multiplayer board Game platform.

Preconditions: Program is running on the Main Menu Screen not currently logged into an existing profile.

Trigger: The new unregistered player wants to register for a new profile account to use the service.

Scenario:

1. The unregistered player clicks the Create a New Account Button.

2. The unregistered player is taken to a new Account Creation Screen.

Post conditions: The new Account Creation Screen prompts the unregistered player to fill in text boxes to input an E-mail Address, a username, and a password. There is a button at the end to confirm and complete registration.

Exceptions:

1. A Profile is already currently logged in, so Create an Account is unavailable.

2. Program malfunctions and does not properly load the new Account Creation Screen.

3. The Create a New Account Button is unresponsive when selected.

4. The Create a New Account Button does not appear.

Priority: High. The creation of Profiles is required to properly track players' game history, leaderboard rankings, player stats and win/loss ratios for appropriate match making. Profiles also allow players to find their friends' profiles to play against.

When available: Start of Iteration 3.

Frequency of use: Once for every Profile created on the platform.

Channel to actor: Using a mouse to click the button on the screen.

Secondary actors: N/A

Channel to secondary actors: N/A

Use case: Add E-mail Address

Iteration: 2

Primary Actor: New (Unregistered) Player

Goal in context: Add e-mail to associate with the account.

Preconditions: The program is on the Account Registration page.

Trigger: The unregistered player is prompted to enter an e-mail address that is not associated with any current account.

Scenario:

1. Click on the text box marked e-mail.

2. Unregistered player types in e-mail address.

3. Press Enter.

4. Once submitted, verified to be a unique e-mail address on the server and added to profile.

Post conditions: Profile has a unique e-mail associated to it.

Exceptions:

1. Invalid e-mail address.

2. E-mail address is already associated with another profile.

Priority: Medium Priority. Often a requirement of many account creation processes, which allows the company a means of communication with account holder for news and account changes. May also allow for account verification and permit access if a password is forgotten. A username is also included, which could be used as the only means of profile identification for log in purposes without an e-mail address.

When available: Start of Iteration 3.

Frequency of use: Once for every profile created on the platform.

Channel to actor: Keyboard and mouse

Secondary actors: Profile Server

Channel to secondary actors: Internet connection

Open issues: N/A

Use case: Verify E-mail Address

Iteration: 2

Primary Actor: Profile Server

Goal in context: Check if entered e-mail already is not associated with an already existing profile.

Preconditions: An e-mail has been submitted from either Create an Account page Add E-mail Address or from Update E-mail Address.

Trigger: E-mail Address is submitted to the server.

Scenario:

1. Profile Server checks HashSet profileEmailSet if it contains the requested e-mail.

2. Returns true if the HashSet does not contain the e-mail address, indicating a valid e-mail address allowing Player to proceed.

3. Returns false if the HashSet already contains the e-mail address, indicating that e-mail is not valid due to the e-mail address being already associated with another profile.

Post conditions: Returns back if the E-mail address is valid or invalid.

Exceptions:

1. An Empty string is submitted by the player.

Priority: High priority, A profile needs to have a unique identifier through the e-mail address, so an e-mail cannot be used by multiple profiles.

When available: Start of Iteration 3.

Frequency of use: Not very frequently. At least once for every created profile, with infrequent additional calls for e-mail address updates.

Channel to actor: profileEmailSet HashSet

Secondary actors: N/A

Channel to secondary actors: N/A

Open issues: N/A

Use case: Create Username

Iteration: 2

Primary Actor: New (Unregistered) Player

Goal in context: Create a username associated with the player's account.

Preconditions: The program is on the Account Registration page, and ready to add a username.

Trigger: The player wants to create an account

Scenario:

1. Click on the text box marked username.

2. Player types in the username.

3. Press Enter.

4. Once submitted, verified to be a unique username in the system and then added to player profile.

Post conditions: Profile has a unique username attached to it .

Exceptions:

1. Username is already associated with another profile.

2. Invalid username.

Priority: High Priority. The username is the main reference to the player's account and is specific to each individual account.

When available: At the start of Iteration 3.

Frequency of use: Once for every profile that is created on the platform.

Channel to actor: Keyboard and mouse

Secondary actors: Profile server

Channel to secondary actors: Internet connection

Open issues: Any length/character restrictions for a username (number of characters, no spaces, etc.)? Prevention of offensive usernames?

Use case: Create Password

Iteration: 2

Primary Actor: New (Unregistered) Player

Goal in context: Create a password associated with the player's account.

Preconditions: The program is on the Account Registration page, and ready to add a password.

Trigger: The player is prompted to create a password for the account.

Scenario:

1. Click on the text box marked password.

2. Player types in the password.

3. Press Enter.

4. Once submitted, the password is added to the player profile

Post conditions: Profile has a password attached to it.

Exceptions:

1. Invalid password.

Priority: High Priority. The password is needed for account security, along with signing in to a players account.

When available: Start of Iteration 3.

Frequency of use: Once for every profile that is created on the platform.

Channel to actor: Keyboard and mouse

Secondary actors: Profile server

Channel to secondary actors: Entered password is stored on the Profile server.

Open issues: Specific requirements for a password to be valid (at least 8 characters, a number, etc.)? Password confirmation step to ensure it is entered correctly?

Use case: Add New Profile to Profile Database

Iteration: 2

Primary Actor: New (Unregistered) Player

Goal in context: Create a new player account and add it to the Profile Database.

Preconditions: The program is on the Account Registration Page with all information entered, and ready to finalize the profile.

Trigger: The player initiates finalized registration of the account.

Scenario:

1. The player clicks complete to finalize account registration.

2. The system validates the entered information.

3. If valid, the system creates a new profile.

4. The profile is stored in the Profile Database.

5. The system confirms successful completion of profile.

Post conditions: Profile is created and added to the Profile Database

Exceptions:

1. Any invalid fields.

2. Server issues.

Priority: High Priority. Essential to creating a new profile allowing players to join the server.

When available: Start of Iteration 3.

Frequency of use: Once for every profile created on the server.

Channel to actor: Keyboard and mouse

Secondary actors: Profile server

Channel to secondary actors: Finalized account is stored on the Profile server.

Open issues: N/A

Use case: Change E-mail Address

Iteration: 2

Primary Actor: Existing Player

Goal in context: Change the e-mail address associated with the player's current account.

Preconditions: The program is on the Edit Information page, and ready to change the e-mail.

Trigger: The player initiates changing the e-mail associated with the account.

Scenario:

1. Player selects the option to change the e-mail.

2. Click on the text box marked e-mail.

3. Player types in the new e-mail address.

4. Press Enter.

5. Once submitted, verified to be a unique e-mail address on the server and added to player profile.

Post conditions: Profile is updated to have the new e-mail address associated with the account.

Exceptions:

1. Invalid e-mail address.

2. E-mail address is already associated with another profile.

Priority: Medium Priority. Not essential for profile function but may add to player experience.

When available: Start of Iteration 3.

Frequency of use: Low, only used when the player wants to change their username.

Channel to actor: Keyboard and mouse

Secondary actors: Profile server

Channel to secondary actors: New e-mail entered is searched up on the Profile server. An error is thrown if the e-mail address is associated with an already created profile.

Open issues: Limits to number of e-mail changes?

Use case: Change Username

Iteration: 2

Primary Actor: Existing Player

Goal in context: Change a username associated with the player's current account.

Preconditions: The program is on the Edit Information page, and ready to change the username.

Trigger: The player initiates changing the username for the account.

Scenario:

1. Player selects the option to change the username.

2. Click on the text box marked username.

3. Player types in the new username.

4. Press Enter.

5. Once submitted, verified to be a unique username in the system and then added to the player profile to replace the old username.

Post conditions: Profile is updated to have the new unique username attached to it.

Exceptions:

1. Username is already associated with another profile.

2. Invalid username.

Priority: Medium Priority. Not essential for profile function but may add to player experience.

When available: Start of Iteration 3.

Frequency of use: Low, only used when the player wants to change their username.

Channel to actor: Keyboard and mouse

Secondary actors: Profile server

Channel to secondary actors: New username is searched on the Profile server and stored if valid. An error is thrown if the username is associated with an already existing profile.

Open issues: Limits to amount of username changes? Any restrictions to the username itself?

Use case: Change Password

Iteration: 2

Primary Actor: Existing Player

Goal in context: Change the password associated with the player's current account.

Preconditions: The program is on the Edit Information page, and ready to change the password.

Trigger: The player initiates changing the password for the account.

Scenario:

1. Player selects the option to change the password.

2. Click on the text box marked password.

3. Player types in the new password.

4. Press Enter.

5. Once submitted, verified to be a unique password in the system and then added to player profile to replace the old password.

Post conditions: Profile is updated to have the new password attached to it.

Exceptions:

1. Invalid password.

Priority: Medium Priority. Not essential for profile function but may add to player experience.

When available: Start of Iteration 3.

Frequency of use: Low, only used when player wants to change their password.

Channel to actor: Keyboard and mouse

Secondary actors: Profile server

Channel to secondary actors: New entered password is stored on the Profile server.

Open issues: Password confirmation step to ensure correct entry? Restrictions to the password itself?

Use case: Forgot Username

Iteration: 3

Primary Actor: Existing Player

Goal in context: To recover the username associated with a players account.

Preconditions: The player has an account with an email attached to verify account owner.

Trigger: The player selects the forgot username option on the login screen.

Scenario:

1. Player clicks the Forgot Username button on the login page.

2. Clicks on the text box marked e-mail.

2. The player enters the e-mail associated with the account.

3. Verifies email associated with the account.

4. If valid, displays the username associated with the e-mail.

Post conditions: Player recovers the username associated with the account and is able to complete login.

Exceptions:

1. Invalid e-mail.

2. No account matching the e-mail.

Priority: High Priority. Essential for account recovery.

When available: End of Iteration 3.

Frequency of use: Low, only used when player forgets their username.

Channel to actor: Keyboard and mouse

Secondary actors: Profile server

Channel to secondary actors: Username is found on the profile server based on the accounts e-mail address.

Open issues: Lock out after too many failed entries? Security issues?

Use case: Forgot Password

Iteration: 3

Primary Actor: Existing Player

Goal in context: To reset the password associated with a players account when attempting to log in.

Preconditions: The player has an existing account with an email attached for verification.

Trigger: The player selects the forgot password option on the login screen.

Scenario:

1. Player clicks the Forgot Password button on the login page.

2. Clicks on the text box marked e-mail.

3. Player enters the e-mail associated with the account.

4. Clicks on the text box marked username.

5. Player enters the username associated with the account.

6. Verifies the e-mail and username entered.

7. If valid, the system prompts the player to set a new password.

8. Click on the text box marked password.

9. Player types in the new password.

10. Press Enter.

11. Once submitted, verified to be a unique password in the system and then added to player profile to replace the old password.

Post conditions: Player's password associated with the account is changed, and player is able to log in.

Exceptions:

1. Invalid e-mail or username.

2. No account matching the entries is found.

3. Invalid new password.

Priority: High Priority. Essential for account recovery.

When available: End of Iteration 3.

Frequency of use: Low, only used when player forgets their password.

Channel to actor: Keyboard and mouse

Secondary actors: Profile server

Channel to secondary actors: E-mail and username are stored on the Profile server and are used for account verification.

Open issues: Security issues?

Use case: Edit Information

Iteration: 2

Primary Actor: Existing Player

Goal in context: To allow player to edit their information associated with their account.

Preconditions: Player has an existing account and is on their profile.

Trigger: The player initiates edits by entering the Edit Information page.

Scenario:

1. Player selects Edit Information.

2. System displays information in editable fields.

3. Player makes changes to desired information.

4. If valid entries, the system updates the information.

Post conditions: The players information is updated on their profile and in the system.

Exceptions:

1. Invalid entries

Priority: Medium Priority. Not essential for profile function but may add to player experience.

When available: Start of Iteration 3.

Frequency of use: Low, only when player wants to change their information.

Channel to actor: Keyboard and mouse

Secondary actors: Profile server

Channel to secondary actors: Information is stored on the Profile server and is updated when changes are made.

Open issues: Verification before changes?

Use case: Search for Other Users

Iteration: 2

Primary Actor: Player

Goal in context: Search username of another player to view their profile.

Preconditions: Player is logged in.

Trigger: The player wants to find a profile of another player to view current status, rank, and recent matches.

Scenario:

1. Player clicks on Profile Search box.

2. Player types in username using a keyboard.

3. Player clicks enter.

4. Profile database is searched and returns usernames that contain the searched term. The highest results are profiles that most match the search term.

Post conditions: Usernames of profiles that match search input are displayed. The player may select one to view the profile.

Exceptions:

1. Error searching the database not terminating or returning any results.

2. No results match the search term.

Priority: High priority. The platform is to play multiplayer games, so it is important that players can find their friends' profiles to play against them and see profile ranks.

When available: Start of Iteration 3.

Frequency of use: Medium frequency. Players will search for friends, but then will be more likely to use the Friends Lists regularly view friend’s profiles more quickly.

Channel to actor: Keyboard

Secondary actors: Profile database server

Channel to secondary actors: Search algorithm through all profile usernames in the database.

Open issues: N/A

Use case: View Current Status

Iteration: 2

Primary Actor: Player

Goal in context: View the current status of a searched player.

Preconditions: The player has searched for and selected a Profile.

Trigger: The player wants to see if the player is online and available to play.

Scenario:

1. Player selects view status.

2. Server checks current profile isOnline variable status

3. If false and therefore not logged in, profile is displayed as offline.

4. If true and therefore logged in, profile is displayed as online.

5. If logged in, Server checks current profile currentGame variable status.

6. If null, player is not currently in a game, profile is displayed as available.

7. If variable contains a string of a game name, then profile displays the current game name.

Post conditions: Current status is displayed on Profile, showing if online, offline, or what game the profile is currently playing.

Exceptions:

1. isOnline and currentGame variables display not up-to-date information

Priority: High priority, because players need to know if a player is currently available to be able to invite them to play a game together.

When available: Start of Iteration 3.

Frequency of use: Constantly updated when on the Friends List page or on a Profile Page.

Channel to actor: Displayed on a Profile Page Screen, or on the Friends List Page.

Secondary actors: Server

Channel to secondary actors: Obtains profile of selected profile to obtain variable contents.

Open issues: N/A

Use case: View Bio

Iteration: 2

Primary Actor: Existing Player

Goal in context: View the bio of another player.

Preconditions: The player has searched for and selected another player's profile.

Trigger: The player wants to view the bio of another player

Scenario:

1. Player clicks into another player's profile.

2. The other player's profile loads up and their name and part of their bio is immediately visible.

3. If there is more to a player's bio than can fit in the immediately presented section of the profile, the user can click "show more" to see the other user's full bio.

Post conditions: The player who wanted to see the bio of the other player is able to see the other player's bio and has extended it if they wanted to.

Exceptions:

1. The other player has no bio set.

Priority: High, because the bio is a fundamental part of player profiles. It allows users to express themselves and makes the site more social.

When available: Start of Iteration 3.

Frequency of use: Medium, players won't always be searching for other players but will occasionally want to look up other players to see how they're performing or to add them as friends.

Channel to actor: Displayed on another player's profile page,

Secondary actors: Profile database server

Channel to secondary actors: Stored bio information for each user server-side

Open issues: N/A

Use case: Change Bio

Iteration: 2

Primary Actor: Existing Player

Goal in context: Change your own profile's bio.

Preconditions: The player has clicked into and is on their own profile page.

Trigger: The player wants to change their own bio.

Scenario:

1. Player clicks into their own profile.

2. Their profile loads up and their name and part of their bio is immediately visible.

3. There is a button that says "Edit profile" somewhere on their profile page.

4. They click that button and are brought to an edit profile page where they can edit their bio's information.

5. The user clicks a "save" button to save their changes.

6. The user is kicked back to their profile page where the changes have been applied.

Post conditions: The player has edited their bio and saved the changes. The changes are reflected on their profile.

Exceptions:

1. The player has no bio set.

Priority: High, because the bio is a fundamental part of player profiles. It allows users to express themselves and makes the site more social.

When available: Start of Iteration 3.

Frequency of use: Low, players will not often want to change their bio after they've set it for the first time, but when things change in their life or their interests shift, they might want to reflect that on their profile page.

Channel to actor: Displayed on a player's own profile page.

Secondary actors: Profile database server

Channel to secondary actors: Stored bio information for each user server-side

Open issues: N/A

Use case: View Nickname

Iteration: 2

Primary Actor: Existing Player

Goal in context: View the nickname of another player.

Preconditions: The player has searched for and selected another player's profile or is looking at their friends list.

Trigger: The player wants to view the nickname of another player

Scenario:

1. Player clicks into another player's profile or hovers over the user in their friends list.

2. The other player's profile loads up and their name and nickname in brackets is immediately visible if they clicked into their profile, or the other users' nickname is shown in brackets if the player hovers over the other player on their friends list.

Post conditions: The player who wanted to see the nickname of the other player is able to see the other player's nickname.

Exceptions:

1. The other player has no nickname set.

Priority: Medium, because nicknames, while a fundamental part of player profiles, are not critical to the social aspects of the website.

When available: Start of Iteration 3.

Frequency of use: Medium, players won't always be searching for other players but will occasionally want to look up other players to see how they're performing or to add them as friends.

Channel to actor: Displayed on another player's profile page, or on hover on their friends list.

Secondary actors: Profile database server

Channel to secondary actors: Stored nickname information for each user server-side

Open issues: N/A

Use case: Change Nickname

Iteration: 2

Primary Actor: Existing Player

Goal in context: Change your own profile's nickname.

Preconditions: The player has clicked into and is on their own profile page.

Trigger: The player wants to change their own nickname.

Scenario:

1. Player clicks into their own profile.

2. Their profile loads up and their name and nickname is immediately visible.

3. There is a button that says "Edit profile" somewhere on their profile page.

4. They click that button and are brought to an edit profile page where they can edit their nickname.

5. The user clicks a "save" button to save their changes.

6. The user is kicked back to their profile page where the changes have been applied.

Post conditions: The player has edited their nickname and saved the changes. The changes are reflected on their profile.

Exceptions:

1. The player has no nickname set.

Priority: Medium, because nicknames, while a fundamental part of player profiles, are not critical to the social aspects of the website.

When available: Start of Iteration 3.

Frequency of use: Low, players will not often want to change their bio after they've set it for the first time, but when things change in their life or their interests shift, they might want to reflect that on their profile page.

Channel to actor: Displayed on a player's own profile page.

Secondary actors: Profile database server

Channel to secondary actors: Stored bio information for each user server-side.

Open issues: N/A

Use case: Add Friend.

Iteration: 2

Primary Actor: Existing Player.

Goal in context: Add another user as your friend.

Preconditions: The player is on another users' profile page.

Trigger: The player wants to add another user as a friend.

Scenario:

1. Player clicks into another users' profile.

2. Their profile loads up and their name and there is an "add friend" button visible.

3. The player clicks the add friend button.

4. A friend request is sent to the other player, who can either accept or deny the request.

Post conditions: The initiating player has sent a friend request to the other player who can choose to accept or deny the request.

Exceptions:

1. The player is already the initiating player's friend.

Priority: High, the friend system is an integral part of the social experience of the website.

When available: Start of Iteration 3.

Frequency of use: Medium, players will want to add friends on a regular basis.

Channel to actor: Displayed on another player's profile page.

Secondary actors: Other players, profile database server.

Channel to secondary actors: Players' friends information stored server side, requests to other players sent to a notifications section.

Open issues: N/A

Use case: Manage Friend Requests

Iteration: 2

Primary Actor: Player

Goal in context: Accept or Reject friend requests to complete adding a profile to the two players friends lists.

Preconditions: A friend request has been sent to the player's profile by another player.

Trigger: Player opens Friend Request menu to view pending friend requests.

Scenario:

1. Player clicks Friend Request menu.

2. System obtains from Profile Database for friendRequest ArrayList that contains profiles that have made unhandled friend requests.

3. Player selects to "Accept" or "Reject" each friend request.

4. Once selected, the profile is removed from the profile's friendRequest ArrayList.

5. If accepted, the player's profile is added to the requesting profile's FriendsList List and the requesting profile is added to the player's profile FriendsList List.

Post conditions: The player's and requesting player's FriendsLists are modified to reflect the decision of the player on the received friend request.

Exceptions:

1. The requesting profile's profile has been deleted in between sending and the player reviewing friend request.

Priority: Medium priority. It is important for safety that players have the ability to decide for someone to have access to view their profile activity and send game invites. Due to time constraints, it would save development time to automatically add profiles to the FriendsList automatically after selecting to add a profile as a friend.

When available: Start of Iteration 3.

Frequency of use: Medium. It will be used as frequently as the Add Friend use case.

Channel to actor: Player selects viewing Friend requests and accepting or rejecting friend requests using the mouse to click on GUI buttons.

Secondary actors: Profile Database.

Channel to secondary actors: Method calls to server.

Open issues: It may be good to have a notification pop up on a profile when a friend request is sent if time allows.

Use case: Get Friends List.

Iteration: 2

Primary Actor: Player

Goal in context: Obtain the player profile's FriendsList information to display the FriendsList with current Status information.

Preconditions: Player is logged in.

Trigger: Player selects the Friends Tab on the Main Menu.

Scenario:

1. Method call to obtain the Friends List associated to the player's profile.

2. Display all Profiles contained within the list.

3. The current status of each profile will be presented alongside the username or nickname (if provided). Profiles may be selected to view more info.

Post conditions: The player can now view current information of profiles within their Friends List on the Friends List Menu.

Exceptions:

1. An empty friends list will prompt a message to explain to the player how to find profiles and add them as friends.

Priority: High Priority. It is important for players' ease of use to find friend’s profiles in one list to quickly initiate games.

When available: Start of Iteration 3.

Frequency of use: Often. Players will want to often easily access the Friends List to check if their friends are online and available to play a game with them.

Channel to actor: Clicking menu tabs with a mouse.

Secondary actors: Profile Database

Channel to secondary actors: Method calls to the server.

Open issues: N/A

Use case: Obtain Game History

Iteration: 2

Primary Actor: Player

Goal in context: Obtain the Game History of the requested profile to either display the Most Recent Games, or the Full Game History.

Preconditions: The profile exists and has played at least one game.

Trigger: The Most Recent Games of another player's profile, or the Full Game History of the player has been selected to be viewed by the player.

Scenario:

1. Player makes the request to view another player's profile or their own profile.

2. The appropriate information is displayed

Post conditions: A List of Game History is obtained to be viewed by the player.

Exceptions:

1. Will return an exception if empty for information to be displayed that the profile has not played any games yet.

Priority: High priority. Not only does this provide information for assignment requirements, it may also be useful for calculating rankings for implementing a proper matchmaking system.

When available: Start of Iteration 3.

Frequency of use: Used whenever viewing another player's profile, or viewing the personal profile of the player.

Channel to actor: Interactions with the GUI system using the mouse.

Secondary actors: Profile Database.

Channel to secondary actors: Accessing HashMap of Profiles using the .get() method.

Open issues: N/A

Use case: View Player Stats.

Iteration: 2

Primary Actor: Existing Player.

Goal in context: Check your own statistics (win loss ratio, games played, etc.).

Preconditions: The player has clicked into and is on their own profile page.

Trigger: The player wants to check their stats.

Scenario:

1. Player clicks into their own profile.

2. Their profile loads up and their name and some of their stats are immediately visible.

3. There is a button that says "See more" beneath their already displayed stats that allows players to see more detailed statistics.

4. If they click that button, they are brought to a statistics page where they can see detailed statistics for each game on the site.

Post conditions: The player has viewed their statistics.

Exceptions:

1. The player has no statistics (no games played).

Priority: Medium, because statistics, while a fun feature of player profiles, are not required to create a minimum viable product. Will be implemented later on in development.

When available: Start of Iteration 3.

Frequency of use: Low, players will not often want to check their statistics. It's only something that dedicated players would care about.

Channel to actor: Displayed on a player's own profile page.

Secondary actors: Profile database server.

Channel to secondary actors: Stores statistics for each user server-side.

Open issues: N/A

Use case: View Win/Loss Records.

Iteration: 2

Primary Actor: Existing Player.

Goal in context: Check your own win/loss records.

Preconditions: The player has clicked into and is on their own profile page.

Trigger: The player wants to check their win/loss record.

Scenario:

1. Player clicks into their own profile.

2. Their profile loads up. Their win/loss records for each game are immediately visible on the right side of the screen.

Post conditions: The player has viewed their win loss record.

Exceptions:

1. The player has no win/loss record (no games played).

Priority: High, because win loss records, should be relatively simple to implement and is an important social aspect relevant for the leaderboard system.

When available: Start of Iteration 3.

Frequency of use: Medium, players will see their win/loss records whenever they're on their profile page.

Channel to actor: Displayed on a player's own profile page.

Secondary actors: Profile database server

Channel to secondary actors: Stores win loss records for each user server-side

Open issues: N/A

Use case: View Recent Matches

Iteration: 2

Primary Actor: Existing Player

Goal in context: Check the recent match history of other players.

Preconditions: The player has clicked into and is on another player's profile page.

Trigger: The player wants to check another players' recent matches.

Scenario:

1. Player clicks into another user's profile.

2. Another player's profile loads up. Their recent match history is immediately visible on the lower half of the screen. The player that looked for it can scroll to see more games.

Post conditions: The player has viewed another player's match history.

Exceptions:

1. The player being viewed has not played any games.

Priority: High, because match history is an important feature relevant to the leaderboard system and important for maintaining a social vibe on the site.

When available: Start of Iteration 3.

Frequency of use: Low, players will see another user's match history whenever they search for them which won't be often.

Channel to actor: Displayed on another player's profile page.

Secondary actors: Profile database server

Channel to secondary actors: Stores match history for each user server-side

Open issues: N/A

Use case: Update Rank.

Iteration: 2

Primary Actor: Server.

Goal in context: Update the rank of a player that wins or loses a game.

Preconditions: A player has won or lost a game.

Trigger: The player's rank needs to be updated to reflect the change in their win loss record.

Scenario:

1. A player finishes a game and either wins or loses.

2. The server either increases or decreases the player's elo based on if they won or loss and the elo of their opponent dictates how much elo they lose or gain.

Post conditions: The player's rank has been updated.

Exceptions:

1. The player does not have a rank.

Priority: High, because rank is a critical aspect of the matchmaking system.

When available: Start of Iteration 3.

Frequency of use: Frequent, once per game.

Channel to actor: Result sent to server, change reflected client side.

Secondary actors: Player.

Channel to secondary actors: Player profile.

Open issues: N/A

Use case: Update Player Win/Loss Records Based On The Result Of A Game.

Iteration: 2

Primary Actor: Server.

Goal in context: Update the Win/Loss Records to reflect the results of a game match.

Preconditions: Player is logged in.

Trigger: Player completes a game match, either winning or losing.

Scenario:

1. Server automatically updates the win loss record of the player's profile for the category of the game played.

2. updateWinLossRecord() is called with a boolean input True if game was won or False if game was lost.

3. The Win/Loss ratio totals all wins and loses the player has obtained over the entire tracked Game History of the profile.

4. The Win/Loss ratio updates the Win/Loss Ratio variable of the player's profile.

Post conditions: The Win/Loss ratio now reflects the change of the ratio from the results of the completed game.

Exceptions:

1. Game ends in a tie, so Win/Loss ratio is not changed.

Priority: High Priority. The Win/Loss Record is important for ranking players and displaying leaderboards. Such data is imperative to formulate balanced match making.

When available: Start of Iteration 3.

Frequency of use: Frequent. Occurs after every game match.

Channel to actor: Method calls to server to update profile Win/Loss Record variables.

Secondary actors: Player.

Channel to secondary actors: Player completes a game.

Open issues: N/A

Use case: Challenge Users to Games.

Iteration: 2

Primary Actor: Existing Player.

Goal in context: Allow player to challenge another player to a game.

Preconditions: Player and player to be challenged must both have active accounts.

Trigger: Player wants to initiate a challenge to another player.

Scenario:

1. Player selects opponent from available players.

2. Player sends a challenge request.

3. The challenged player is notified of the request.

4. If accepted the game is initiated.

5. If declined the player is notified of the declining.

Post conditions: A game starts if the challenge is accepted, or the player is notified of rejection if declined.

Exceptions:

1. Challenged player does not respond.

2. Player cancels the challenge.

Priority: High Priority. Allows players to challenge other players and complete matches.

When available: Start of Iteration 3.

Frequency of use: Medium, used whenever a player wants to challenge a specific player in a game.

Channel to actor: Keyboard and mouse.

Secondary actors: Challenged player.

Channel to secondary actors: Server.

Open issues: Only able to challenge friends? Restrictions based on rank?

Use case: Get Profile Picture

Iteration: 2

Primary Actor: Profile Server

Goal in context: Obtain the image file associated with the specified profile.

Preconditions: A profile picture image has been set for the profile.

Trigger: Profile image is to be displayed on certain screens such as a profile page, or the Friends list.

Scenario:

1. Server searches Profile Database for username of the profile.

2. From the profile object, calls getProfilePic() on the profilePic variable to retrieve the BufferedImage datatype to obtain the image file to display in the GUI.

Post conditions: Image file associated to the profile has been retrieved to be displayed.

Exceptions:

1. An image has not been set for the profile, so a placeholder image is used.

Priority: Low. It is helpful for players to both quickly recognize friend profiles. It also some players with have an more enjoyable experience being able to customize their profile.

When available: Start of Iteration 3.

Frequency of use: High. Multiple menus may show profile images, requiring retrieval from profile database.

Channel to actor: Method calls to database info on the server.

Secondary actors: N/A

Channel to secondary actors: N/A

Open issues: N/A

Use case: Sign Out.

Iteration: 3

Primary Actor: Existing Player.

Goal in context: Allow the player to sign out of their account.

Preconditions: Player is logged in.

Trigger: The player initiates signing out by selecting the Sign-Out option.

Scenario:

1. Player selects the Sign-Out option.

2. System logs out the player from the account.

3. The Player is redirected to the login page.

Post conditions: Player is logged out.

Exceptions:

1. Connection error.

Priority: High Priority. Necessary for the player to be able to sign out of their account.

When available: End of Iteration 3.

Frequency of use: Medium, used whenever player wants to sign out of their account.

Channel to actor: Keyboard and mouse.

Secondary actors: Server.

Channel to secondary actors: Server signs out player and returns to login screen.

Open issues: Confirm before signing out?

GUI Team:

Use case: Customize profile.

Iteration: 1

Primary Actor: Player

Goal in context: The player can update a banner and profile picture.

Preconditions: The player has an active account.

Trigger: The player navigates to their profile page.

Scenario:

1. The player selects "Edit Profile."

2. The UI displays fields for profile picture and banner.

3. The player updates either of these fields and saves changes.

4. The system verifies the updates and applies them.

Post conditions: The player's profile is updated successfully. The changes are viewable by the player and other players.

Exceptions:

1. Picture fails to upload properly, so UI re-displays fields and prompts player to re-select an image.

2. File size is too large, so UI re-displays fields and prompts player to select an image of a smaller size.

3. Wrong file type, so UI re-displays fields and prompts to select a png image.

Priority: Medium. An asset to build a multiplayer community, but not detrimental to system function.

When available: 2nd or 3rd project iteration.

Frequency of use: Occasional. Once the player has created their profile and wishes to customize it.

Channel to actor: In the player’s profile page, under the settings menu.

Secondary actors: N/A

Channel to secondary actors: N/A

Open issues: Should players be able to upload their own profile pictures/banners, or should they only be able to choose from a predefined selection?

Use case: Login to existing account

Iteration: 1

Primary Actor: Player

Goal in context: The player is able to log into their account securely in order to access their profile and play games.

Preconditions: The player has an existing account and the system's authentication system is responsive.

Trigger: The player selects the "login" button on the system's home page.

Scenario:

1. A login screen is displayed to player prompting them to enter their username and password.

2. The player enters their username and password and hits enter.

3. The system verifies the player's credentials using the authentication database.

4. The system verifies the player's credentials and redirects them to the page displaying their profile.

5. If the username and/or password entered by the user are invalid, the screen displays an error message and the user is prompted to re-enter their credentials, or to create an account if they do not have an existing one.

Post conditions: The player has successfully logged in and now has access to their profile, leaderboard, and the systems various game features.

Exceptions:

1. Username and/or password not correct. System displays error message and prompts user to try again.

2. Player has forgotten password. Screen displays "forgot password" option, enabling player to reset their password through email.

Priority: High. Essential in order for players to access the gaming platform and its various features, some of which include playing games, checking leaderboards, and viewing their profile

When available: Second or third project iteration.

Frequency of use: Most, if not every use. Players will need to log in every time they attempt to access the platform after closing it.

Channel to actor: Via the touch screen interface (login screen on the homepage).

Secondary actors: Authentication system/database.

Channel to secondary actors: User database.

Open issues:

1. If the login page provides the option to access the system through third-party logins, how will this be presented on the screen?

2. How will error messages be displayed on the screen?

Use case: Allow players to communicate during gameplay using in-game chat.

Iteration: 1

Primary Actor: Player

Goal in context: Player can communicate with opponents during a match.

Preconditions: The player is actively playing a multiplayer game.

Trigger: The player selects the chat icon on the game interface during a match.

Scenario:

1. Player selects the “chat” button displayed during the match.

2. A chat bar is displayed on the screen, through which players are enabled to type the message they wish to send to their opponents.

3. Player types a message to opponent.

4. Player selects “send” option on the screen to deliver their message.

5. Player may also receive messages from opponents.

Post conditions: The player has communicated with their opponent during an ongoing multiplayer game.

Exceptions:

1. If there is an error that prevents communication, the system will display an error message.

2. Any inappropriate messages sent by a player will be filtered by the system and will not be received by opponent.

Priority: Medium. May increase player’s sense of community and enhance their experience using the system, but not required in order to play games.

When available: Second or third iteration.

Frequency of use: Somewhat frequent. Different players may choose to utilize this functionality more or less frequently based on personal preference.

Channel to actor: The interface available to users during a game match.

Secondary actors: N/A

Channel to secondary actors: N/A

Open issues:

1. When a player receives a message from an opponent, will the system automatically open the chat bar on the receiving player's end, or will they receive a notification which prompts them to open the chat bar on their own (if they choose to do so)?

2. What message/warning will the system display to users who write inappropriate/offensive messages?

Use case: Allow player to choose game from game library.

Iteration: 1

Primary Actor: Player

Goal in context: Player is able to view games offered by the platform and select one to play.

Preconditions: The player has logged in.

Trigger: After submitting login credentials, the player is redirected to the main game page.

Scenario:

1. On the main game page screen, the system displays a list of games available for the player to choose from.

2. When requested by the player, the system displays a brief description for each game selection on the game library page interface.

3. Player selects desired game from the list.

4. Player is redirected to the game play screen.

Post conditions: The player has selected a game to play and has now been redirected to the screen where they will play the chosen game.

Exceptions:

1. In the case of a server issue, the system will display an error message to the user to try again later.

2. If there are no opponents available to match, the system will display this message to the user and request that they wait for a match.

Priority: High. Selecting games is necessary in order to be able to play games using the system.

When available: Second or third iteration.

Frequency of use: Frequent. Players will need to select a game each time they wish to start playing a game or enter a new match.

Channel to actor: Game page screen.

Secondary actors: N/A

Channel to secondary actors: N/A

Open issues:

1. How will information be displayed for each game? Will the player have the option to choose to view instructions on how to play if needed?

Use case: Allow users to view their game history.

Iteration: 1

Primary Actor: Player

Goal in context: By viewing their profile, the player can access their past game history (wins, losses, playing statistics) and rankings.

Preconditions: The user has logged into their account on the system.

Trigger: The user selects the “profile” button on the home game page.

Scenario:

1. After player selects the profile button on the home screen, the system displays a list of past games played by the logged in player and win/loss and rank statistics.

2. Player selects specific games played to view more specific details of a given match, such as loss or win or ranking, which is displayed by the system.

Post conditions: The player has viewed their game history.

Exceptions:

1. If the user has not played any games, the system will display a message to the user redirecting them back to the home game page to select a game.

Priority: Medium priority. While users may be curious about their game history/status, it is not essential in order for the gameplay features of the system to run.

When available: Second or third iteration.

Frequency of use: Somewhat frequent. Some players may occasionally wish to view their game history out of curiosity, or to improve their gameplay skills.

Channel to actor: Via the GUI (game selection homepage).

Secondary actors: N/A

Channel to secondary actors: N/A

Open issues:

1. Other player’s have access to view some of their opponent’s game history (i.e. ranks, current status, recent matches). Should there be an option presented to users allowing them to make this information private?

Use case: Volume settings

Iteration: 1

Primary Actor: Player

Goal in context: Change the volume of background music.

Preconditions: The player has started a match.

Trigger: Pressing the settings button.

Scenario:

1. The player selects a game (connect 4, checkers, tic tac toe) on the game page.

2. The player presses the settings button.

3. The player adjust the volume to their desired intensity.

4. The player goes back to match.

Post conditions: The volume has been adjusted accordingly

Exceptions:

1. The volume is lowered

2. The volume is intensified

Priority: Low. Not all players will need to change the background music, it will only be for those that want it to be louder or lower.

When available: 2nd or 3rd project iteration.

Frequency of use: Occasional. If the player finds the volume too loud or too quiet they can change the volume to their need.

Channel to actor: In the active game Window, under settings icon.

Secondary actors: N/A

Channel to secondary actors: N/A

Open issues: Am not sure if we will have the music throughout the whole game. Would the music also be playing while game is selected? If so there will be more than one case description for this.

Use case: Toggle between light and dark mode.

Iteration: 1

Primary Actor: Player

Goal in context: Allow player to switch their display settings to either light or dark mode, ensuring visual comfort and accessibility.

Preconditions: The player is logged into their account.

Trigger: The player navigates to the settings menu and selects the “Appearance” button.

Scenario:

1. By toggling a switch, the player selects either light mode or dark mode.

2. The player’s selection takes effect on their visual display.

3. Player saves the changes and the system applies these visual changes to the rest of the system.

4. Player may press the back button to exit settings and return to the main interface.

Post conditions: The UI has updated to the correct theme to match the user’s preferences (light or dark mode).

Exceptions:

1. If the changes made by the player are not saved, the system reverts to a default (light mode).

Priority: Low. Not needed in order for game to function, but may improve accessibility aspect.

When available: Second or third iteration.

Frequency of use: Occasionally. User may decide to change light mode or dark mode based on their preferences or current lighting conditions.

Channel to actor: GUI settings menu.

Secondary actors: N/A

Channel to secondary actors: N/A

Open issues:

1. Should the user be able to set their light mode/dark mode to change automatically based on the time of day? (i.e. switch to dark mode at 8:00pm)

Game Logic Team:

Use case: Quit game

Iteration: 1

Primary Actor: Player

Goal in context: Allow the player To Voluntarily Quit The Game.

Preconditions: The player has joined a match, and the game is ongoing.

Trigger: The player wishes to leave the ongoing game.

Scenario:

1. The player presses the “quit” button.

2. The player is warned, before leaving, that he will be given a loss for leaving the match.

3. The player presses the “continue” button.

4. The game is ended and the leaving player is given a loss, while the other player is awarded a win.

Post conditions: The player has voluntarily left the game, and the game ends.

Exceptions: N/A

Priority: Medium-High Priority. Is an almost essential feature for many players to have available to them. Although it is not essential for gameplay functionality, it is a good feature to provide to our players to ensure they do not feel "trapped" in a game.

When available: Second or third iteration.

Frequency of use: Somewhat frequent. Some players may wish to leave as a form of conceding to their opponent, others may be under a time crunch, and many other reasons.

Channel to actor: GUI

Secondary actors: N/A

Channel to secondary actors: N/A

Open issues: If a player consistently leaves ongoing games, should they receive incrementing forms of punishment? (lower matchmaking priority, temporary matchmaking bans, timeouts, other?)

Use case: Update Current Game State

Iteration: 1

Primary Actor: Game Logic software

Goal in context: To update the current state of the game being played to the player in real-time. The game logic should be able to know when a player is ending/beginning their turn. It should also know when a game is over and update the game state to be completed.

Preconditions: The player has joined a match, and the game state has initiated.

Trigger: Whenever a game is started and a player is designated to make their turn, or whenever the game has concluded.

Scenario:

1. The game state is initiated and a player takes their turn.

2. The game state is updated to ensure the correct player is able to make their turn as designated.

3. The designated player makes their turn.

Post conditions: The game state is updated and the game is either initiated, awaiting a player's turn, or completed.

Exceptions:

1. The game ends

Priority: High priority.

When available: 2nd or 3rd iteration.

Frequency of use: Always. Game state is being constantly updated through gameplay.

Channel to actor: Game Logic Java Classes.

Secondary actors: N/A

Channel to secondary actors: N/A

Open issues:

1. If there are network errors or disconnects, how will we handle delayed game status updates? Must prepare for this case to ensure that the game state is appropriately updated when possible for both players.

Use case: Hint

Iteration: 1

Primary Actor: Player

Goal in context: Help new players understand game rules.

Preconditions: The player has started a game.

Trigger: The player clicks on the button for a hint on the screen.

Scenario:

1. The player selects the tutorial or hint button from the game interface.

2. the player will receive a game-appropriate hint that nudges them in the right direction in relation to the game.

3. The player should be allowed to then close the hint and continue playing.

Post conditions: The player received a tutorial/hint and continues with the game.

Exceptions:

1. The game may not have a hint to give.

Priority: Medium priority. It's not extremely necessary as these games are quite common and popular, you can find the rules anywhere, but some users may not know it well enough and would need a tutorial or hints or both.

When available: Between 2nd and 3rd iterations.

Frequency of use: Somewhat frequent. As mentioned above, some people may need to use it and some don't.

Channel to actor: The game's interface.

Secondary actors: N/A

Channel to secondary actors: N/A

Open issues: How would the hints be generated?

Use case: Rematch Request

Iteration: 1

Primary Actor: Player

Goal in context: The player is able to request a new game with the same opponent with whom they have just played.

Preconditions: The player finishes a game.

Trigger: Player selects the 'Request Rematch' button in the game interface.

Scenario:

1. The player selects the "Request Rematch" button to notify the opponent.

3. Opponent is notified of Rematch Request.

4. If Opponent denies the request, both players are returned to game lobby.

5. If Opponent accepts, the same game loads once more with the same player.

Post conditions: The player and opponent are reloaded to the same game in its initial state to play again.

Exceptions:

1. One player breaks connection with the server before the request is made.

2. Opponent does not respond

3. Both players send a rematch request

Priority: medium to high priority. Rather basic functionality of a game system.

When available: Between 2nd and 3rd iterations

Frequency of use: Somewhat frequent. Players might wish to play the same opponent again, or they may not.

Channel to actor: The game's interface.

Secondary actors: Opponent

Channel to secondary actors: Server

Open issues:

Use case: Determine match outcome.

Iteration: 1

Primary Actor: Game Logic Software.

Goal in context: Allow the game logic to determine the result of a match, assigning a winner or a draw.

Preconditions: A game is ongoing between 2 different players from a game

Trigger: A player has either made a move which satisfies a winning condition, or no more legal moves are allowed to be taken from either player.

Scenario:

1. The game results are created

2. The game results are stored onto the database

3. The players are notified of the outcome

Post conditions: The match outcome has been determined, and the game ends.

Exceptions:

1. A state where players have legal moves yet cannot end the game

Priority: High priority. Essential for the standard functionality of the game logic. Also used to determine what exactly are the win conditions for each game within the software.

When available: Between 2nd and 3rd iterations.

Frequency of use: Always, will be used once during every match.

Channel to actor: Game Logic Java Classes.

Secondary actors: Players

Channel to secondary actors: Internet connection

Open issues: It may be difficult or troublesome to effectively implement the different winning conditons required for each game, so we should ensure that we can find a simple and effective way to implement this functionality within the software.

Networking Team:

Use case Join a multiplayer game (client-server Model).

Iteration: 1

Primary Actor: Player (client)

Goal in context: The player can start a game session.

Preconditions:

1. The player is logged into their account.

2. The client is successfully connected to the server.

3. The player is on the game selection screen.

4. The game server is online and responsive.

Trigger: The player selects a game and requests to join a session.

Scenario:

1. The player selects from the list of board games to play.

2. The client sends a game session request through the NetworkManager.

3. The ServerController receives the request and either finds an existing session or instructs the Game Session Manager to create a new session.

4. The GameSessionManager establishes a new session and assigns the player via their PlayerHandler.

5. The game initializes through the Game State Synchronizer, and the initial game state is shared.

Post conditions: The player is successfully connected to a game session through a client-server socket connection.

Exceptions:

1. The server is unreachable, preventing the player from joining.

2. The client loses connection before the game session is initialized.

3. The game session has reached the player limit.

4. The request is incorrect, or corrupted during connection phase.

5. A timeout threshold is surpassed, where either the server or client do not receive a response within a predefined time.

Priority: High priority. Establishing a reliable multiplayer connection is crucial for multiplayer gameplay.

When available: 2nd or 3rd iteration.

Frequency of use: Frequent. Everytime player's would like to play against other human opponents.

Channel to actor:

1. GUI interaction.

2. Socket connection for communication with the server.

Secondary actors: Game Server which handles session management and player matchmaking.

Channel to secondary actors: TCP socket communication between client and server for session handling.

Open issues: What kind of authentication mechanism should be used for verifying player join requests?

Use case: Multiplayer Game State Update (client-server Model).

Iteration: 1

Primary Actor: Player (client)

Goal in context: The player makes an in-game action that is sent to the server. The server then updates the game state and sends the new state to all connected players.

Preconditions:

1. The player is logged into their account.

2. The player is connected to the game.

3. The client is connected to the server.

4. The game server is online and responsive.

Trigger: The player takes an action in the game.

Scenario:

1. The player takes an action in the client.

2. The client sends the action to the server.

3. The server game session manager updates the game state.

4. The server game state synchronizer sends the new game state to all connected clients.

5. The client receives the new game state and replaces the old game state with it.

Post conditions: All players receive the updated game state.

Exceptions:

1. The server is unreachable, so the action is not sent to the server.

2. The client loses connection before the game state is synchronized.

3. A move is invalid or corrupted during the game state update.

4. A timeout threshold is surpassed, where either the server or client do not receive a response within a predefined time.

Priority: High priority. Being able to make moves that all players can see is crucial for multiplayer gameplay.

When available: 2nd or 3rd iteration.

Frequency of use: Very Frequent. Every time any player takes an action in a multiplayer game.

Channel to actor:

1. GUI interaction.

2. Socket connection for communication with the server.

Secondary actors:

1. Game Server which handles session management and player matchmaking.

2. Other player clients

Channel to secondary actors:

1. TCP socket communication between client and server for session handling.

Open issues:

1. How to handle simultaneous moves from different players.

2. How to make sure the game state is updated at the same time for all players.

Use case: Server Account Creation (client-server Model)

Iteration: 1

Primary Actor: Player (client)

Goal in context: The player creates a new account by entering their details on the registration screen, their credentials are then stored on the server.

Preconditions:

1. The client is connected to the server.

2. The player is on the registration screen.

3. The game server is online and responsive.

Trigger: The player selects the option to register a new account.

Scenario:

1. The player enters the required registration details.

2. The client sends the registration information and request to the server.

3. The server checks the validity of the submitted information.

4. The server stores the player's registration info in its database.

5. The server sends confirmation back to the client and connects to the player's account.

Post conditions: The player's account is successfully created, and is connected to the server.

Exceptions:

1. The server is unreachable, so the registration info was not sent to the server.

2. The database is unreachable and cannot store the data.

3. The player's registration information is invalid

Priority: High priority. Having an account is necessary to connect to the multiplayer aspects of the game.

When available: 2nd or 3rd iteration.

Frequency of use: Low. Will be used when a player first begins using the multiplayer aspects of a game

Channel to actor:

1. GUI interaction.

2. Socket connection for communication with the server.

Secondary actors:

1. Game Server

2. Game Database

Channel to secondary actors:

1. TCP socket communication between client and server for session handling.

Open issues:

Use case: Player Disconnection & Reconnection

Iteration: 1

Primary Actor: Player

Goal in context: Ensure that if a player disconnects from an ongoing game session, their game state is retained, allowing them to reconnect and resume the match if they return within a certain timeframe.

Preconditions:

1. The player is currently connected to an active multiplayer session.

2. The server is online and managing the session.

3. The game session is still ongoing when the disconnection occurs.

Trigger: The player's connection is cut due to network issues, system crashes, or manual disconnection.

Scenario:

1. The player is currently within a multiplayer game session.

2. The player disconnects unexpectedly due to internet issues, game/system crashes, or manual disconnection.

3. The player’s status is updated via PlayerHandler, marking them as temporarily disconnected.

4. The Game Session Manager keeps the session active, while Game State Synchronizer freezes the game state.

5. If the player reconnects:

- The client sends a reconnection request to the server - AuthHandler verifies the player.

- NetworkManager restores the socket connection.

- The GameSessionManager resumes the player’s position and state.

Post conditions: If the player successfully reconnects, they resume the game from the last known state.

Exceptions:

1. The player does not reconnect within the allocated time, and the game session is forced to continue.

2. If the server goes down before the player reconnects, their game state may be lost.

3. The player reconnects but fails identity verification.

4. If the player repeatedly disconnects and reconnects, the server may impose a limit.

Priority: High. Disconnection and reconnection handling is essential for preventing abuse of online system, and ensuring smooth multiplayer experience.

When available: 2nd or 3rd iteration.

Frequency of use: Variable. Some player's may rarely disconnect, while others may have unstable connections, or abuse the system.

Channel to actor:

1. Player interacts via the game client.

2. Reconnection occurs via the server-client communication.

Secondary actors: N/A

Channel to secondary actors: N/A

Open issues:

1. How long should the server retain the player's game state before removing them from the session?

2. How should the system handle repeated connections and disconnections from players?

3. What security measures are in place to prevent exploitation of disconnect/reconnect logic?

4. How long should the reconnect threshold times be, and should it depend on game type?

Use case: Authentication & Session Handling (Server-client model)

Iteration: 1

Primary Actor: Server

Goal in context: Verify player login credentials

Preconditions:

1. The player has submitted login credentials from client side.

2. The server is up and running, and connected to the database.

3. The database is available and responding to query requests.

Trigger: The server receives a login request from the player.

Scenario:

1. The player starts the game client, and enters their login credentials.

2. The server receives the login request containing their username and password. AuthHandler processes the request.

3. Database Connector queries the database for matching username.

4. If the username exists, the database returns the stored password hash.

5. The server then compares the received password (after applying same hashing method) to the password stored in the database

6. If the password is valid:

7. AuthHandler generates a unique token for the player

8. The server stores this session token linked to the player's account - Player's PlayerHandler is created and attached.

9. The server then sends a "login successful" back to the client.

10. If the password is invalid, or the username does not exist, then the server sends a "login failed" response back to the player.

Post conditions:

1. On success, the player gets an active game session on the server, identified by their unique token.

2. On failure, the player remains in the login page, and may retry login.

Exceptions:

1. Player is unable to start a game session due to server or database malfunction.

2. Incorrect login details. Server rejects login attempt.

Priority: High. Authentication is critical for players accessing game sessions.

When available: 2nd iteration.

Frequency of use: Frequent. Player's need to log in at the beginning of every session.

Channel to actor:

1. TCP socket communication between client and server.

Secondary actors: Database

Channel to secondary actors:

1. SQL query from server to database.

Open issues:

1. How are passwords securely stored? what kind of hashing algorithm will be used to ensure this.

2. Should multiple failed login attempts lock the account after a certain amount of retries?

3. How will session tokens be handled. How will they expire or be refreshed?

4. Should tokens persist after accidental disconnect? Will the player need to re-log in to continue the session?

Use case: Turn Timeout Handling

Iteration: 1

Primary Actor: Server

Goal in context: Ensuring timely gameplay by starting a countdown timer when a player begins their turn.

Preconditions:

1. A game session is active between two connected players.

Trigger: A player's turn begins.

Scenario:

1. The GameStateSynchronizer signals the player's turn.

2. The server starts a countdown timer for said player.

3. PlayerHandler sends an alert to the player notifying them that it's their turn, and time limit is displayed.

4. If the player makes a valid move before the time expires:

5. The server receives the move from the player.

6. The move is validated and game state is updated for both players.

7. The timer is cancelled.

8. If the timer expires before a valid move is received:

9. The server decides on a timeout event (based on game rules).

10. The server updates the game state to reflect those rules, and both players are notified.

11. The server passes the turn to the next player and repeats the process.

Post conditions:

1. The game proceeds to the next player's turn.

2. The game state is updated for both clients to reflect move choice.

Exceptions:

1. Player disconnects during their turn, connections are dropped.

2. Network delays cause late move submissions.

Priority: High priority. Ensures smooth, consistent gameplay.

When available: 2nd or 3rd iteration.

Frequency of use: Frequent. Occurs at every turn.

Channel to actor:

1. Server controlled timer with notifications sent over TCP sockets.

Secondary actors: N/A

Channel to secondary actors: N/A

Open issues:

1. What should the standard timeout length be? should it be different for each game?

2. What should happen if the timer threshold is exceeded? should a move be done by default? Should the player's turn be skipped?

3. How are reconnecting player's handled if they come back midturn?

4. Should timeouts affect a player's ranking/statistics?

Use case: End/Delete Game Session

Iteration: 1

Primary Actor: Server

Goal in context: To terminate an active game session when the match ends, a player quits, or disconnection occurs.

Preconditions:

1. An active game session exists and managed by the GameSessionManager.

2. Players are connected through their respective PlayerHandlers.

3. The game state is actively synchronized by the GameStateSynchronizer.

Trigger: A game ending condition is met (win, draw, player quit, or disconnection).

Scenario:

1. GameSessionManager detects the session should end (through game logic or player action).

2. GameStateSynchronizer finalizes the last state and communicates the final result to both players.

3. ServerController is notified the game session is complete.

4. NetworkManager sends a session termination message to clients.

5. Database stores the match result in match history and updates any affected leaderboard

data.

6. Player handlers update the current state (no session) and players are returned to a waiting, or menu screen.

7. The GameSessionManager deletes the session instance.

Post conditions:

1. The game session is terminated.

2. Results from the match are stored in the database.

3. Players are notified that the session has ended and removed from the game session.

Exceptions:

1. Either clients disconnect during the game ending process, or the server goes down.

2. Match results cannot be stored due to database failure.

Priority: High. Proper session termination is critical for maintaining a stable system, and accurate player records.

When available: Iteration 2

Frequency of use: High frequency. Occurs at the end of every match.

Channel to actor:

1. Triggered internally via the game session manager.

Secondary actors:

1. Player handlers

2. Database connector

Channel to secondary actors:

1. TCP socket communication with clients.

2. SQL queries.

Open issues:

1. In the event of a server, or database failure mid-process, what should the contingency plan be?

2. If a player disconnects unexpectedly (network failure, system crash etc.), should the game end immediately or allow a reconnection window?

Use case: Chat Message Handling (client-server model)

Iteration: 1

Primary Actor: Player (client)

Goal in context: Allow the player to send chat messages using the chat box. The message is routed through the server and then sent to all other players.

Preconditions:

1. The player is logged into their account.

2. The client is connected to the server.

3. The player is connected to the game.

4. The game server is online.

5. The chat box is active

Trigger: The player types and sends a chat message

Scenario:

1. The client sends the message to the game server.

2. The server checks the appropriateness of the message.

3. The server sends the chat message to the other connected players.

4. The client receives the message and updated the chat box.

Post conditions: All players receive the new chat message

Exceptions:

1. The server is unreachable, so the message was not sent to the server.

2. A timeout threshold is surpassed, where either the server or client do not receive a response within a predefined time.

3. The player's message is inappropriate

Priority: Low priority. Messaging is secondary to the games core functions.

When available: 2nd or 3rd iteration.

Frequency of use: Medium. Will be used every time a player wants to send a message to other players.

Channel to actor:

1. GUI interaction.

2. Socket connection for communication with the server.

Secondary actors:

1. Game Server

2. Other player clients

Channel to secondary actors:

1. TCP socket communication between client and server for session handling.

Open issues:

1. How to handle message moderation

2. How to handle simultaneous messages from different players

3. How to handle very large chat messages

4. How to make sure the chat box is updated for all players

Use case: Server Validates Action (client-server Model)

Iteration: 1

Primary Actor: Game Server

Goal in context: The server verifies that each move submitted by a player is allowed before updating the game state.

Preconditions:

1. The client is connected to the server.

2. The game server is online and responsive.

3. An active game is in progress

Trigger: The client sends a player action to the server.

Scenario:

1. The server receives the move.

2. The server checks the move against game rules and current state.

3. If the move is valid, the server game session manager updates the game state.

Post conditions: The game state is updated and sent to all player clients.

Exceptions:

1. The player's action is invalid

Priority: High priority. Ensuring that the game works is a necessary part of the multiplayer game.

When available: 2nd or 3rd iteration.

Frequency of use: High. Will be used every time a player makes a move in multiplayer.

Channel to actor:

1. GUI interaction.

2. Socket connection for communication with the server.

Secondary actors:

1. Game Clients

Channel to secondary actors:

1. TCP socket communication between client and server for session handling.

Open issues:

1. How to manage simultaneous moves

Leaderboard and Matchmaking Team:

Use case: View Leaderboard

Iteration: 1

Primary Actor: Player

Goal in context: Allow players to view the ranking of top players based on performance metrics

Preconditions:

1. Player must have access to the game interface

2. Leaderboard data is available and updated

Trigger: Player navigates and selects the leaderboard section in game menu

Scenario:

1. Player selects the leaderboard option in game menu
2. System/server retrieves and updates the latest player rankings
3. GUI displays the list of top players sorted by rating (descending order by default)
4. Player can scroll through the leaderboard list
5. Player can search for specific users

Post conditions: The player can view the leaderboard.

Exceptions:

1. No ranking data is available

2. Network error preventing leaderboard data from being retrieved

Priority: High

When available: In 1 sprint, by end of project iteration 3

Frequency of use: Moderate, varies on player interest

Channel to actor:

1. In game GUI

2. Physical interaction using input devices such as keyboard or mouse

Secondary actors: Player database (for retrieving ranking data)

Channel to secondary actors: System communication

Open issues: Should there be additional filtering options for different ranking types (eg, game stats)?

Use case: Update Leaderboard Data

Iteration: 1

Primary Actor: Server

Goal in context: Ensure leaderboard displays most recent player ranking and/or performance data

Preconditions:

1. Player completes a ranked game

Trigger: A ranked game is completed

Scenario:

1. System receives match results (win/loss data, etc.)

2. System updates player ratings and rankings based on performance (remove rating score if player loses, grants rating score if player wins)

3. System sorts the updated list and stores it in database

Post conditions: Leaderboard data accurately reflects the most recent rankings

Exceptions:

1. Network/database failure – preventing updates

Priority: High

When available: 1 sprint, by end of project iteration 2

Frequency of use: After every ranked match completion

Channel to actor: System communication

Secondary actors: Player database

Channel to secondary actors: System communication

Open issues: Do we have to have the leaderboards updated after every match or could we update them less to reduce load?

Use case: Sort leaderboard

Iteration: 1

Primary Actor: Server

Goal in context: Retrieve an ordered list of players based on ranking metrics

Preconditions: Leaderboard data is available

Trigger: Player requests top players

Scenario:

1. Server receives a request for player rankings

2. System finds the highest ranked players from the leaderboard

3. Server sends back the appropriate list of players

Post conditions: Request player rankings are displayed or used in another system process.

Exceptions:

1. Database failure preventing retrieval

2. No data is available for processing

Priority: Medium/high

When available: 1 sprint, by end of project iteration 3

Frequency of use: Moderate, varies on player interest.

Channel to actor: System communication

Secondary actors: Player database

Channel to secondary actors: System communication

Open issues: What other sorting options do the leaderboard settings offer?

Use case: Search for specific player (including themselves) on the leaderboard

Iteration: 1

Primary Actor: Player

Goal in context: Allow player to search for specific player (by username or maybe other identifiers) to view their ranking

Preconditions:

1. Leaderboard must be accessible

2. Player is viewing leaderboard

3. Searched player exists in the ranking database

Trigger: Player searches a username in leaderboard menu

Scenario:

1. Player navigates to the leaderboard.

2. Player searches for a username.

3. If username is found, system removes all other players except the searched player.

4. If username is not found, system displays a “player not found” error message.

Post conditions: Player can view the searched player’s ranking and statistics.

Exceptions:

1.Searched player does not exist

2. Database error fails to retrieve player statistics

3. Network error prevents GUI and database from communicating

Priority: Medium

When available: 2 sprints, end of project iteration 3.

Frequency of use: Moderate, varies on player interest.

Channel to actor: GUI

Secondary actors: Player database

Channel to secondary actors: System communication

Open issues: Should search feature support partial matches?

Use case: View friends on leaderboard

Iteration: 1

Primary Actor: Player

Goal in context: Allow players to filter the leaderboard to view only their friend's rankings.

Preconditions:

1. Player has access to leaderboard menu.

2. Leaderboard data and friends list must be available and updated.

Trigger: Player selects the option to view only friends on leaderboard.

Scenario:

1. Player navigates to leaderboard menu.

2. Player selects "View Friends" filter option.

3. System retrieves player's friends list.

4. System filters the leaderboard to display rankings of the player's friends.

5. Player can scroll and view the filtered list.

Post conditions: The leaderboard shows only the player's friends.

Exceptions:

1. Network error prevents leaderboard data from being retrieved

2. Network error prevents access to friend's list

Priority: Medium

When available: 1 sprint, by end of project iteration 3

Frequency of use: Moderate, varies on player interest.

Channel to actor: GUI

Secondary actors: Player database

Channel to secondary actors: System communication

Open issues: Should players be able to sort the friend's leaderboard separately?

Use Case: Start Matchmaking

Primary Actor: Player

Goal in Context: To allow players to enter a matchmaking queue and find a suitable match

Preconditions:

1. Player is logged on

Trigger: Player clicks “Start Game”

Scenario:

1, Player selects game mode (Casual, Ranked, and etc)

2. System searches opponents depending on factors such as rank, preference, region, and etc.

3. System finds a match and displays confirmation

4. System adds player into the lobby

Postconditions: Player is delivered into the game lobby with an opponent of a similar rank

Exception(s):

1. No match is found for a player

2. Network connection is unstable, system cannot proceed

Priority: High

When Available: Within 1 to 2 sprints

Channel to Actor: Game UI

Secondary Actors: Player Database (To access ranks)

Channel to Actor: System Communication

Open Issues:

- If a player leaves a match mid process, do they suffer consequences?

- Should there be a confirmation before a player is placed into a game lobby with the opponent?

Use Case: Match Confirmation

Primary Actor: System

Goal in Context: To display confirmation of match found

Preconditions:

1. Player is logged on

2. Payer has initiated the matchmaking process

Trigger: System has found an appropriate match

Scenario:

1. Player has initiated the matchmaking process.

2. System finds an appropriate match and displays the opposition players name on the game interface

Postconditions: The system displays opposition players ranking and name

Exceptions:

1. System does not find any appropriate match

Priority: High

When available: 1 to 2 sprints

Channel to Actor: Game UI

Secondary Actors: None

Channel to Actor: N/A

Open Issues:

Use Case: Finding Opponent

Primary Actor: Sever

Goal in Context: Pairing two similar ranking opponents

Preconditions:

1. The server is online

Trigger: Player clicks Start Game

Scenario:

1. Players join the matchmaking process.

2. System goes through the queue list of players waiting to matched with an opponent in the same game

3. System finds an opponent with a ranking within a range of the with the pairs ranking

Postconditions: A game is started with the players

Exceptions:

1. There are no players in queue

Priority: High

When available: 1 to 2 sprints

Channel to Actor: Game UI

Secondary Actors: Players

Channel to Actor: Game UI

Open Issues: What if the system cannot find an opponent player with ranking within the range? Should the algorithm expand its range each time it fails?

Use Case: Game Invite

Primary Actor: Player

Goal in Context: Send a game invite to a friend.

Preconditions:

1. Player has opponent as friend.

2. Player's friend is online

Trigger: Player clicks "Send Challenge"

Scenario:

1. The player opens their friends profile.

2. The player presses the "invite" button

3. The player selects a game

4. The opponent either accepts or rejects the invite

Postconditions: A game is initiated between the two players

Exceptions:

1. Opponent player disconnects

2. Opponent player rejects the invite

Priority: Medium

When available: 2 to 3 sprints

Channel to Actor: Game UI

Secondary Actors: Server

Channel to Actor: Internet connection

Open Issues: Should there be an invite timeout if the opponent doesn't respond?

Use Case: Invite Confirmation

Primary Actor: Player

Goal in Context: Accepting a game invite to join a friends game

Preconditions: The player is logged in

Trigger: The player receives a game invite

Scenario:

1. The invite is shown to the player

2. The player presses the “accept” button

Postconditions: Game session is initialized

Exceptions:

1. Either one of the player loses network connectivity during the process

2. Player declines invite

Priority: Medium

When available: 1 to 2 sprints

Channel to Actor: Game UI

Secondary Actors: System

Channel to Actor: System database and Internal communication

Open Issues: