List of Files Being Reviewed:

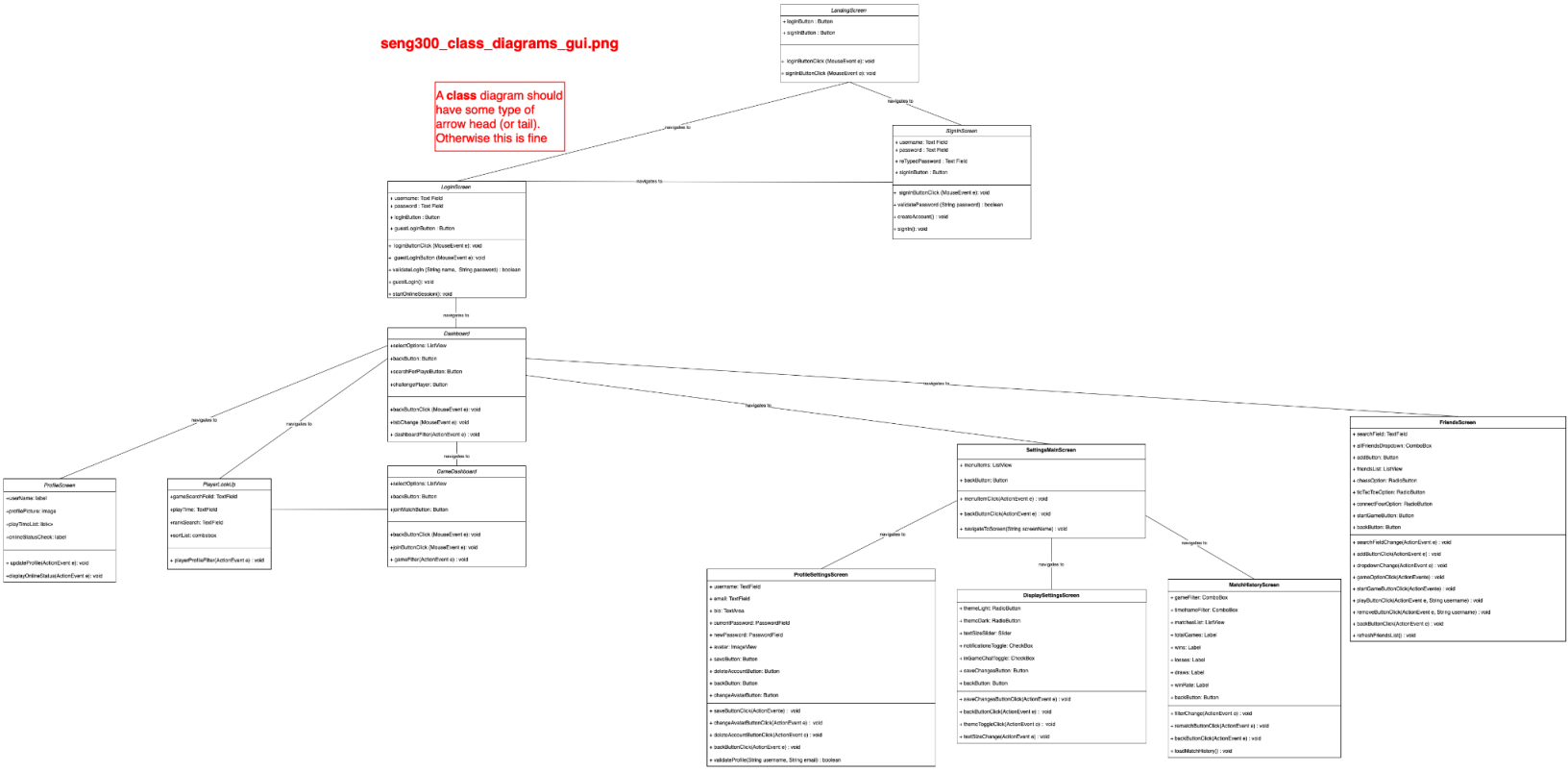
Structure diagrams:
class_structure_diagram_integreation.png - Iman Hamzat, complete(March 14, 11:02pm)
seng300_class_diagrams_gui.png - Iman Hamzat, complete(March 14, 11:12pm)
class_diagram_Authentication.png - Iman Hamzat, complete(March 14, 11:21pm)
Leaderboard_and_matchmaking_class_diagram.png - Iman Hamzat, complete(March 14, 11:35pm)
Class_diagram_networking.png - Iman Hamzat, complete(March 14, 11:43pm)
Class_structure_diagram_chess.png - Brad Cronin, complete (2025-03-15 2:06 pm)

Use Case diagrams

Use Case Descriptions

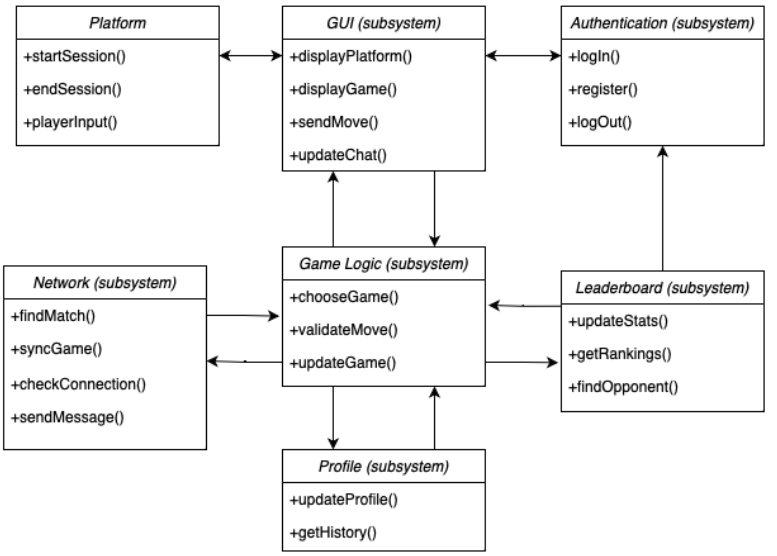
Planning and Timeline Documents

UI Mockups



class_diagram_integration.png

This diagram seems to show how different components are connected. The plan looks good. The main issues are with the **syntax** of a UML **class** diagram



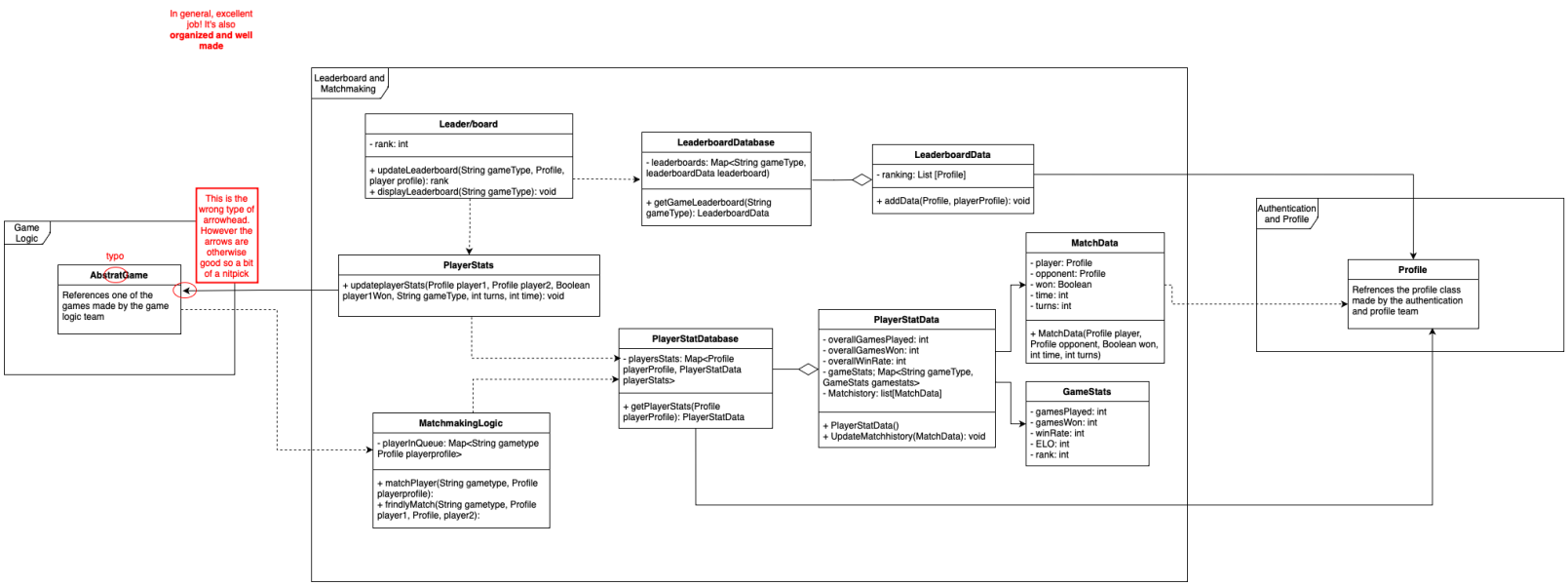
Each class appears to be incomplete (or simply lacks fields) but this is fine for planning purposes

This is the wrong type of arrow for a UML **class** diagram. Depending on what the diagram wants to communciate however, it may be acceptable

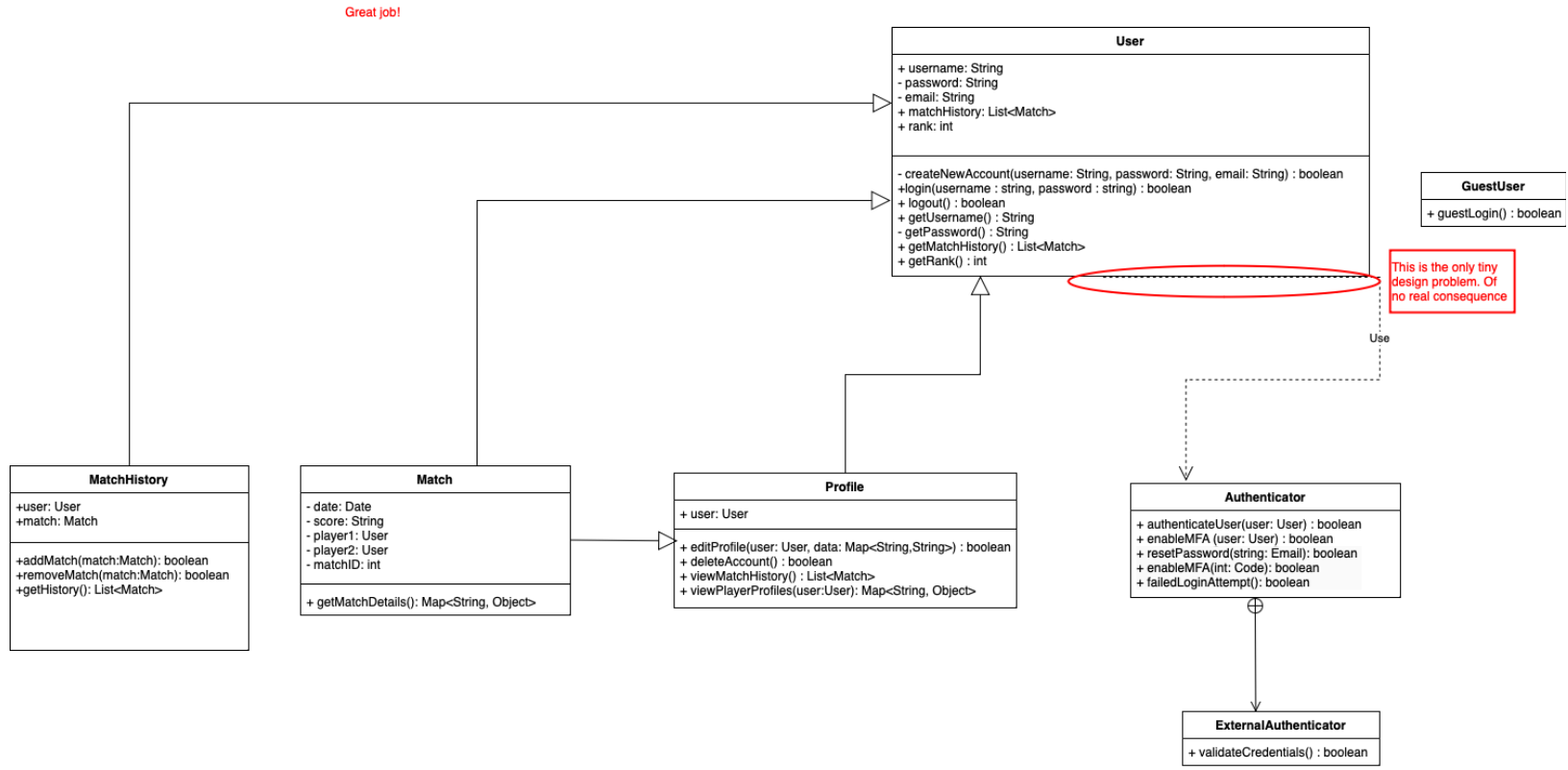
Platform	Authentication
Methods: startSession(): void Initializes the platform and all subsystems. endSession(): void Closes the session and shuts down subsystems. playerInput(input: string): void Routes player actions to the appropriate subsystem.	**Methods:** logIn(username: string, password: string): boolean Authenticates the player. register(username: string, password: string): boolean Registers a new player. logOut(): void Ends the player's session.
GUI	**Networking**
Methods: displayGame(gameState: GameState): void Renders the game board and interface. updateChat(message: string): void Displays chat messages from the opponent. sendMove(move: string): void Sends the player's move to the GameLogicManager.	**Methods:** findMatch(playerID: string): string Searches for an opponent using the LeaderboardManager. sendMessage(message: string): void Sends chat messages to the opponent. syncGame(gameState: GameState): void Synchronizes the game state between players. checkConnection(): boolean Verifies the player's internet connection.
Game Logic	**Leaderboard and Matchmaking**
Methods: validateMove(move: string): boolean Ensures the player's move is valid. updateGame(move: string): GameState Updates the game state after a move. endGame(): Result Determines the winner and sends results to LeaderboardManager.	**Methods:** updateStats(playerID: string, result: Result): void Updates player stats after a game. getRankings(): Ranking[] Retrieves the current leaderboard rankings. findOpponent(playerID: string): string Finds a suitable opponent based on skill level.
Profile	
Methods: updateProfile(playerID: string, stats: Stats): void Updates the player's profile with new stats and game history. getHistory(playerID: string): GameHistory[] Retrieves the player's game history.	

Not sure what the purpose of having this present in the diagram is. It repeats the same infomation above but with more detail. This is not necessary and could have been part of the diagram. However if it's for readability and planning purposes then it's fine

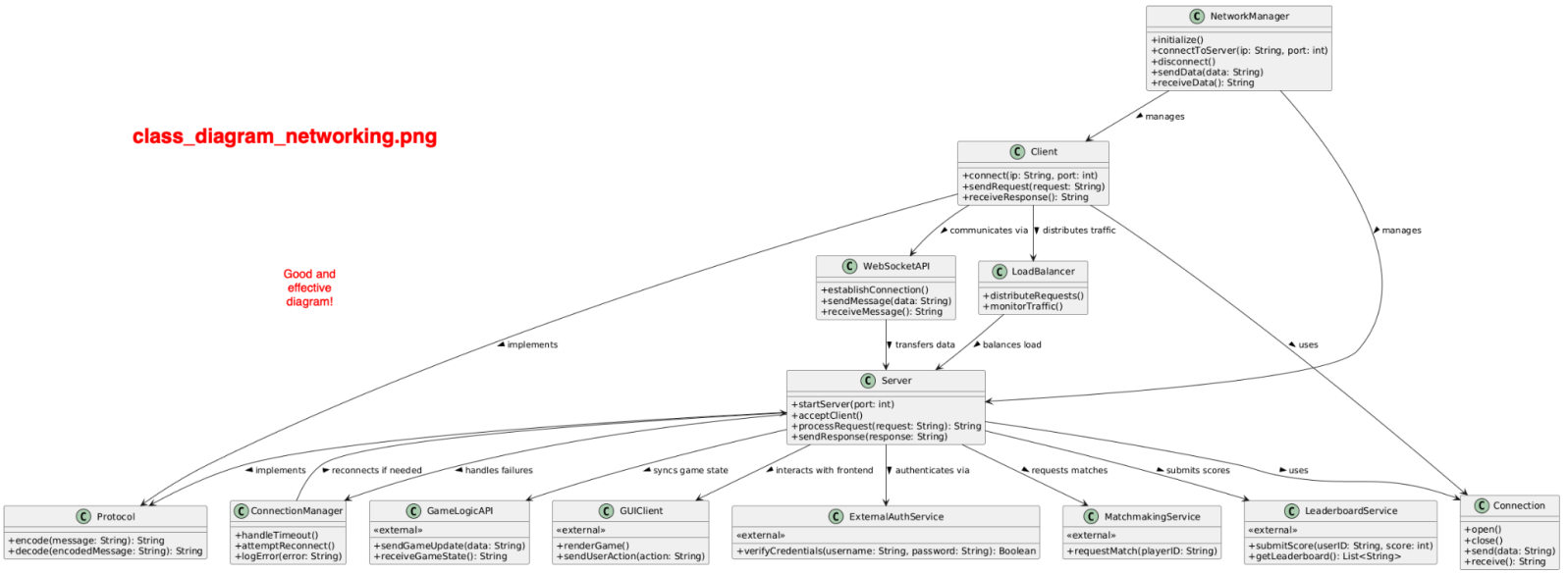
leaderboard_and_matchmaking_class_diagram.png



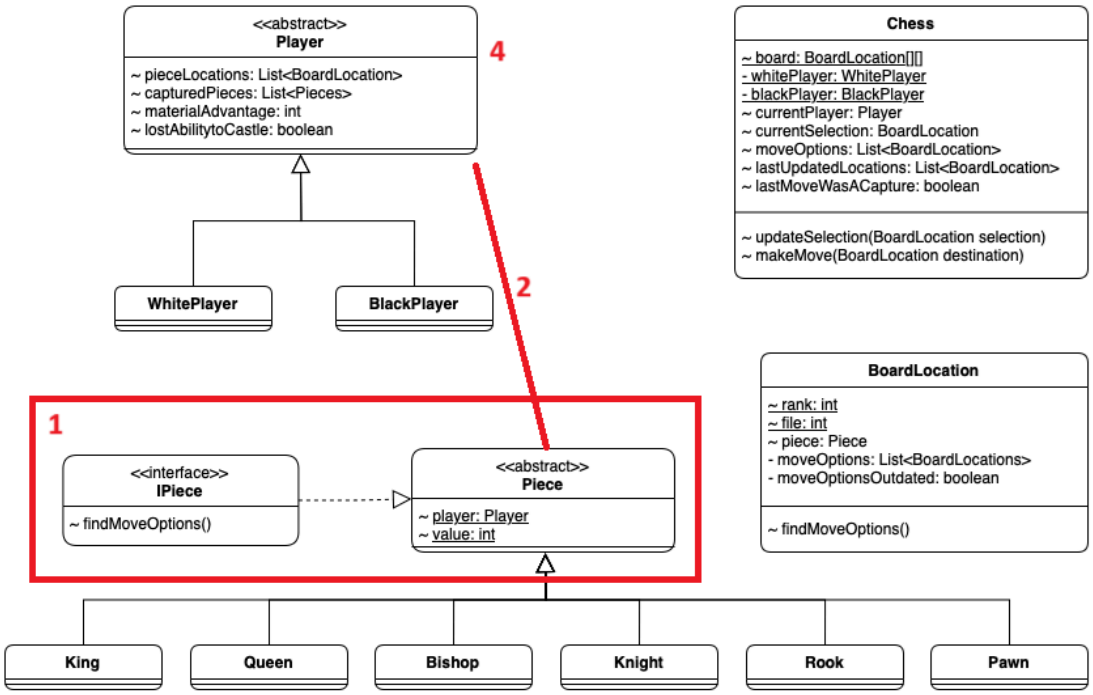
class_diagram_Authentication.png



class_diagram_networking.png



class_structure_diagram_chess.png

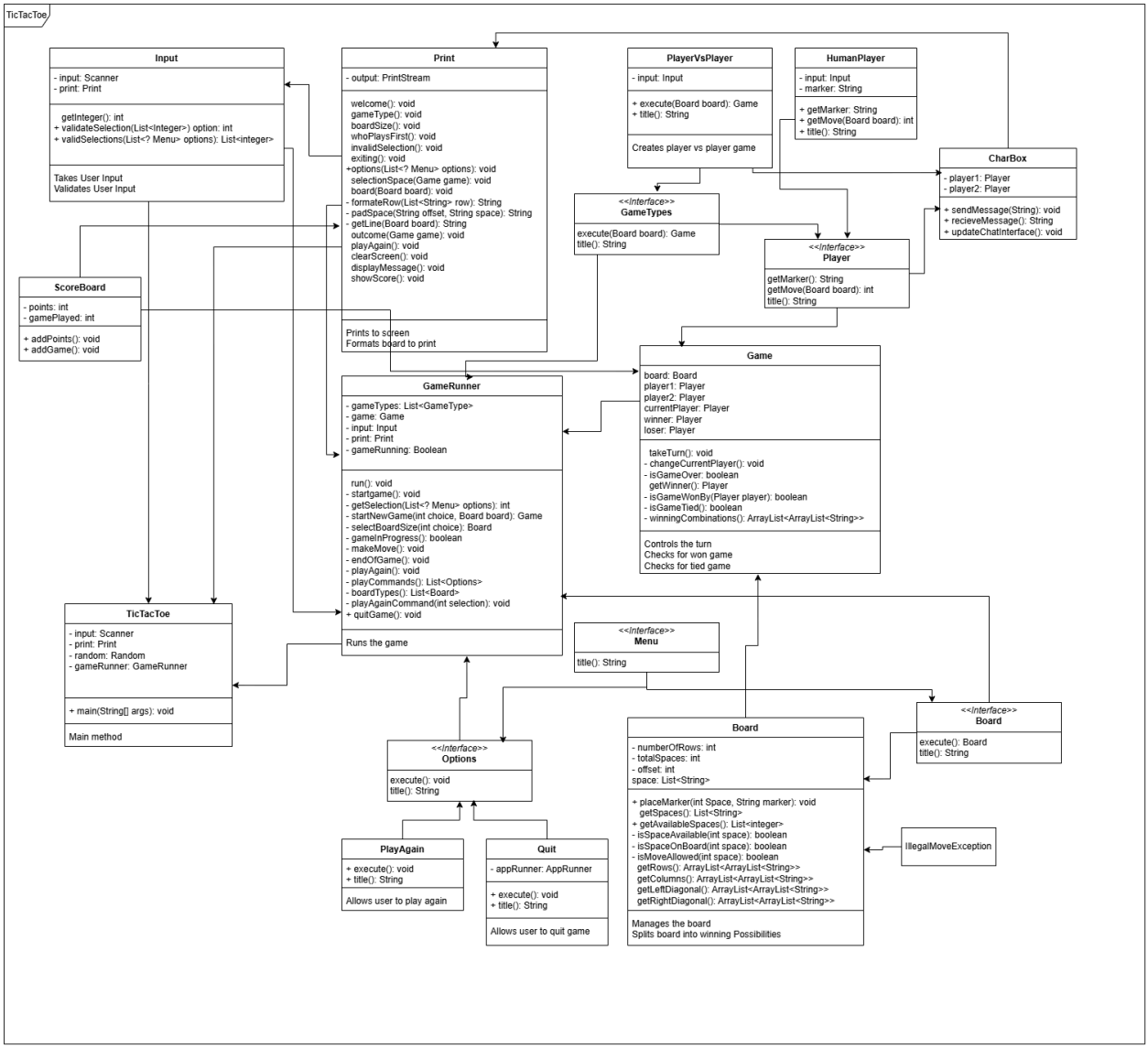


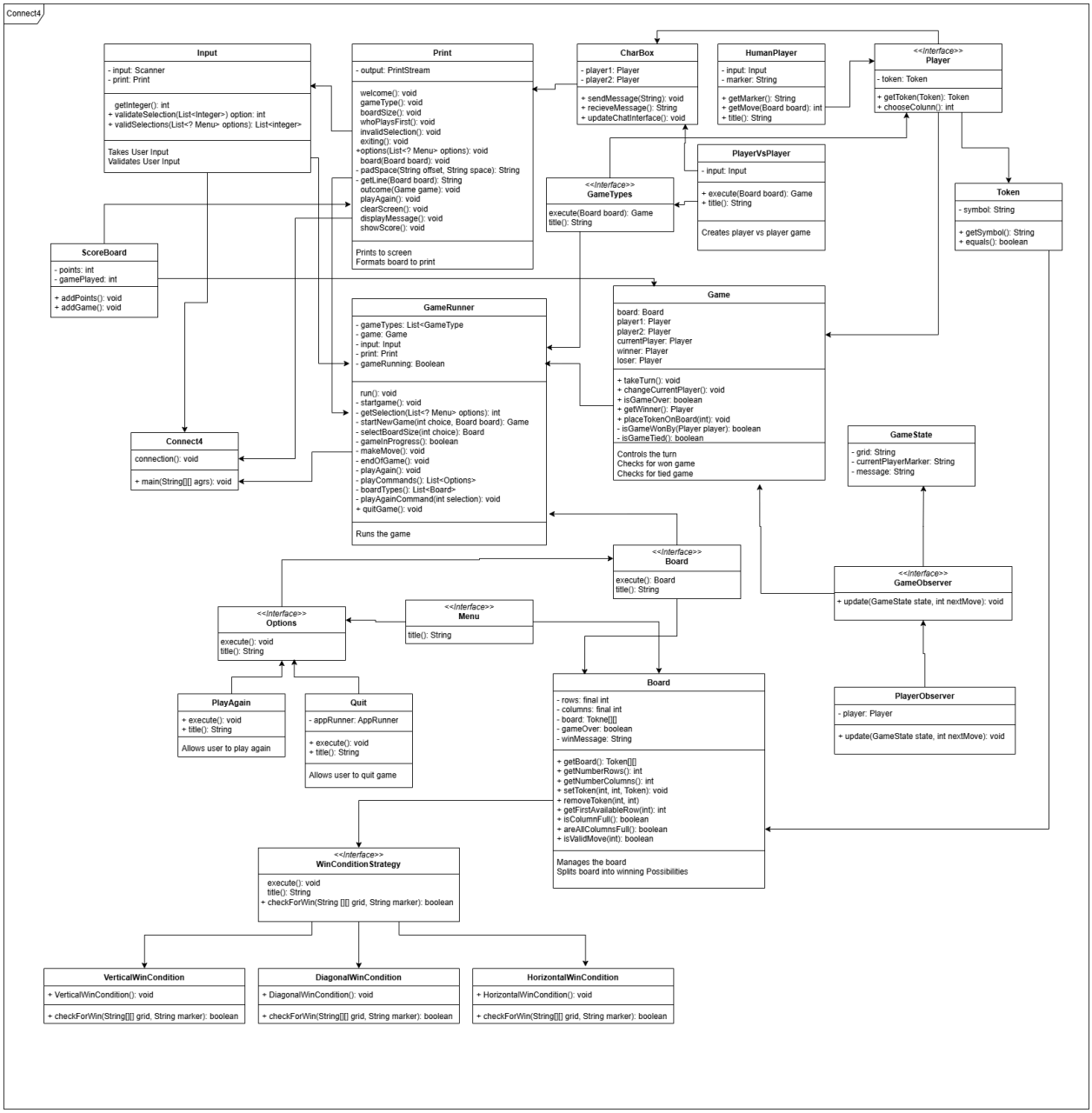
3

GUI Process:
When a square is clicked in the game.currentPlayer's Screen:
If associated BoardLocation object != currentSelection, perform first stage of screen update (i.e. deselect the square marked as selected and remove all move option markings from the screen in which the square resides). Then
If associated BoardLocation object is in game.moveOptions, perform game.makeMove(<associated-board-location>) then update both player screens (i.e. for each BoardLocation object in game.lastUpdatedLocations, change piece image in corresponding square and if game.lastMoveWasACapture, present new whitePlayer.capturedPieces and blackPlayer.capturedPieces as well as new whitePlayer.materialAdvantage and blackPlayer.materialAdvantage);
Otherwise, perform game.updateSelection(<associated-board-location>) then update the screen in which the square resides (i.e. mark the square associated with game.currentSelection as selected, and for each BoardLocation object in game.moveOptions, mark the associated square as a move option).

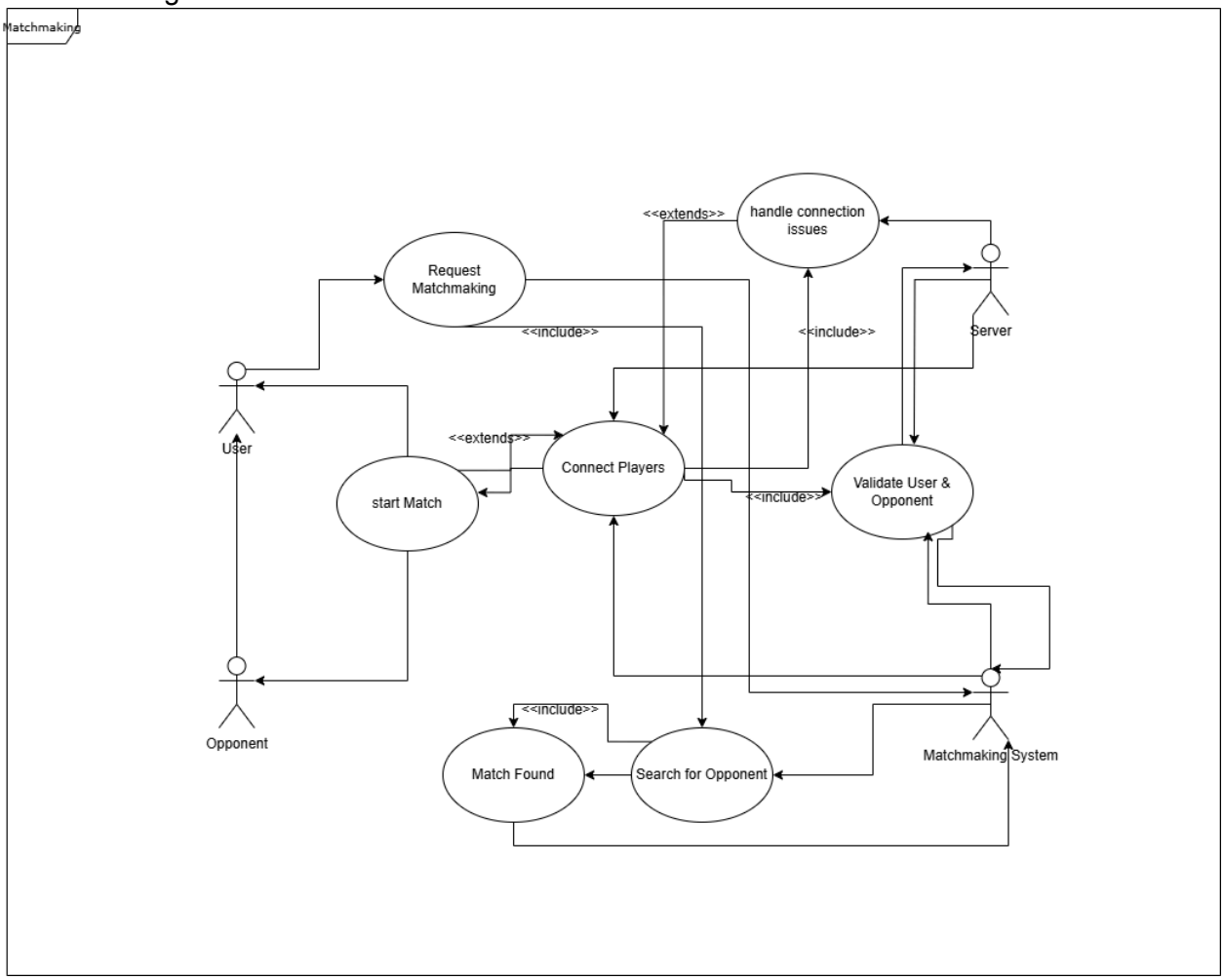
5

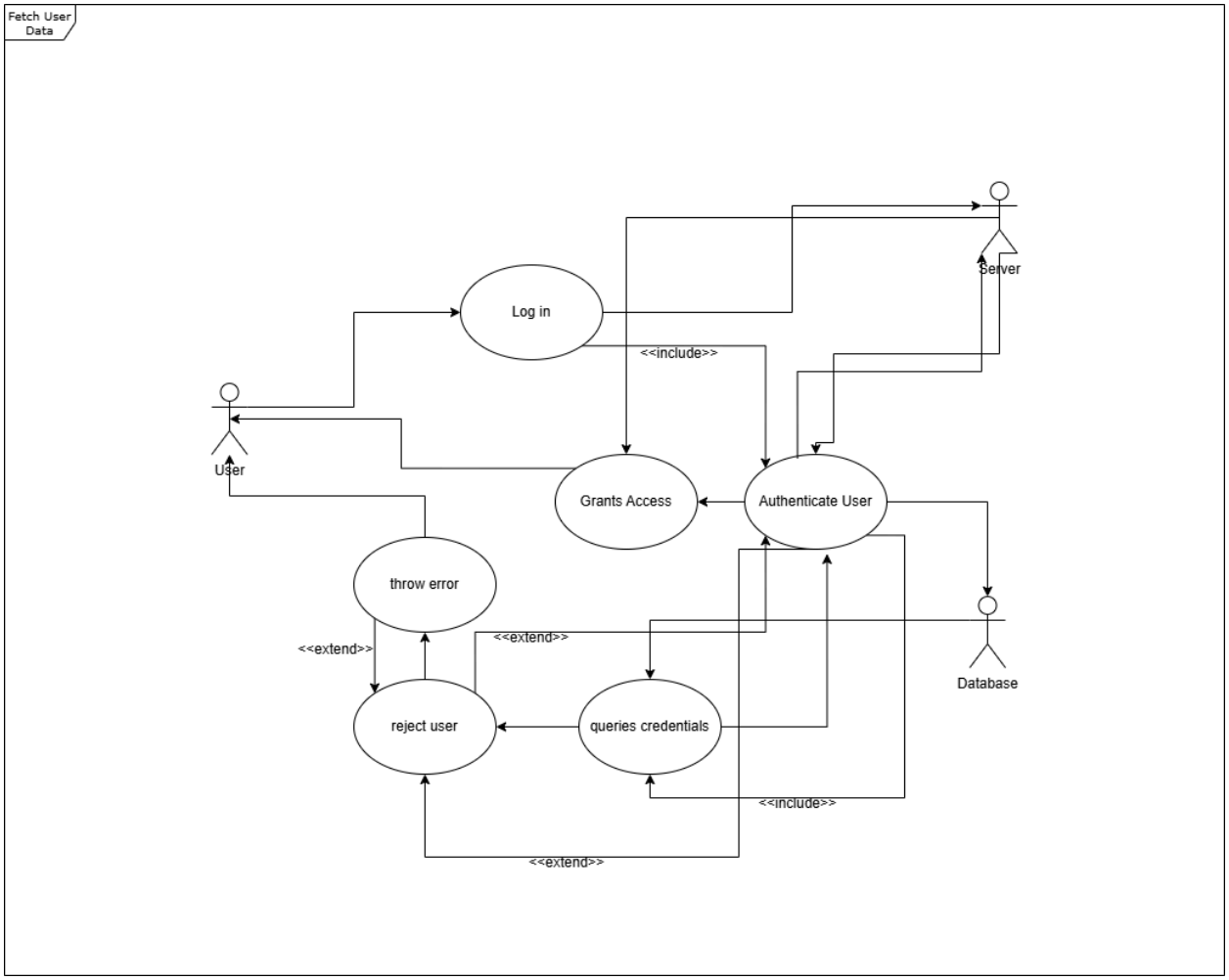
1. Does this need to be a separate interface if there is only one inheritor?
2. Piece requires Player (should show connection)
3. Please left-justify (difficult to follow when centred). Also consider bullet points or a list
4. How do players (white/black) link to a user's account?
5. Would recommend GUI is only updated once for the non-active player's turn (at the end of opponent's turn)

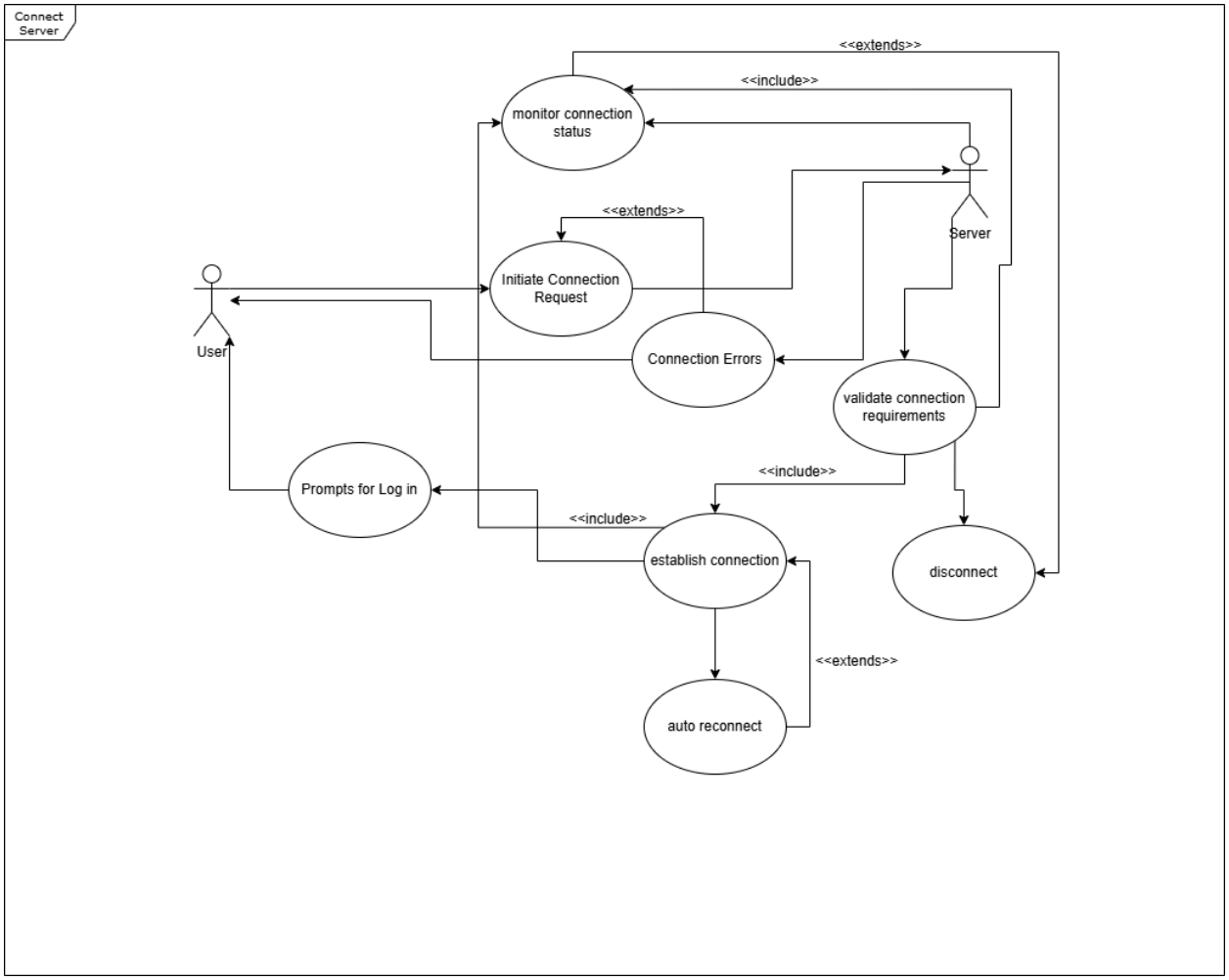

TicTacToe
Input
- input: Scanner
- print: Print
getInteger(): int
+ validateSelection(List<Integer>) option: int
+ validSelections(List<? Menu> options): List<integer>
Takes User Input
Validates User Input
Print
- output: PrintStream
welcome(): void
gameType(): void
boardSize(): void
whoPlaysFirst(): void
invalidSelection(): void
exiting(): void
+options(List<? Menu> options): void
selectionSpace(Game game): void
board(Board board): void
- formateRow(List<String> row): String
- padSpace(String offset, String space): String
- getLine(Board board): String
outcome(Game game): void
playAgain(): void
clearScreen(): void
displayMessage(): void
showScore(): void
Prints to screen
Formats board to print
PlayerVsPlayer
- input: Input
+ execute(Board board): Game
+ title(): String
Creates player vs player game
HumanPlayer
- input: Input
- marker: String
+ getMarker: String
+ getMove(Board board): int
+ title(): String
CharBox
- player1: Player
- player2: Player
+ sendMessage(String): void
+ recieveMessage(): String
+ updateChatInterface(): void
<<Interface>>
GameTypes
execute(Board board): Game
title(): String
<<Interface>>
Player
getMarker(): String
getMove(Board board): int
title(): String
ScoreBoard
- points: int
- gamePlayed: int
+ addPoints(): void
+ addGame(): void
Game
board: Board
player1: Player
player2: Player
currentPlayer: Player
winner: Player
loser: Player
takeTurn(): void
- changeCurrentPlayer(): void
- isGameOver: boolean
getWinner(): Player
- isGameWonBy(Player player): boolean
- isGameTied(): boolean
- winningCombinations(): ArrayList<ArrayList<String>>
Controls the turn
Checks for won game
Checks for tied game
GameRunner
- gameTypes: List<GameType>
- game: Game
- input: Input
- print: Print
- gameRunning: Boolean
run(): void
- startgame(): void
- getSelection(List<? Menu> options): int
- startNewGame(int choice, Board board): Game
- selectBoardSize(int choice): Board
- gameInProgress(): boolean
- makeMove(): void
- endOfGame(): void
- playAgain(): void
- playCommands(): List<Options>
- boardTypes(): List<Board>
- playAgainCommand(int selection): void
+ quitGame(): void
Runs the game
TicTacToe
- input: Scanner
- print: Print
- random: Random
- gameRunner: GameRunner
+ main(String[] args): void
Main method
<<Interface>>
Menu
title(): String
<<Interface>>
Options
execute(): void
title(): String
Board
- numberOfRows: int
- totalSpaces: int
- offset: int
space: List<String>
+ placeMarker(int Space, String marker): void
getSpaces(): List<String>
+ getAvailableSpaces(): List<integer>
- isSpaceAvailable(int space): boolean
- isSpaceOnBoard(int space): boolean
- isMoveAllowed(int space): boolean
getRows(): ArrayList<ArrayList<String>>
getColumns(): ArrayList<ArrayList<String>>
getLeftDiagonal(): ArrayList<ArrayList<String>>
getRightDiagonal(): ArrayList<ArrayList<String>>
Manages the board
Splits board into winning Possibilities
<<Interface>>
Board
execute(): Board
title(): String
IllegalMoveException
PlayAgain
+ execute(): void
+ title(): String
Allows user to play again
Quit
- appRunner: AppRunner
+ execute(): void
+ title(): String
Allows user to quit game

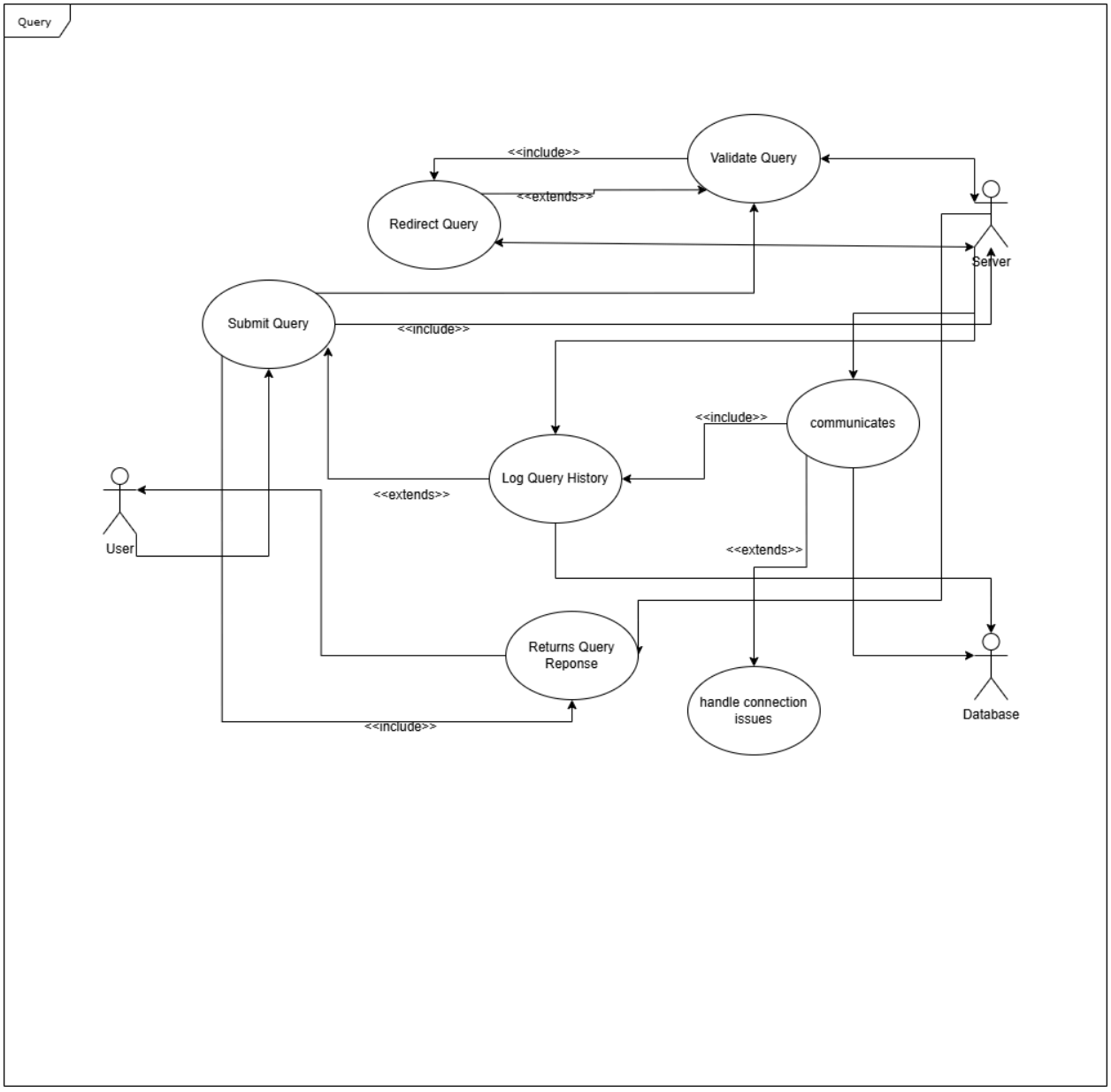

Connect4
Input
- input: Scanner
- print: Print
getInteger(): int
+ validateSelection(List<Integer>) option: int
+ validSelections(List<? Menu> options): List<integer>
Takes User Input
Validates User Input
Print
- output: PrintStream
welcome(): void
gameType(): void
boardSize(): void
whoPlaysFirst(): void
invalidSelection(): void
exiting(): void
+options(List<? Menu> options): void
board(Board board): void
- padSpace(String offset, String space): String
- getLine(Board board): String
outcome(Game game): void
playAgain(): void
clearScreen(): void
displayMessage(): void
showScore(): void
Prints to screen
Formats board to print
CharBox
- player1: Player
- player2: Player
+ sendMessage(String): void
+ recieveMessage(): String
+ updateChatInterface(): void
HumanPlayer
- input: Input
- marker: String
+ getMarker(): String
+ getMove(Board board): int
+ title(): String
<<Interface>>
Player
- token: Token
+ getToken(Token): Token
+ chooseColunn(): int
PlayerVsPlayer
- input: Input
+ execute(Board board): Game
+ title(): String
Creates player vs player game
<<Interface>>
GameTypes
execute(Board board): Game
title(): String
Token
- symbol: String
+ getSymbol(): String
+ equals(): boolean
ScoreBoard
- points: int
- gamePlayed: int
+ addPoints(): void
+ addGame(): void
GameRunner
- gameTypes: List<GameType
- game: Game
- input: Input
- print: Print
- gameRunning: Boolean
run(): void
- startgame(): void
- getSelection(List<? Menu> options): int
- startNewGame(int choice, Board board): Game
- selectBoardSize(int choice): Board
- gameInProgress(): boolean
- makeMove(): void
- endOfGame(): void
- playAgain(): void
- playCommands(): List<Options>
- boardTypes(): List<Board>
- playAgainCommand(int selection): void
+ quitGame(): void
Runs the game
Game
board: Board
player1: Player
player2: Player
currentPlayer: Player
winner: Player
loser: Player
+ takeTurn(): void
+ changeCurrentPlayer(): void
+ isGameOver: boolean
+ getWinner(): Player
+ placeTokenOnBoard(int): void
- isGameWonBy(Player player): boolean
- isGameTied(): boolean
Controls the turn
Checks for won game
Checks for tied game
Connect4
connection(): void
+ main(String[][] agrs): void
GameState
- grid: String
- currentPlayerMarker: String
- message: String
<<Interface>>
Board
execute(): Board
title(): String
<<Interface>>
GameObserver
+ update(GameState state, int nextMove): void
<<Interface>>
Options
execute(): void
title(): String
<<Interface>>
Menu
title(): String
PlayAgain
+ execute(): void
+ title(): String
Allows user to play again
Quit
- appRunner: AppRunner
+ execute(): void
+ title(): String
Allows user to quit game
Board
- rows: final int
- columns: final int
- board: Tokne[][]
- gameOver: boolean
- winMessage: String
+ getBoard(): Token[][]
+ getNumberRows(): int
+ getNumberColumns(): int
+ setToken(int, int, Token): void
+ removeToken(int, int)
+ getFirstAvailableRow(int): int
+ isColumnFull(): boolean
+ areAllColumnsFull(): boolean
+ isValidMove(int): boolean
Manages the board
Splits board into winning Possibilities
PlayerObserver
- player: Player
+ update(GameState state, int nextMove): void
<<Interface>>
WinConditionStrategy
execute(): void
title(): String
+ checkForWin(String [][] grid, String marker): boolean
VerticalWinCondition
+ VerticalWinCondition(): void
+ checkForWin(String[][] grid, String marker): boolean
DiagonalWinCondition
+ DiagonalWinCondition(): void
+ checkForWin(String[][] grid, String marker): boolean
HorizontalWinCondition
+ HorizontalWinCondition(): void
+ checkForWin(String[][] grid, String marker): boolean

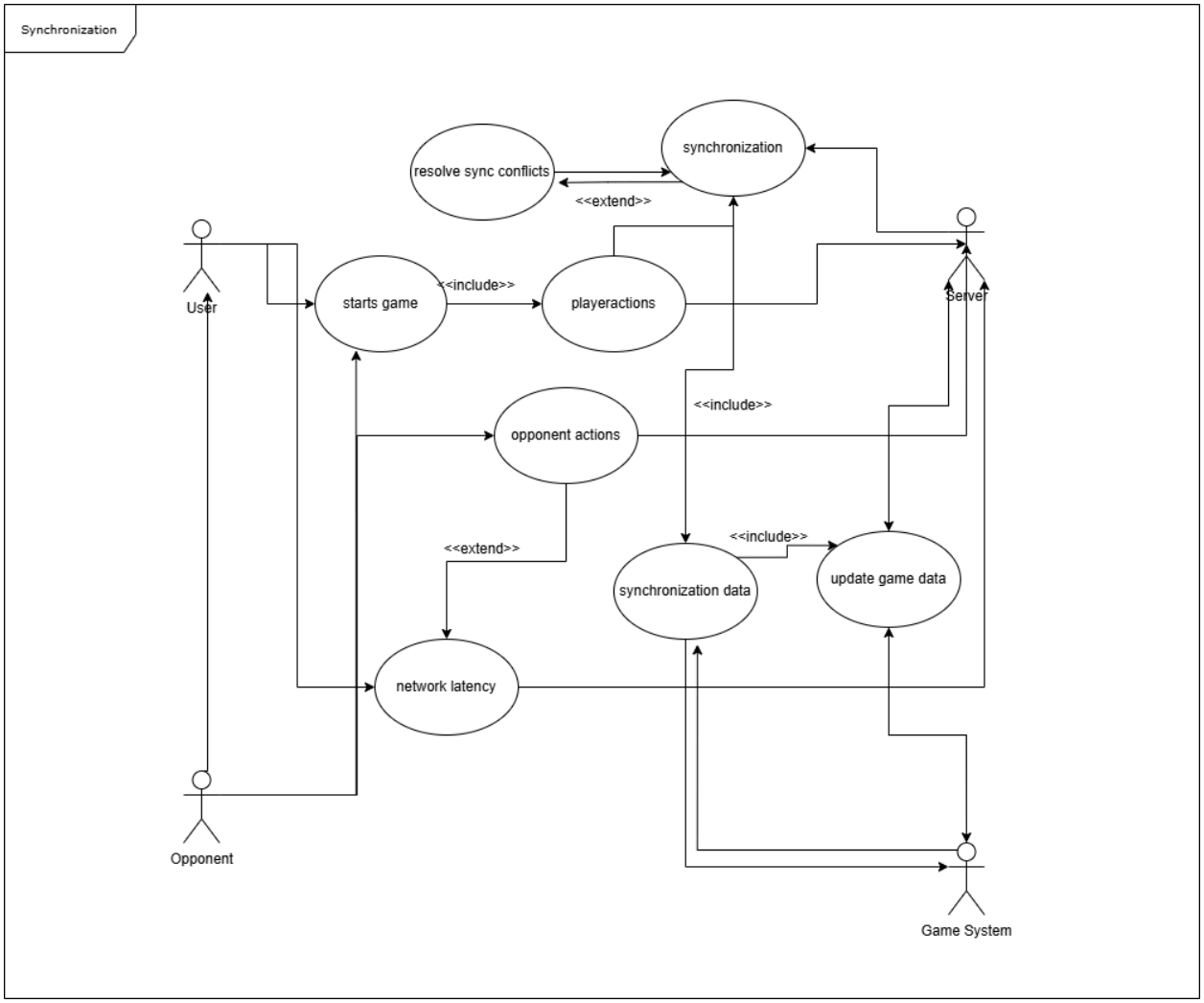
Use case diagrams

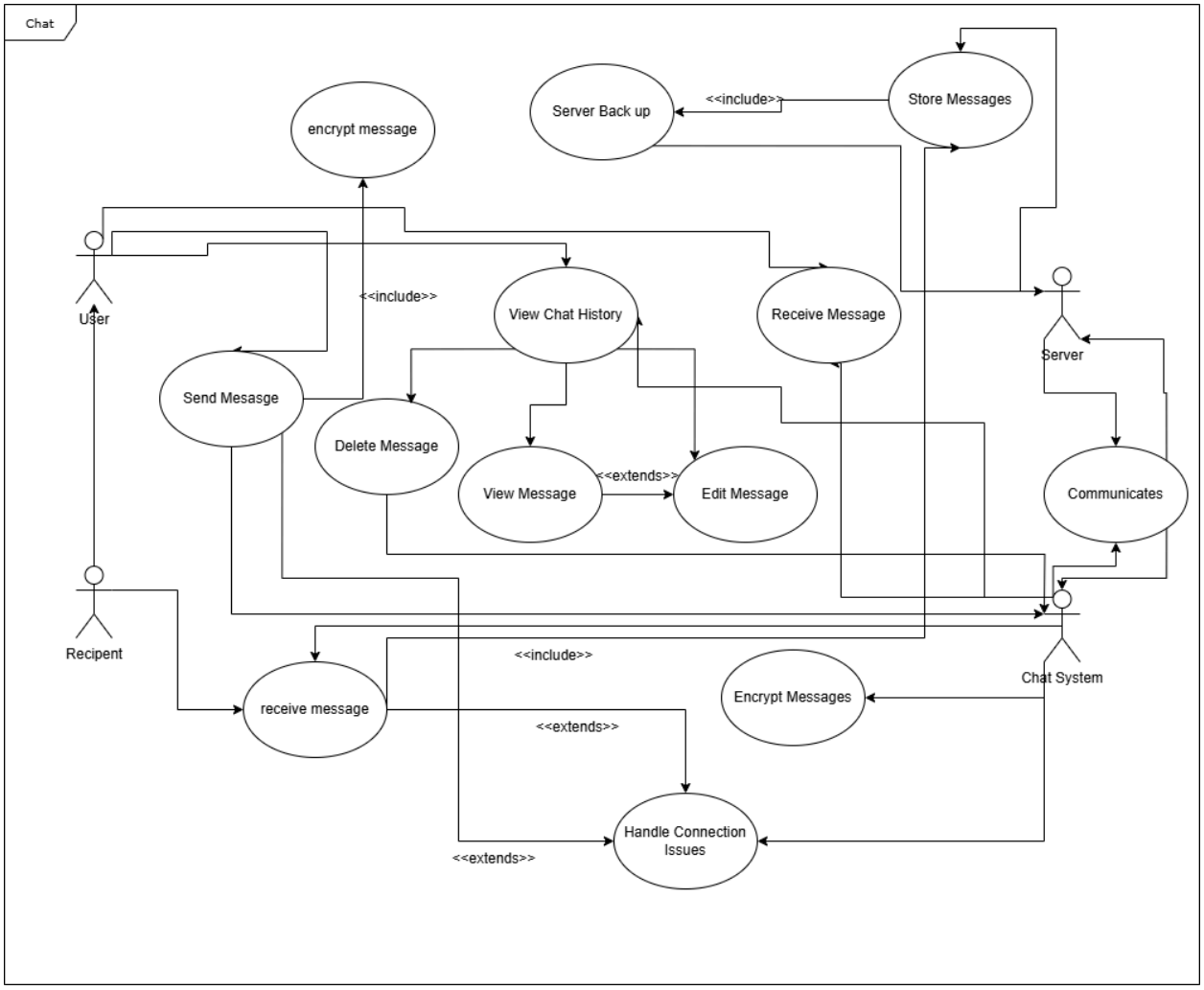


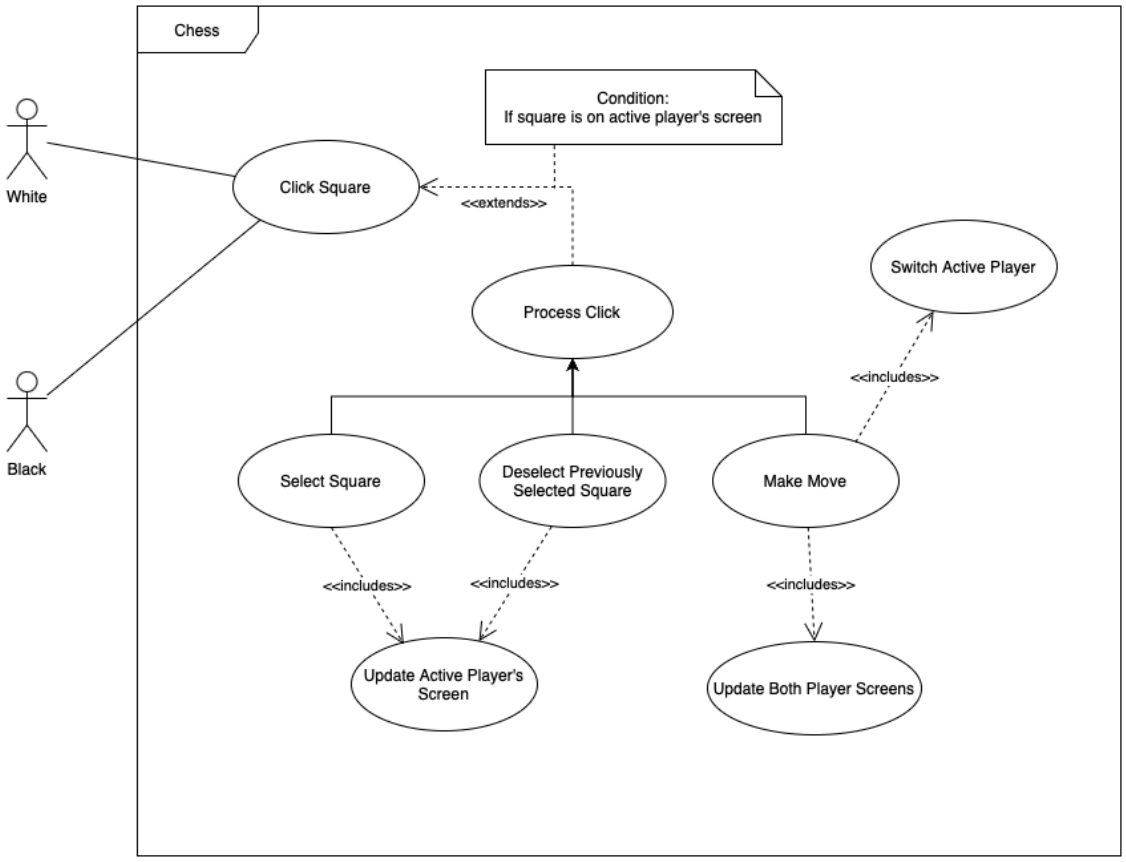

Fetch User Data
Server
Log in
<<include>>
User
Grants Access
Authenticate User
throw error
<<extend>>
<<extend>>
Database
reject user
queries credentials
<<include>>
<<extend>>

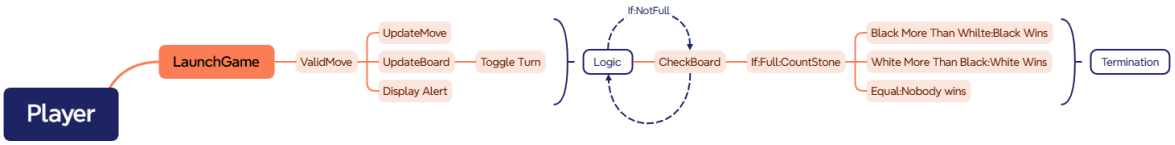

Connect Server
<<extends>>
<<include>>
monitor connection status
Server
<<extends>>
Initiate Connection Request
User
Connection Errors
validate connection requirements
Prompts for Log in
<<include>>
<<include>>
establish connection
disconnect
<<extends>>
auto reconnect

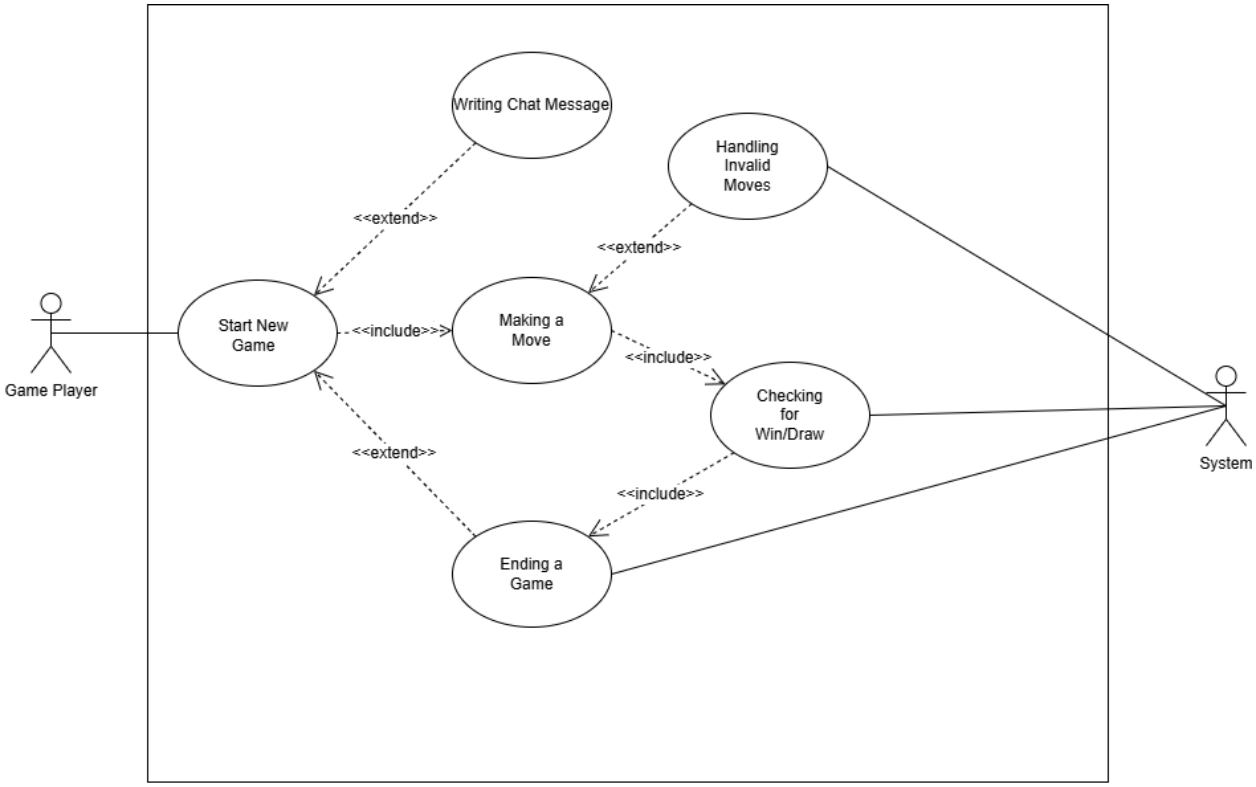

Query
Validate Query
<<include>>
<<extends>>
Redirect Query
Server
Submit Query
<<include>>
<<include>>
communicates
Log Query History
<<extends>>
User
<<extends>>
Returns Query Reponse
handle connection issues
Database
<<include>>

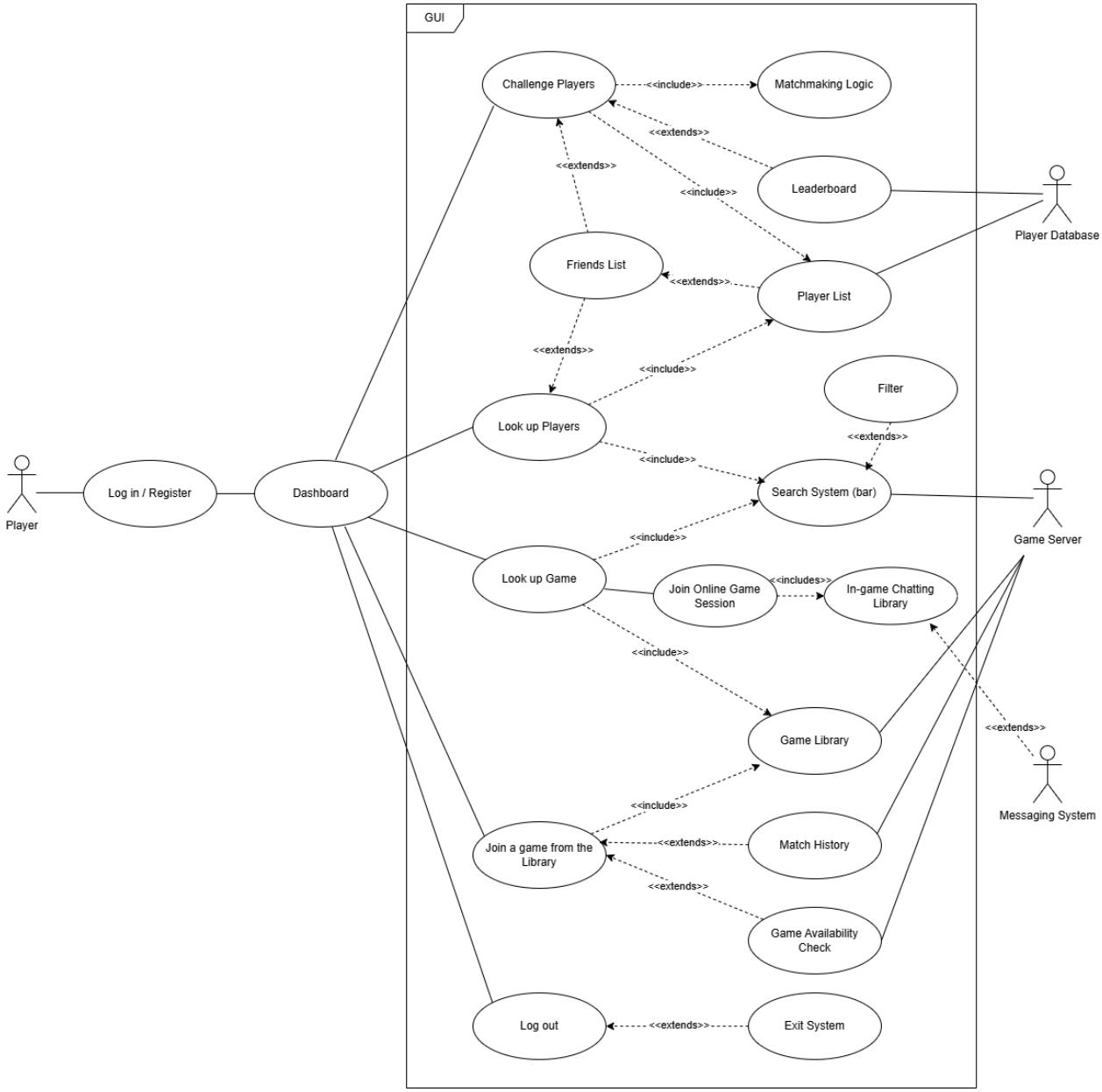

Synchronization
synchronization
resolve sync conflicts
<<extend>>
User
starts game
<<include>>
playeractions
Server
<<include>>
opponent actions
<<include>>
<<extend>>
synchronization data
update game data
network latency
Opponent
Game System


Chat
encrypt message
Server Back up
<<include>>
Store Messages
User
<<include>>
View Chat History
Receive Message
Server
Send Mesasge
Delete Message
View Message
<<extends>>
Edit Message
Communicates
Recipent
<<include>>
Chat System
receive message
Encrypt Messages
<<extends>>
Handle Connection Issues
<<extends>>


Chess
Condition:
If square is on active player's screen
White
Click Square
<<extends>>
Switch Active Player
Process Click
<<includes>>
Black
Select Square
Deselect Previously Selected Square
Make Move
<<includes>>
<<includes>>
<<includes>>
Update Active Player's Screen
Update Both Player Screens




Writing Chat Message
Handling Invalid Moves
<<extend>>
<<extend>>
Game Player
Start New Game
<<include>>
Making a Move
<<include>>
Checking for Win/Draw
System
<<extend>>
<<include>>
Ending a Game


GUI
Challenge Players
<<include>>
Matchmaking Logic
<<extends>>
<<extends>>
<<include>>
Leaderboard
Player Database
Friends List
<<extends>>
Player List
<<extends>>
<<include>>
Filter
Look up Players
<<extends>>
<<include>>
Search System (bar)
Game Server
Player
Log in / Register
Dashboard
<<include>>
Look up Game
Join Online Game Session
<<includes>>
In-game Chatting Library
<<include>>
<<extends>>
Game Library
Messaging System
<<include>>
Join a game from the Library
<<extends>>
Match History
<<extends>>
Game Availability Check
Log out
<<extends>>
Exit System

Use Case: Start Session

Primary Actor: Player
Goal in Context: The player begins a session to play a game.

Preconditions:

1. The player is connected to the internet.
2. The player has the platform installed (if necessary).

Trigger:

The player starts the game platform.

Scenario:

1. The player opens the platform.
2. Platform GUI starts.
3. Player logs into their account (authentication, profile, GUI, and network).

Postconditions:

1. Player can check stats, view the leaderboard, or start a game.

Exceptions:

1. Player does not have the platform installed (if necessary).
2. Player is not connected to the internet.

Priority: Essential

When Available: Once the platform is installed.
Frequency of Use: Whenever the platform is opened.
Channel to Actor: Platform, electronic device (computer, laptop, etc.).

Secondary Actors:

- GUI
- Authentication and Profile
- Network

Channels to Secondary Actors: Platform

Open Issues:

1. Does the platform require installation, or is it available on a web browser?

Use Case: Matching Players Together

Primary Actor: Player
Goal in Context: Connect the player to a suitable opponent.

Preconditions:

1. The player is connected to the internet.
2. The player is logged into their account.

Trigger:

The player selects which game they want to play.

Scenario:

1. The system finds all other players looking for a match (network).
2. The system selects an opponent with the closest skill rating (leaderboard and matchmaking).
3. The game begins (GUI and game logic).

Postconditions:

1. The player may now begin the game.

Exceptions:

1. The player is offline.
2. There are no available opponents to match up with.

Priority: Essential

When Available: Once the game is selected.
Frequency of Use: Often.
Channel to Actor: Platform, electronic device (computer, laptop, etc.).

Secondary Actors:

- Opponent
- Network
- Leaderboard and Matchmaking
- GUI
- Game Logic

Channels to Secondary Actors: Platform

Open Issues:

1. What if there are no online opponents within a reasonable skill range of the player?

Use Case: Game/Server Selection

Primary Actor: Player
Goal in Context: Matches the player to online opponents in the desired server and game.

Preconditions:

1. The player is connected to the internet.
2. The player is logged into their account.

Trigger:

The player wants to begin a game.

Scenario:

1. The player selects which server they want to play on (network and GUI).
2. The player selects which game they want to play (GUI).

Postconditions:

1. Player can now match up with an opponent.

Exceptions:

1. The player is not connected to the internet.

Priority: Essential

When Available: Whenever the platform is open and a game is not already in play. Frequency of Use: Often.
Channel to Actor: Platform, electronic device (computer, laptop, etc.).

Secondary Actors:

● Network

● GUI
Channels to Secondary Actors: Platform

Open Issues:

1. Will different servers be available on launch?

Use Case: Registering Gameplay

Primary Actor: Player
Goal in Context: Update gameplay and GUI for both players.

Preconditions:

1. The game has already begun.
2. The player is connected to the internet.
3. The player is logged into their account.

Trigger:

The player chooses what move they want to make.

Scenario:

1. The player makes a move (game logic).
2. GUI updates to show the move (GUI).
3. Opponent's GUI updates to show the player's move (network, GUI, game logic).

Postconditions:

1. Opponent can now make their move.

Exceptions:

1. Player does not make a valid move.
2. Player disconnects from the network.
3. Player closes the game/platform.

Priority: Essential

When Available: Whenever the game is in play.
Frequency of Use: Often.
Channel to Actor: Platform, electronic device (computer, laptop, etc.).

Secondary Actors:

- Opponent
- Network
- GUI
- Game Logic

Channels to Secondary Actors: Platform

Open Issues:

1. What will happen if either player leaves mid-game?

Use Case: Updating Profile

Primary Actor: Player
Goal in Context: After the game concludes, both players' profiles are updated to reflect new stats.

Preconditions:

1. Player is connected to the internet.
2. Player is logged into their account.
3. Player has completed at least one match.

Trigger:

Game has been completed.

Scenario:

1. Win/loss is added to both players' records (leaderboard, matchmaking, and network).
2. Both players' stats are updated according to the game outcome (leaderboard, matchmaking, and network).

Postconditions:

1. The player can check new stats on their profile.

Exceptions:

1. Either player leaves before the game is completed.

Priority: High

When Available: Right after the game is completed.
Frequency of Use: Often.
Channel to Actor: Platform, electronic device (computer, laptop, etc.).

Secondary Actors:

- Opponent
- Network
- Leaderboard and Matchmaking

Channels to Secondary Actors: Platform

Open Issues:

1. How are stats affected if a player leaves mid-game?
2. Can the game end in a draw?

Use Case: Updating Leaderboard

Primary Actor: Player
Goal in Context: Update the global leaderboard based on the outcome of a match.

Preconditions:

1. Player is connected to the internet.
2. Player is logged into their account.
3. Player has completed at least one match.

Trigger:

Match has been completed.

Scenario:

1. Players' individual stats are updated (leaderboard, matchmaking, and network).
2. Players' position on the leaderboard is updated based on new stats (leaderboard, matchmaking, and

   network).

Postconditions:

1. Player can check their new position on the leaderboard.

Exceptions:

1. Game doesn't finish, or the game result is inconclusive.

Priority: Medium

When Available: After the game is finished.
Frequency of Use: Often.
Channel to Actor: Platform, electronic device (computer, laptop, etc.).

Secondary Actors:

- Opponent
- Leaderboard and Matchmaking
- Network

Channels to Secondary Actors: Platform

Open Issues:

1. What statistics are used to determine the leaderboard? (e.g., win rate, number of games won).

Use Case: End Session

Primary Actor: Player
Goal in Context: Player ends the session.

Preconditions:

1. Game is not currently in progress.
2. Player is connected to the internet.
3. Player is logged into their account.

Trigger:

Player clicks the log-out button (GUI).

Scenario:

1. Player logs out of their account (authentication, profile, and GUI).
2. Player closes the platform (GUI).

Postconditions:

1. Player is disconnected from the platform.

Exceptions:

1. Player is currently playing a game.

Priority: Essential

When Available: Whenever the game is not in session.
Frequency of Use: Often.
Channel to Actor: Platform, electronic device (computer, laptop, etc.).

Secondary Actors:

- GUI
- Authentication and Profile
- Network

Channels to Secondary Actors: Platform

Open Issues:

1. Can the session be ended while the game is in progress?
2. Will a "save login info" feature be implemented so the user doesn't have to log out at the end of the

   session?

Registering/Creating New Account Primary Actor:

Game Player

Goal in Context:

Enable any person to create an account.

Pre-Conditions:

- **Networking is running as needed.**

- **It's a human player and not a bot that is trying to overload the server.**

Trigger:

● Player clicks the create new account after opening the platform.
Scenario:

1. **Player loads the platform through a web browser.**
2. **Player clicks "Create An Account" after being prompted to login.**
3. **Player inputs sufficient details for Username, Password, Verify Password &**

   Email.
4. **If inputted invalid details where passwords don't match the requirements or**

   if username already exists, it asks the user to perform actions again until

   valid details are put in.
5. **When player's details are valid, it creates account for the player and asks the**

user to login.

Post-Conditions:

- **The player has created an account with an username and passwords that meet the minimum requirements.**

- **The account details get added to the authentication database and thus allows the player to save progress in game and have a tracked identity in the platform.**

Exceptions:

- **Server malfunction due to overloading Priority:**

- **High – Essential for all parties in the game. When Available:**

- **First increment.**

Frequency of Use:

● Frequent – Couple times every day or couple times every minute depending on the popularity of the platform.

Channel to Actor:

● Click of a button using touchscreen or mouse click. Secondary Actors:

● Developers & Testers Channels to Secondary Actors:

● Raw Code Open Issues:

1. At what point should the server forcibly shut own the player(if the user is creating many a accounts using the same IP)?

User Login Primary Actor:

Game Player

Goal in Context:

Allow a player to login.

Pre-Conditions:

- **The system is accessible and online.**
- **The player has an already registered account.**

Trigger:

- **Player initiates login by clicking the "Login" button.**

- **Player selects "Guest Login"**

Scenario:

Login:

1. **Player opens the platform and is prompted to login**
2. **Player clicks the "login" button.**
3. **Players enters their username and password.**

4. **System validates their login credentials.**

5. **if credentials are valid, the player is granted access.**

6. **If credentials are invalid, an error message is displayed and the player is**

   prompted to try again.

7. **The system may temporarily lock the account if the player has multiple**

   failed attempts.

8. **If player does not have an account, they click "Create An Account" and are**

   redirected to the creating an account page.

Guest Login:

1. **Player clicks on "Guest Login"**

2. **System grants limited access, restricting certain featues like, saving**

   progress, or accessing leaderboards.

Post-Conditions:

- **After login, the player has access to their profile, and the game features.**

- **After guest login, the player has limited access.**

Exceptions:

- **Server downtime.**

- **Multiple incorrect credentials may result in a temporary lock.**

Priority:

● High. Essential for security, and also for user access control. When Available:

● First increment.

Frequency of Use:

● Frequent. Every time a player accesses the platform. Channel to Actor:

● Web interface via mouse click or touch interaction. Secondary Actors:

● Database
Channels to Secondary Actors:

● Code Open Issues:

1. A possible "Remember Me" option for easier access.
2. Should multifactor authentication be implemented?
3. What specific limitations should be placed on guest users?

User Logout Primary Actor:

Game Player

Goal in Context:

Allow a player to logout.

Pre-Conditions:

- **The system is accessible and online.**
- **The player is logged in.**

Trigger:

- **Player initiates logout by clicking the "Logout" button.**

- **Automatic logout due to inactivity.**

Scenario:

1. Player clicks the "logout" button.
2. System ends the session.
3. Player is redirected to the login page.

Post-Conditions:

● After logout, the session is ended, and the player must login again to access their profile.

Exceptions:

- **Server downtime.**

- **Server error prevents proper session termination.**

Priority:

● High. Essential for security, and also for user access control. When Available:

● First increment. Frequency of Use:

● Frequent. Everytime a player exits the platform. Channel to Actor:

● Web interface via mouse click or touch interaction. Secondary Actors:

● **Database**
Channels to Secondary Actors:

● **Code Open Issues:**

1. After how long should the system automatically logout due to inactivity? 2. Should there be a confirmation prompt?

Edit Profile Primary Actor:

Game Player

Goal in Context:

Allow user to make changes to profile when needed.

Pre-Conditions:

- **Networking is running as needed.**
- **Player has an existing profile and wants to make changes to it.**
- **Player is logged in.**

Trigger:

● Player clicks the edit profile button after logging in to existing account. Scenario:

1. **Player loads the platform through a web browser.**
2. **Player is able to log in to the system.**

3. **Player clicks "Edit Profile" after logging in.**

4. **Player is able to edit username or profile picture. They are also able to**

   change email address or password with proper verification.

5. **If invalid details detected the user performs actions again until valid details**

   are put in.

6. **When player's details are valid, it changes the desired information.**

Post-Conditions:

- **The player has successfully edited or changed their information.**

- **The database is updated as required.**

Exceptions:

● Server malfunction due to overloading Priority:

● Medium. When Available:

● Second Iteration. Frequency of Use:

● Only when desired by user. Channel to Actor:

● Click of a button using touchscreen or mouse click. Secondary Actors:

● User Database.
Channels to Secondary Actors:

● Code Open Issues:

1. Security measures for user verification? 2. Require password input to edit profile?

Multifactor Authentication (MFA) Primary Actor Game Player Goal in Context

● Requires users to provide an addtional verification step beyond standard username/password to increase account security

Pre-Conditions

- **the system is accessible and online**

- **the played has a registered account and a valid username/password**

- **MFA feature is enabled on the platform**

Trigger -PLayer attempts to log in with correct username and password, prompting the system to initiate the MFA process

Scenario

1. **Player enters valid username and password on the login screen.**
2. **System validates these credentials.**
3. **Upon successful validation, the system requests a second factor (e.g., code**

   sent to email, SMS, authenticator app).
4. **Player inputs the one-time code or approves the request via an authenticator**

app.

5. System verifies the code or approval.

6. If the second factor is correct, the player is fully logged in.

7. If the second factor is incorrect or not entered within a certain time frame,

   the system denies access and prompts the user to try again.

Post-Conditions

- Upon successful MFA, the player gains access to their profile and game features.

- If MFA fails, the player remains locked out until they provide the correct second factor.

Exceptions

- Delivery delay or failure for the one-time code (e.g., email or SMS not received).

- **Authenticator app malfunction or time-synchronization errors causing valid codes to be rejected.**

Priority

- **Medium**

When Available

- **Third iteration Frequency of Use:**

- **Every login attempt if MFA is enforced by default.**

- **Optional if the player chooses to enable it.**

Channel to Actor

● Click of a button using touchscreen or mouse click. Secondary Actor

● External authentication service (e.g., authenticator app) Open Issues

- **Handling of backup authentication methods when the user cannot access their primary second factor?**

- **Storage and management of user MFA preferences?**

Viewing Match History Primary Actor Game Player Goal in Context

● To enable the user to view his/her match history and other stats related to it Pre-Conditions

- **the system is accessible and online**

- **the user is registered and has a username**

Exceptions The user has not played any previous match is using the platform for the first time.

Scenario

1. **Player enters valid username and password on the login screen.**

2. **System validates these credentials.**

3. **The player clicks on match history**

4. **the match history is visible to the player through a user interface.**

5. **if the player has no match history, the UI shows a 0/NA or an error window**

   letting the user know the absence of the match history

Post-Conditions

● The player is able to view the matches, their serial number and the stats related to the match.

Priority

● High- Match History is crucial in mathcmaking and keeping the stats of the player

When Available

● first iteration Frequency of Use:

- **Optional: if the player wants to use it**
- **high for the system for the process of matchmaking**

Channel to Actor

● Click of a button using touchscreen or mouse click. Secondary Actor

● Database Open Issues

- **what stats should the player be able to see?**
- **should the stats that the system can utilize be the same as the stats that the**

player can see.

Failed Login Attempt Primary Actor

● Game Player Goal in Context

● Highlight the scenario where a player fails to provide valid login credentials and the resulting system behaviour

Pre-conditions

- **The system is accessible and online.**
- **The player has a registered account.**

- **The player is attempting to log in.**

Trigger

● Player enters incorrect login credentials (e.g., username or password) during the login process.

Scenario

1. Player clicks the "Login" button.
2. Player inputs their username and password incorrectly.
3. System checks the credentials against the stored user data.
4. System detects invalid credentials.
5. System displays an error message indicating the login has failed.

Post-Conditions

- **The player is not granted access to the system if credentials remain invalid.**

- **The system may lock the account if the failed attempts exceed the allowed**

number.

Exceptions

● Data inconsistency or database error causing valid credentials to be rejected.

Priority

● High Availability

● First Iteration Frequency of Use

● Occasional – Occurs only when the user enters incorrect credentials. Channel to Actor

● Web interface via mouse click or touch interaction. Secondary Actors

● Database
Channel to Secondary Actor

● Code Open Issues

1. How many failed attempts should be allowed before locking the account? 2. Should the lock duration be fixed or dynamic based on repeated offenses?

Password Reset Primary Actor:

Game Player

Goal in Context:

Allow a player to reset their password if they forget it.

Pre-Conditions:

- **The system is accessible and online.**
- **The player has a registered account.**

● The player has access to the registered email. Trigger:

● Player clicks on "Forgot Password" on the login page. Scenario:

1. **Player clicks on "Forgot Password"**

2. **System prompts the player to enter their registered email address.**

3. **Player enters the email and submits a request.**

4. **System verifies if the email is registered to an account.**

5. **If the email is valid. the system sends a password reset link to the email.**

6. **Player clicks on the password reset link. Which leads them to a password**

   reset page.

7. **Player enters a new password.**

8. **System verifies the new password against security requirements.**

9. **If valid, the password is updated and the player is notified.**

10. **Player can now login using the new password.**

Post-Conditions:

- **The password is updated, and the player is notified.**
- **System prevents the old password from being used.**

Exceptions:

- **Email entered does not match a registered email.**
- **Reset link expires after a certain time.**
- **The new password does not meet security requirements.**
- **System fails to send the reset email.**

Priority:

● High. Essential for account recovery. When Available:

● **First increment.**

Frequency of Use:

● **Medium. Only for when players forget their passwords. Channel to Actor:**

● **Web interface via mouse click or touch interaction. Secondary Actors:**

● **Database**
Channels to Secondary Actors:

● **Code Open Issues:**

1. How long should the reset link remain valid?
2. Should MFA be required?
3. Should there be a limit to the number od resets with in a certain timeframe?

Delete Account **Primary Actor:**

Game Player

Goal in Context:

Enable any person to delete their account from the profile management page.

Pre-Conditions:

- **Network is running as needed.**
- **Person is logged into their account**

Trigger:

● Player clicks the delete account button from their profile management page. Scenario:

1. **Player logs in to their account.**
2. **Player goes the profile management page.**
3. **Player clicks on the delete account button.**
4. **Player gets asked to confirm if they want to delete account.**
5. **If players confirms deletion, they are asked to input their password again and**

if the passwords matches with the database then player is officially logged and account gets deleted

Post-Conditions:

- **The person gets logged out of the account and lands to the homepage.**

- **The person gets a sidebar notification saying account deleted.**

Exceptions:

- **database/server malfunction. Priority:**

- **Mid – not essential to the function of the platform. When Available:**

- **Second increment. Frequency of Use:**

- **moderate – once in a while. Channel to Actor:**

● Click of a button using touchscreen or mouse click under the profile management page.

Secondary Actors:

● Developers, Testers, Database Channels to Secondary Actors:

● Code Open Issues:

1. Should we let players recover account during a period of time if it's already deleted from database?

View Player Profile Primary Actor:

Game Player

Goal in Context:

Allow user to view other players profile.

Pre-Conditions:

- **Networking is running as needed.**
- **Player has an existing profile.**
- **Player is logged in and has started a game.**
- **Player wants to view opponent profile.**

Trigger:

● Player clicks on the opponents name/ profile picture. Scenario:

1. **Player loads the platform through a web browser.**

2. **Player is able to log in to the system.**

3. **Player selects a game from the library and is matched with an opponent.**

4. **Player wishes to view opponent information.**

5. **Player clicks on the opponent username or profile picture.**

6. **Player is able to see information like ranking, games played etc.**

7. **Player is able to successfully exit out of the opponent profile and can**

   continue the game as desired.

Post-Conditions:

● The player has successfully viewed other player's profile.
Exceptions:

- **Server malfunction.**

- **Malfunction in code/matchmaking.**

Priority:

● Medium. When Available:

● Second Iteration. Frequency of Use:

● Only when desired by user. Channel to Actor:

● Click of a button using touchscreen or mouse click. Secondary Actors:

- **Database**

Channels to Secondary Actors:

- **Code Open Issues:**

- **Extent of information the player is able to see?**

Use Case: View Other Players Profile/Statistics
Iteration: 1
Primary Actor: Game Player
Goal in Context: Allow a player to view detailed information about other players' profiles, game statistics, and performance metrics.

Preconditions:

The system is accessible and online.

Player is logged into the platform.

The player whose profile is being viewed has a public profile or appropriate privacy

settings.
Trigger: Player selects another player's username from a leaderboard, match history, or search results.
Scenario:

1. Player finds another player through leaderboard, search function, or match history.
2. Player selects/clicks on the other player's username.
3. System retrieves the selected player's profile data and statistics from database.
4. System displays the profile page showing the player's: a. Username and profile

information b. Game-specific statistics (wins, losses, draws) c. Ranking and rating

across di□erent games d. Recent match history e. Achievements and badges

5. Player can toggle between di□erent statistical views (by game type, by time period).

6. Player can select the option to challenge this player to a game if desired.

Post conditions:
Player has viewed the requested profile and statistics.

Exceptions:

1. Private Profile: If the selected player has set their profile to private, system displays

   limited information or a privacy notice.

2. Data Unavailable: If player has no match history or statistics, system displays

   appropriate message indicating no data available.

3. System Error: If database connection fails, system displays error message and

   suggests trying again later.

Priority: Medium – Important for social and competitive aspects of the platform. When Available: First increment.
Frequency of Use: High – Players will regularly check other players' statistics for competitive analysis.
Channel to Actor: Click of a button using touchscreen or mouse click. Secondary Actors: Database System
Channel to Secondary Actors: Data retrieval interface.
Open Issues:

Should players be able to hide certain statistics from public view?

Use Case: Challenge Player from Friends List
Iteration: 1
Primary Actor: Game Player
Goal in Context: Allow a player to directly challenge a friend to a game without a□ecting either player's ranking.

Preconditions:

The system is accessible and online.

Player is logged into the platform.

Player has at least one friend added to their friends list.

The friend is currently online and available. Trigger: Player selects a friend from their

friends list and clicks "Challenge to Game". Scenario:

1. Player navigates to their friends list in the platform.

2. System displays all online friends with their current status (available, in-game, busy).
3. Player selects a specific friend who is marked as available.
4. Player chooses "Challenge to Game" option.
5. System presents game selection menu with available board games.
6. Player selects desired game type (e.g., Chess, Connect Four).
7. Player selects "Friendly Match" option to indicate the game won't a□ect rankings.
8. Player confirms and sends challenge request.
9. System delivers notification to friend.
10. Friend receives and accepts challenge.
11. System initializes the game session between both players.

Post conditions:
Both players are placed in the same game session with selected parameters.

Match is recorded in the system as a "Friendly Match" that doesn't a□ect player

rankings.

Players' match history is updated to include the friendly match. Exceptions:

1. Friend Unavailable: If the selected friend changes status to unavailable before challenge is sent, system notifies player and prevents challenge.
2. Challenge Declined: If friend declines challenge, system notifies challenger and returns to friends list.
3. Challenge Timeout: If friend doesn't respond within 2 minutes, challenge expires and system notifies challenger.

Priority: Medium – Important for social aspects but not critical to core gameplay. When Available: First increment.
Frequency of Use: Moderate – Will be used regularly by players who prefer playing with friends.

Channel to Actor: Click of a button using touchscreen or mouse click. Secondary Actors: Friend (Challenged Player)
Channel to Secondary Actors: Game client interface, notification system. Open Issues:

1. Should players be able to spectate friendly matches between other players?
2. Should there be an option to convert a friendly match into a ranked match if both players agree?
3. Should friendly matches have di□erent rules or settings available compared to

ranked matches?

Use Case: Find Matchmade Opponent
Iteration: 1
Primary Actor: Game Player
Goal in Context: Enable a player to be matched against another player of similar skill level for a selected game.

Preconditions:

The system is accessible and online.

Player is logged into the platform.

Player has completed enough games to have an established skill rating.

Trigger: Player chooses a game type and selects 'Find Opponent'. Scenario:

1. Player chooses a game type and selects 'Find Opponent'.
2. System retrieves player's current skill rating and game history from database.
3. System searches for other online players who are also seeking matches for the same game.
4. System identifies players within an appropriate skill range (±100 rating points).
5. System selects the best match based on skill similarity, connection quality, and wait

time.

6. System notifies both players that a match has been found.
7. Both players confirm readiness within 30 seconds.
8. System initializes the game session with both players.
9. Game begins with standard rules and settings.

Post conditions:

Both players are placed in the same game session.

Match is recorded in the system for future skill calculations.

Players' status is changed to "In Game".

Exceptions:

1. No Suitable Opponent: If no suitable opponents are found within 45 seconds,

system expands search criteria to include wider skill range.

2. Search Cancellation: If player cancels search, system removes player from matchmaking queue.

3. Readiness Confirmation Failure: If matched player doesn't confirm readiness, system cancels match setup and searches again.

Priority: High – Essential for the matchmaking functionality of the platform.
When Available: First increment.
Frequency of Use: Very frequent – Will be used constantly as players seek matches.
Channel to Actor: Click of a button using touchscreen or mouse click.
Secondary Actors: Matchmaking System, Other Players
Channel to Secondary Actors: Game client interface, network communication. Open Issues:

1. How quickly should the skill range expand if matches aren't found?

2. Should players be able to set preferences for opponents beyond skill level?

3. How to handle disconnections during the matchmaking process?

Use case: Player can view Game Leaderboard
Iteration: 1
Primary actor: Player
Goal in context: To allow the player check their regional/global ranking and able to see their rank with other players

Preconditions:

The player has an active and verified account

The player has played all of their placement game(s) and received a ranking

The leaderboard system is active and accessible

Trigger: The player select the leaderboard button from the game menu Scenario:

1. The player selects the Leaderboard option from the game menu.
2. The player can select the filter to display the local regional leaderboard or the global

   leaderboard
3. The system retrieves and displays the top 100 ranked players based on Elo/MMR.
4. The player sees their current rank, statistics and match history.

Exceptions:
If the leaderboard fails to load, the system displays an error message.

If the player hasn't played any ranked matches, the system will display an unranked status

Priority: High
When available: First increment
Frequency of use: Frequent as players like to check their ranking Channel to actor: When leaderboard button in menu is selected Secondary actors: Server database (fetching and updating leaderboard) Channel to secondary actors: API request to server database
Open issues: None

Use case: Player viewing their own statistics
Iteration: 1
Primary actor: Player
Goal in context: To allow the player to view their personal profile, including their game statistics, ranking, match history, and highest ranked achieved

Preconditions:

The player has an active and verified account

The player has played all of their placement game(s) and received a ranking

Trigger: The player select their profile from the game menu Scenario:

The player clicks on their profile from the game menu

The system retrieves the player's stored profile data from the database.

The system displays key information, including:

- Username and profile picture - Current ranking and rank title

- W/L ratio

- Recent match history - Total games played

The player clicks on any one of the options above and each options will display its own features (i.e. when clicked on recent match history it will

show the player and opponent name and ranking, results, match duration)

Exceptions:

If there is a server issue or missing data, the system displays an error message and

prompts the player to retry

If the player hasn't played any ranked matches but has played casual matches,

the system will display an unranked status and statistics for casual matches will

be provided and vice versa Priority: High

When available: First increment
Frequency of use: Frequent as players like to check their profile as they want to check for improvements
Channel to actor: When profile in game menu is selected
Secondary actors: Server database (fetching and updating profile data)

Channel to secondary actors: API request to server database Open issues: None

Use case: Rematch Request
Iteration: 1, last modification: Mar 7 by Min Oh
Primary actor: Player
Goal in context: To allow the player to send a rematch request to their most recent opponent
Preconditions:

The player has an active and verified account

The player just completed a match

The opponent is still online and agrees to the rematch

Trigger: The player selects “Request Rematch” after game ends Scenario:

1. After finishing a game the request rematch prompt appear under the continue prompt
2. The player selects rematch request
3. The system sends a rematch invitation to the opponent.
4. The opponent can accept or decline.
5. If accepted, the system starts a new game

Exceptions:

If the opponent is no longer online the system will prompt a message “Your opponent is no longer online.”

If the opponent refused the rematch then the system will prompt a message “Rematch Declined” Priority: High

When available: First increment

Frequency of use: Frequent as players like to check their profile as they want to check for improvements
Channel to actor: In game screen after a finished match
Secondary actors: Server database (fetching and updating info from most recent match) Channel to secondary actors: API request to server database
Open issues: None

Actor Actions

User selects matchmaking option.

Actor accepts the match.

Actor Actions

User System Flows

CONNECT WITH PLAYERS IN MATCHMAKING

Name: Connect with Players in Matchmaking
Requirement ID: ID01
Actors: User, Opponent, Matchmaking System, Server
Pre-conditions: User is connected to the server and has an active matchmaking request. **Trigger:** User searches for a match.
Post-conditions: User is successfully connected to an opponent and enters a game session.

Main Success Path (Primary Flow)

System Response

Client receives matchmaking request and searches for an appropriate match based on skill level and availability.

Server finds a match and notifies both players.
Game session is initialized, and players are connected.

Alternate Path A1

System Response

Actor rejects the match. Goes back to main menu/application screen. # Exception Path E1

Exception System Response

No suitable match is found immediately. User is placed in a waiting queue. Server encounters an issue while matching players. User receives an error message.

User receives an error message and matchmaking request is canceled.

Scenario

Scenario

User enters matchmaking but disconnects before finding an opponent.

Server removes the user from the matchmaking queue.

Unable to matchmake error message given to actor.

Actor Action

Can requeue again.

Can requeue again.

DATA RETRIEVAL OF USER FROM SERVER

Name: Data Retrieval of User from Server
Requirement ID: ID02
Actors: User, Server, Database
Pre-conditions: User is connected to the server and authenticated. **Trigger:** User requests to view their profile or leaderboard. **Post-conditions:** User data is retrieved and displayed on the client side.

Main Success Path (Primary Flow)

Actor Actions System Response

User opens app and logs in. Client sends request to server for user data. Server retrieves data from the database. Server sends the data back to the client.

User reviews their profile and data.

REAL-TIME GAME SYNCHRONIZATION

Name: Real-Time Game Synchronization
Requirement ID: ID03
Actors: User, Server, Opponent, Game System
Pre-conditions: User is connected to the server and in an active game session. **Trigger:** A player makes a move.
Post-conditions: Game state updates are synchronized across all players' clients.

Main Success Path (Primary Flow)

Actor Actions System Response

Player makes a move. Client sends move data to the server.
Server validates and updates the game state.

Server broadcasts the updated game state to all clients. Other players see the new game state in real time.

CHAT

Name: Chat
Requirement ID: ID04
Actors: User, Recipient, Chat System, Server

Pre-conditions: User is connected to the server and has an active matchmaking request **Trigger:** User types and sends a chat message.
Post-conditions: Message is successfully delivered to the recipient(s).

Main Success Path (Primary Flow)

Actor Actions System Response

User types a message. Client sends message to the server.
Server routes the message to the intended recipient(s).

Recipient(s) receive and display the message. Alternate Path A1

Actor Actions System Response

Actor does not type a message System waits for any message from the actor.
Exception Path E1

Exception System Response

Network failure occurs. User receives an error message and message is not sent.

Scenario

Scenario System Response

Recipient is offline. Server queues the message for later delivery. User sends a message but experiences lag. No response due to latency issue.

REDIRECT CLIENT QUERIES

Name: Redirect Client Queries
Requirement ID: ID05
Actors: User, Server, Database
Pre-conditions: User is connected to the server.
Trigger: User initiates an action.
Post-conditions: The request is processed and directed to the appropriate system.

Main Success Path (Primary Flow)

Actor Actions System Response

User performs an action. Client sends the request to the server.
Server routes the request to the relevant system.

The system processes the request and returns a response. User receives confirmation or the requested information.

Alternate Path A1

Actor Actions System Response

User does not perform any query System waits for any query from the user.

Exception Path E1

Exception System Response

Server fails to process the request. User receives an error message. Scenario

Scenario System Response

Server fails to route the query. Server error - Unable to process request error thrown. Database is down or unavailable. Profile data unavailable. Please try again later. Chat message fails to deliver. Message failed to send.

CONNECT TO SERVER

Name: Connect to Server
Requirement ID: ID06
Actors: User, Server, Authentication System
Pre-conditions: User has an internet connection, application installed, met system requirements, and followed instructions.
Trigger: User launches the application and tries to connect.
Post-conditions: User is successfully connected to the server and can access the platform and all features and games.

Main Success Path (Primary Flow)

Actor Actions System Response

User opens app. Client sends request to server. Server authenticates the request.

Server maintains a connection and sends response back. User gains access to the system.

SESSION RECONNECTION

Name: Session Reconnection
Requirement ID: ID07
Actors: User, Server, Game System

Pre-conditions: User was previously connected to a game session but got disconnected due to network issues.
Trigger: User attempts to reconnect to the game.
Post-conditions: User successfully rejoins the active game session without data loss.

Main Success Path (Primary Flow)

Actor Actions System Response

User loses connection. Server detects disconnection and stores session state. User reconnects to the game. Server verifies session ID and checks match status. Server restores user's state. User resumes gameplay from where they left off.

Alternate Path A1

Actor Actions System Response

User takes too long to reconnect. Server removes the user from the session. User must start a new match. Matchmaking system finds a new opponent.

Exception Path E1

Exception System Response

Game session has already ended. User is notified that the match is no longer available.

LATENCY COMPENSATION & NETWORK STABILITY

Name: Latency Compensation & Network Stability
Requirement ID: ID08
Actors: User, Opponent, Server, Game System
Pre-conditions: Users are actively playing a match, but network conditions fluctuate. **Trigger:** A player's network latency increases beyond an acceptable threshold. **Post-conditions:** Gameplay remains smooth with minimal disruption despite network instability.

Main Success Path (Primary Flow)

Actor Actions System Response

User experiences Server detects high latency and applies compensation mechanisms network lag. (e.g., input delay buffering, rollback adjustments).

Opponent Server ensures the game state remains synchronized for both players. continues playing.

Alternate Path A1

Actor Actions System Response

Latency is temporary and stabilizes. Server automatically resynchronizes the game state.

Exception Path E1

Exception System Response

Exception

Player’s connection drops long-term.

System Response

Player is disconnected, and match continues or ends based on game rules.

Use case: Click Square
Iteration: 1
Primary Actor: The Players
Goal in context: To receive input from player **Preconditions:** The chess game has started
Trigger: Player clicks square on screen with mouse **Scenario**:

1. A player wants to select one of their pieces
2. A player wants to deselect their currently selected piece
3. A player wants to move their currently selected piece

Post conditions: The GUI checks who the current player is and decides whether to accept the click based on whether it was made on the active player's screen or not

Exceptions:
1. If it isn't the turn of the player that clicked the square, the click will be ignored.

Priority: High. It is necessary for playing the game. **When available**: 1st iteration

Frequency of use: High (ie. every time a player decides to select, deselect, or move a piece in a chess game)

Channel to actor: Window on personal device screen that corresponds to the player's assigned colour (i.e. white or black)

Secondary actors: n/a
Channel to secondary actors: n/a

Open issues: Should the inactive player's screen be designed to not listen to clicks or should the clicks be heard but lead to no action?

Use case: Process Click

Iteration: 1

Primary Actor: The active player

Goal in context: To perform the action the active player instructed the game to do when they clicked a square.

Preconditions: The active player clicked a square **Trigger**: The active player clicked a square. **Scenario**:

1. The active player wants to select one of their pieces
2. The active player wants to deselect their currently selected piece
3. The active player wants to move their currently selected piece

Post conditions: Based on whether a square was already selected and whether the click was performed on a valid move destination, the necessary game state and screen presentation updates will be made.

Exceptions: Hardware issues.

Priority: High. Necessary for playing the game.

When available: 1st iteration.

Frequency of use: High (i.e. every time the active player decides to select, deselect, or move a piece in a chess game)

Channel to actor: Window on personal device screen that corresponds to the player's assigned colour (i.e. white or black)

Secondary actors: n/a

Channel to secondary actors: n/a

Open issues: Should the GUI call different methods from the Game Logic layer based on whether the click was performed on a valid move option and then update the screen in a targeted manner that is enabled by that knowledge or should there be a generic processClick() function that is executed followed by a generic process to update the screen?

Use case: Select Square

Iteration: 1

Primary Actor: Active player

Goal in context: To select a square that contains one of the active player's pieces and present the associated move options, as well define the piece to be moved as a set up for making a move.

Preconditions: It was determined that the square that the active player clicked on contains one of their pieces.

Trigger: The active player clicks on a square that contains one of their pieces **Scenario**:

1. The active player wants to select one of their pieces in order to move it.
2. The active player wants to select one of their pieces in order be presented with the possible

   moves that can be made in order to be aided in deciding what to do.

Post conditions: The square that was clicked on will be selected and the corresponding move options will be stored in the game logic as well as be visible on the screen. If the active player clicks on a square that represents a valid move option the currently selected piece will be moved.

Exceptions: Hardware issues.

Priority: High. Necessary for playing the game.

When available: 1st iteration.

Frequency of use: High (i.e. every time the active player in a chess game clicks on a square that contains one of their pieces)

Channel to actor: Window on personal device screen that corresponds to the player's assigned colour (i.e. white or black)

Secondary actors: n/a
Channel to secondary actors: n/a

Open issues: Since the GUI will present the move options, it could theoretically determine whether a clicked square is a valid move option based on whether it is marked as a move option, so should the GUI take advantage of this possibility or instead rely on the Game Logic to determine if a clicked square is a valid move option?

Use case: Deselect Previously Selected Square Iteration: 1
Primary Actor: The active player

Goal in context: To deselect the currently selected square so as to remove the distraction of being presented with the possible moves corresponding to a particular piece on the board and/or avoid the risk of accidentally making a move by clicking on a valid move option.

Preconditions: It was determined that the square that the active player clicked on is neither a valid move option nor a square that contains one of the player's pieces.

Trigger: The active player clicks on a square that both isn't a valid move option and doesn't contain one of their pieces.

Scenario:

1. The active player wants deselect the currently selected square without selecting another square in order to remove the distraction of seeing the move options for a specific piece.

2. The active player wants to not have any piece selected in order to avoid accidentally making a move by clicking on a valid move option.

Post conditions: There will no longer be a selected piece and no move options. The only action the player can take to continue the game is to select a piece.

Exceptions: Hardware issues.

Priority: Low. Not necessary for playing the game, but slightly improves the experience of the players.

When available: 1st iteration.

Frequency of use: Medium to High (i.e. every time the active player decides to remove all selections from their screen during a chess game)

Channel to actor: Window on personal device screen that corresponds to the player's assigned colour (i.e. white or black)

Secondary actors: n/a
Channel to secondary actors: n/a

Open issues: Should it instead be the case that when a player clicks on a square that is neither a valid move options nor a square that contains one of their pieces, that this action causes the square to be selected instead of just deselecting the currently selected square.

Use case: Make Move

Iteration: 1

Primary Actor: The active player

Goal in context: To move the currently selected piece to the square that is clicked on

Preconditions: A piece that is capable of moving is selected on the active player's screen and it is determined that the square that the active player clicked on is a valid move option.

Trigger: The active player clicks on a square that constitutes a valid move option for the currently selected piece.

Scenario:
1. The active player has decided what move they want to make and wants to instruct the game to

make that move.

Post conditions: The move will have been made and the new game state will be stored in the Game Logic and presented on both of the player's screens. The other player will be the active player.

Exceptions: Hardware issues.

Priority: High. Necessary for playing the game.

When available: 1st iteration.

Frequency of use: High (i.e. every time a player makes a move in a chess game)

Channel to actor: Window on personal device screen that corresponds to the player's assigned colour (i.e. white or black)

Secondary actors: n/a
Channel to secondary actors: n/a

Open issues: Since the GUI will present the move options, it could theoretically determine whether a clicked square is a valid move option based on whether it is marked as a move option, so should the GUI take advantage of this possibility or instead rely on the Game Logic to determine if a clicked square is a valid move option?

Use case: Update Active Player's Screen

Iteration: 1

Primary Actor: The active player

Goal in context: To update the active player's screen to present the new state of selection and corresponding move options.

Preconditions: The active player clicks on a square that either results in a new square selection to be made or an absence of a square selection to be established.

Trigger: The active player clicks on a square that either results in a new square selection to be made or an absence of a square selection to be established.

Scenario:
1. The active player has changed or undone the current square selection and so the new state of

selection and move options must be presented to the active player.

Post conditions: The active player will be able to see the new state of selection and move options.

Exceptions: Hardware Issues

Priority: High. Very important to show the active player which of their pieces is currently selected

When available: 1st iteration

Frequency of use: High (i.e. every time a player changes the state of their selection during a chess game)

Channel to actor: Window on personal device screen that corresponds to the player's assigned colour (i.e. white or black)

Secondary actors: n/a
Channel to secondary actors: n/a

Open issues: What specific method will we implement to mark squares as valid move options on the screen?

Use case: Update Both Player Screens

Iteration: 1

Primary Actor: the active player

Goal in context: To update both player's screens to show the new state of the game after a move is made.

Preconditions: The active player has made a move and the screens have not yet been updated to the new state of the game.

Trigger: The active player makes a move. **Scenario**:

1. A player has made a move and so both player's need to be presented with the new state of the game.

Post conditions: The new active player is able to see the current state of the game and make selections and perform the next move.

Exceptions: Hardware issues.
Priority: High. It is necessary for the player's to know the state of the game.

When available: 1st iteration
Frequency of use: High (i.e. every time a move is made by a player during a chess game)

Channel to actor: Window on personal device screen that corresponds to the player's assigned colour (i.e. white or black)

Secondary actors: n/a
Channel to secondary actors: n/a

Open issues: Should we follow through with the current plan of having two separate windows for each player?

Use case: Switch Active Player

Iteration: 1

Primary Actor: The active player

Goal in context: To switch which player is considered the active player after a move is made.

Preconditions: A player has made a move and the resulting state of that player's captured pieces has been recorded in the Game Logic

Trigger: A move is made by the active player. **Scenario**:

1. A player makes a move which results in it becoming the other player's turn

Post conditions: The player that was previously considered the inactive player will now be considered the active player and be able to select pieces on their screen and make a move,

while the player that was previously considered the active player will be considered the inactive player, and will no longer be able to interact with the chess board on their screen until the other player makes a move.

Exceptions: Hardware issues.

Priority: High. Necessary for playing the game.

When available: 1st iteration.

Frequency of use: High. (i.e. every time a player makes a move in a chess game)

Channel to actor: Window on personal device screen that corresponds to the player's assigned colour (i.e. white or black)

Secondary actors: n/a
Channel to secondary actors: n/a

Open issues: What is the best way of implementing a turn based game in a GUI and, as asked above, will we follow through with the current plan of having two separate windows for each player?

Use Case: Playing Multiplier Tic Tac Toe Primary Actor:

- Tic Tac Toe player one and player two

Goal:

- The goal of each player is to win the game by placing their selected symbol side by side three times either in a row, column or diagonally.

Precondition
- Each player must log into their accounts for the online board game platform.
- To play the game in multiple players must have access to wi-fi.
- In order to run the game, the player must select Tic Tac Toe in the board games menu.

Trigger:

- Once the game has begun running the player selects start and joins an online session with another online player.

Scenario:

1.Player one selects X or O, the symbol that remains unselected gets assigned to player two.

2.Player one selects any empty cell inside of the three-by-three grid. 3.The game places player ones symbol inside of the selected cell. 4.The display changes to show that its

player two's turn
5.Player two selects an empty cell

6.The game places player two's symbol inside of the selected cell.
7.Steps 2 – 6 repeats until either a player wins the game or until all the cells get filled.

8.If a player wins the display changes to show the winner and each player can choose to exit the game, rematch or choice to play another match with someone else.

Exceptions:

1.All cells become occupied: The game ends in a draw and each player can choose to rematch or exit the game.

2.Player selects a cell when it's not their turn: The grid does not change but the player is alerted through a pop up that it's not their turn

3.A player exits their game before the game ends: The remaining player is connected to a new match

4.A player has not interacted with the game for more than two minutes: The inactive player is kicked out of the match and the active player wins the game.

5.Player has a weak or non-existent internet connection: A pop-up window alerts the player of the issue, after which the player is not allowed to join the online game until they have a strong enough connection.

Priority:
- Essential: required to the test basic functionality of the platform.

When available:
- End of second project iteration.

Frequency of use: - Frequent.

Channel to actor:
- Trackpad, mouse and keyboard used by player.

Secondary Actor:
- Dev team responsible for creating and maintaining the online board game platform.

Channel to secondary actor: - Technical support: Email

Open issues:
- Should we implement an offline version of the game.
- For the offline version are we going to incorporate different difficulty levels. - Should we provide the option for players to play on different grid sizes.

1. Start New Game Primary Actor: Game Player

Goal: Initiate a new Connect 4 game session within the application.

Pre-Conditions:

• The game application is running locally. • No active game session is in progress.
• Networking is available.
• The player is logged into the game client.

Trigger:

• The player selects the "New Game" option from the game menu.

Scenario:

1. The player launches the game client.
2. The player selects "New Game" from the main menu.
3. The system initializes an empty custom-size board.
4. Thesystemdetermineswhichplayergoesfirst(eitherrandomlyorbyapredefined rule).

5. The initialized board and the turn indicator are displayed on the local interface.

Post-Conditions:

• A new game session is started with a fresh board.
• The active player is prompted to make the first move.

Exceptions:

• System or application error during game initialization. • Network issues in multiplayer mode.

Priority: High (Essential for starting gameplay)
When Available: At the beginning of every game session.
Frequency of Use: Every time a player wants to start a game.
Channel to Actor: Local game interface
Secondary Actors: System
Channels to Secondary Actors: Internal application processes
Open Issues: Handling unexpected shutdowns or reconnections during game startup.

2. Making a Move
Primary Actor: Game Player

Goal: Allow the active player to drop a disc into a chosen column.

Pre-Conditions:

• A game session is active with an initialized board.
• It is the active player's turn.
• The current board state is displayed on the game client.

Trigger:

• The player selects a column within the allowed range using the local input device (e.g., keyboard).

Scenario:

1. The active player selects a column via the game client interface.
2. The system verifies that the chosen column number is within the valid range.
3. The system checks that the selected column is not already full.
4. The disc is placed in the lowest available row of the selected column.
5. The board is updated immediately on the local interface to reflect the new move.

Post-Conditions:

• The game board displays the new move.
• The turn passes to the opposing player unless a win/draw condition is met.

Exceptions:

• Column selection is out-of-range.
• The chosen column is already full.
• An error occurs during the board update process.

Priority: High (Core gameplay functionality)
When Available: Each turn during an active game session.
Frequency of Use: Multiple moves per game session.
Channel to Actor: Local game UI (input devices such as keyboard or mouse)
Secondary Actors: System
Channels to Secondary Actors: Internal processing and UI rendering
Open Issues: Resolving simultaneous move attempts in multiplayer mode and providing visual feedback (animations, sounds) for move validation.

3. Checking for Win/Draw

Primary Actor: System
Goal: Automatically determine if the latest move results in a win or if the game concludes in a draw.

Pre-Conditions:

• A move has just been made and the board state has been updated. • The game session is active.

Trigger:

• Completion of a player's move.

Scenario:

1. The game engine scans the board after each move.
2. It checks for four consecutive discs in a row horizontally, vertically, or diagonally.
3. Ifawinisdetected:

o The system declares the active player as the winner.

o The game session is ended.
4. If no win is detected and the board is full:

o The system declares a draw.

o The game session is ended. 5. If neither condition is met:

o The turn is passed to the opposing player. Post-Conditions:

• The game session either continues with the next turn or concludes with a win/draw.

Exceptions:

• System error during evaluation.
• Inconsistencies in the board state.

Priority: High (Critical for game resolution)
When Available: After every move.
Frequency of Use: Every turn during the game.
Channel to Actor: Internal processing with results displayed on the game client
Secondary Actors: System

Channels to Secondary Actors: Backend evaluation
Open Issues: Optimizing win/draw detection, especially for customizable board sizes.

4. Handling Invalid Moves Primary Actor: Game Player

Goal: Prevent and handle invalid move attempts gracefully during gameplay.

Pre-Conditions:

• A game session is active. • It is the player's turn.

Trigger:

• The player attempts to drop a disc into an invalid column (either out-of-range or full).

Scenario:

1. The player selects an invalid column.
2. The system detects that the column is either out-of-range or full.
3. An error message is immediately displayed on the local interface, indicating the nature of the invalid move.
4. The system maintains the turn for the same player until a valid move is made.

Post-Conditions:

• The board remains unchanged.
• The player is prompted to select a valid move.

Exceptions:

• Multiple invalid attempts might trigger additional warnings or a temporary move lock-out (if configured).

Priority: High (Essential for maintaining game integrity)
When Available: Every time an invalid move is attempted.
Frequency of Use: As frequently as players may make input errors.
Channel to Actor: Local game UI using available input devices
Secondary Actors: System
Channels to Secondary Actors: Internal input validation and feedback mechanisms
Open Issues: Defining thresholds for temporary lock-outs after repeated invalid moves.

5. Ending the Game

Primary Actor: System
Goal: Conclude the game session when a win, draw, or game end by player condition is met.

Pre-Conditions:

• A winning move, draw, or game end by player condition has been detected. • A game session is active.

Trigger:

• A move results in a win or draw, or the player issues a game end by player command.

Scenario:

1. The game engine detects a win, a draw, or receives a game end by player signal.
2. The system terminates the active game session.
3. The final board state is displayed on the game client.
4. The system announces the game result (win, draw, or game end by player) on the interface.
5. Optionally, the player is offered the option to start a new game.

Post-Conditions:

• The game session is concluded.
• The result is clearly communicated to the player.
• The game client returns to the main menu or game summary screen.

Exceptions:

• System error during game termination.

Priority: High (Necessary for clear and proper game closing)
When Available: At the end of every game session.
Frequency of Use: Once per game session.
Channel to Actor: Local game client displaying alerts and final game summary
Secondary Actors: System

Channels to Secondary Actors: Internal processes and UI notifications
Open Issues: Offering a detailed summary screen with game statistics and a rematch option.

6. Writing Chat Message

Primary Actor: Game Player
Goal: Allow the active player to send a chat message to their opponent or other players during a multiplayer game session.

Pre-Conditions:

• A game session is active.
• Players are in a multiplayer environment (local or online).
• The chat feature is available and enabled in the game client.

Trigger:

• The player selects the "Chat" option from the game interface and types a message.

Scenario:

1. The player accesses the chat window from the game interface.
2. The player types a message and hits the "Send" button.
3. The system validates the message (e.g., checking for inappropriate content or enforcing length restrictions).
4. The system displays the message on the screen for the opposing player(s) to view in real time.
5. Optionally,thesystemplaysanotificationsoundtoinformtheopponent(s)ofthe new message.

Post-Conditions:

• The message is successfully sent and displayed to the other players. • The game continues without interruption.

Exceptions:

• The message contains invalid characters or inappropriate language. • The message fails to send due to network issues.

Priority: Medium (Enhances player interaction but is not critical for gameplay) When Available: Throughout the game, during both player turns and intermissions. Frequency of Use: Occasional, based on player communication needs.
Channel to Actor: Game client interface (via keyboard or chat UI)
Secondary Actors: System
Channels to Secondary Actors: Internal message validation and display

Open Issues: Defining chat message filters for inappropriate content and ensuring smooth performance in multiplayer scenarios.

Use Case Description: BoardGame Go

1. Title

Launch and Play a Game of Go (JavaFX Implementation)

2. Primary Actors

- **Player/User:** Interacts with the game by clicking on the board.
- **System/Application:** The JavaFX program managing game initialization, rendering, and move processing.

3. Preconditions

- **Application Launch:** The JavaFX application is successfully started.
- **Board Initialization:** A 19×19 board is created (represented by a two-dimensional array) and displayed on a canvas.
- **Initial Settings:** The board is rendered with grid lines, star points, and no stones. The turn is set to Black (indicating Black moves first).

4. Main Success Scenario (Basic Flow)

1. **Game Initialization:**
    - **The application window opens with the title "围棋游戏".**
    - **A Canvas is created with dimensions based on the board size and tile size.**
    - **The board array is initialized with all positions set to 0 (empty).**
    - **The drawBoard() method is called to render the grid, star points, and any existing**

      stones.
    - **Mouse click event handling is enabled on the Canvas.**
2.
3. **Player Move Input:**
    - **User Action: The player clicks on a desired intersection on the board.**
    - **System Response: The application calculates the board coordinates by adjusting the**
4.

mouse click position relative to the tile size.

3. Move Validation and Stone Placement:

a.

b.

c.

Validation: The system checks if the clicked coordinates are within bounds and that the corresponding cell in the board array is empty.
Valid Move:

i. The board array is updated with a value (1 for Black, 2 for White) based on whose turn it is.

ii. The drawBoard() method is called again to refresh the display with the new stone.

iii. The turn toggles to the opposing player.

Invalid Move:

i. If the selected position is out-of-bounds or already occupied, an Alert is triggered informing the player with the message "Please select a correct location."

4. Game Continuation:

a. The process repeats as players take turns placing stones until the application is closed or additional game logic (such as game termination or scoring) is introduced.

5. Alternative Flows

• **Invalid Move Attempt:**
 o Scenario: The player clicks on a cell that is either occupied or outside the valid board

area.
 o Outcome: The system displays a warning Alert, and the move is rejected. The turn does

not change.

• **No Further Move Handling:**
 o Scenario: The game currently lacks features such as capturing stones or scoring.
 o Outcome: The game continues solely with stone placement and turn alternation until

the application is terminated.

6. Postconditions

- **Board State Update:** The board (both the visual display and the internal array) reflects the stones placed during the session.

- **Turn Management:** The application maintains which player's turn is next.
- **User Feedback:** Players are informed of invalid moves via alerts, ensuring proper move validation.

7. Extensions and Future Considerations

- **Game Rule Enhancements:** Future versions may include capture detection, scoring logic, and end-game conditions.
- **User Interface Improvements:** Additional UI elements such as a status bar, move history, or reset options could enhance the user experience.
- **Error Handling:** More robust error handling could be introduced to manage unexpected input or runtime exceptions.

Use Case 1: Challenge Players **Iteration:** 1

Primary Actor: Registered Player
Goal in context: Allow a registered player to challenge another player to a

match in an available board game

Preconditions:

1. The user is logged into their account
2. The user has an active internet connection
3. The opponent they wish to challenge must also be a registered user and online
4. The game server must be running
5. At least one game must be available for selection

Trigger: The player selects the option to challenge another player from the GUI

Scenario:

1. The player navigates to the "Challenge Players" section in the GUI
2. The system displays a list of available online players
3. The player selects an opponent from the list
4. The system prompts the player to select a game from the available options
5. The player selects a game and confirms the challenge
6. The system sends a challenge request to the selected opponent
7. The opponent receives a notification of the challenge
8. The opponent either accepts or declines the challenge
9. If accepted, the system initializes the game and directs both players to the game lobby
10. If declined, the system notifies the challenger of the declined request

Postconditions:

1. If the challenge is accepted, both players enter the game lobby, ready to start the match

2. If the challenge is declined, the challenger is returned to the "Challenge Players" screen

Exceptions:

1. The opponent goes offline before responding: the system must notify the challenger and cancel the challenge, sending the user back to the game lobby
2. The game server is unavailable or down: the system must inform the user and prevent challenges from being sent or game selection
3. The opponent is already in another match: the system must notify the challenger and direct them back to the screen of online players
4. The challenger cancels the request before the opponent responds: the system must withdraw the challenge and cancel the challenge notification to the opponent

Priority: High, this feature is essential for multiplayer games and direct player interaction and matchmaking

When available: This feature is always accessible to players in the GUI once they are logged in and connected to the game server

Frequency of use: Every time a player logs in to the server and wants to play a game with another user online

Channel to actor: The player interacts through the GUI by selecting from the options to challenge another player who is online

Secondary actors:

1. Opponent Player (receives the challenge and chooses whether to accept) 2. Game Server (handles matchmaking and game initialization)

Channel to secondary actors:

1. Opponent Player: receives an in-game notification
2. Game Server: Processes and maintains server status, game availability and matchmaking status for players

Open issues:

1. Multiple incoming challenges to the same player
2. Ability to block another user to avoid receiving challenge requests from them
3. Timeout mechanism for responding to challenges to prevent inactive or AFK players from slowing matchmaking times
4. Having a friends list of specific players you frequently play with

Use Case 2: Look up game

Iteration: 1.0, last modification: March 5th by Kemuel

Primary Actor: Registered player
Goal in context: Allows for a player to look through their list of games and read a description of its gameplay.

Preconditions:

1. Player is logged in.

Trigger:

1. When the customer starts the application.
2. When you return to your games list after playing a game.

Priority: High priority needed to interact with the rest of the program.
When is it available: Available anytime the application is open, and when a

game has ended.

Scenario:

1. Player starts game suite application
2. Games on suite popup in a list
3. Player clicks on a game in the list and sees their descriptions

Exceptions:

1. Player Does Not Start application
2. Player is playing another game simultaneously 3. Player is not logged in

When available: Available as soon as the application starts.
Frequency of use: High Frequency. Every time a user wants to look through

their games

Channel to actor: Player uses Keyboard/ mouse interactions to interact with the application

Secondary actors:

1. Game application client.
2. Game servers.

Open Issues:

1. Searching up games by text
2. Different ways to sort games. Example: Aplhabedicly, last played date, most played, self sort.

Use Case 3: Look up Players **Iteration:** 1

Primary actor: Registered Player
Goal in context: To allow users to search for players using the GUI search feature, to access their profiles, and to examine stats before challenging them.
Preconditions:

1. The user is logged in.
2. The user has an internet connection.
3. The dashboard window is fully loaded, displaying a button called “Look up Player”

Trigger: The user selects the user “Look up Players” button with their mouse.
Scenario:

1. The user enters a player's name into the search bar and the list updates dynamically with similar names.
2. The user applies filters (e.g., level, games played) via dropdown menus.
3. The user sorts the list by name, level, or other attributes using sorting buttons.
4. The user enters a player's name into the search bar.

5. The user scrolls through the list using their mouse or the scroll bar.

Postconditions:

1. The selected player’s profile is displayed in a new window.
2. Hovering over a name in the list triggers a pop-up showing the player's name and stats.

Exceptions:

1. If no matching players are found, the GUI displays a "No players found" message.
2. If the user loses connection while searching, the GUI displays a "Connection lost. Couldn’t load list." message.

Priority: High. A multiplayer GUI has to provide a way to access player stats of all players, for users to find and interact with others.

When available: Third iteration.
Frequency of use: High. Players will frequently use this feature to find

opponents.
Channel to actor: The GUI **Secondary actors**:

1. The player database. 2. The backend server.

Channels to secondary actors: 1. Internet

Open issues: None

Use Case 4: Join a game from the library **Iteration**: 1.0

Primary actor: registered player

Goal in context: After looking up for games read descriptions of gameplay or rules, allowing a registered player to participate in the game by clicking and choosing the gameplay.

Pre-conditions:

1. The player is successfully logged into their account.
2. The player is directed to the game library page.
3. The game library is not empty and is available for selection

Trigger: The registered players join a game by choosing and clicking on the game icon or “start” button from the game library

Priority: High-needed, the players should join in the game first, either want to play solo or join a team in multi-player games later.

When available: Available any time if the players have good internet connection and have logged into the game as a registered player to see the game library and pick one from it.

Scenario:

1. Players log into the game application
2. Players navigate to the game library
3. Players read the descriptions and rules of the game
4. Players engage in a game by clicking game icon to start game session

Exceptions:

1. Internet corruption

i. The Wi-fi is so unstable that the game cannot continue

ii. The system prompts a message on refreshing the Wi-fi or reloading the game.

2. Requirement for update

i. The game has updated new features

ii. The system prompts a message on updating the game that players chose.

3. Unavailable participation due to full session

i. The game is full as the number of players for that game reaches the capacity

ii. The system prompts a message for matchmaking the player with the AI bot; or suggests estimated waiting time until the new session is available.

Frequency of use: High frequency, the players engage in the game by first logging in and joining in a game

Channel to actor: the GUI **Secondary actors:** the game server

Channel to the secondary actors: Game server: maintain the server to run 24/7

Open issues: N/A

Use case 5: In-Game Chatting **Iteration**: 1

Primary Actor: Player (active in game session)

Goal in context: Enable real-time texting between players during gameplay without interrupting the gaming experience, allowing for social interaction, game strategy, and sportsmanship.

Preconditions:

1. User is logged in to the system
2. User is currently participating in an active game session
3. At least one opponent or teammate is connected to the same game session
4. User has not been muted or restricted from the chatting feature

Trigger:

1. User activates chat interface by clicking chat icon or using keyboard shortcut User receives message from another player
2. System event occurs requiring player notification (game state changes, turn alerts)

Scenario:

1. The player initiates the chat interface during an active game
2. System displays chat window in a non-intrusive position relative to the game board
3. System loads and displays the most recent message history (last 20 messages or configurable amount).
4. Player types the message in the text input box.
5. Player sends a message by pressing Enter key or clicking Send button
6. System validates messages for length and content restrictions.
7. System transmits message to messaging service with metadata (sender,

timestamp, game session)

8. System displays the message in the local chat history with appropriate formatting

9. Recipients receive and view the message in their chat window

10. System sends a subtle notification sound (if enabled in user preferences)

11. Chat window can be minimized or relocated by user if desired

12. User can scroll through message history while game continues

13. If chat is inactive for 30 seconds, it automatically minimizes (optional tweek)

Postconditions:

1. Messages are left in the chat history throughout the game session 2. All active participants can view the message
3. Game state and performance remain unaffected
4. Chat history is saved with game record for potential review

Exceptions:

1. Message exceeds length limit - System displays character count warning and prevents submission

2. Inappropriate content detection - System filters content according to platform policy, or blocks message and warns user

3. Rapid message flooding - System implements cooldown and notifies user to prevent spam

4. Connectivity issue - System queues message for retry and displays sending status indicator

5. Recipient has blocked sender - Message is sent but not delivered, with no notification to sender

6. User has enabled "focus mode" - Incoming messages are queued but notifications suppressed

7. Game is in critical state (time pressure, final moves) - System may delay non-urgent notifications

Priority: Medium-High
When available: Iteration 2
Frequency of use: Very frequent - multiple times per game session

Channel to actor:

1. Visual chat interface within game UI
2. Notification system (visual and/or audio) 3. Optional keyboard shortcuts

Secondary actors:

1. Other players in the game (Opponents/Opponent) 2. The system that handles messages
3. Basic content filtering

Channel to secondary actors:

1. Basic communication between game components 2. Real-time connection for instant messaging
3. Connection to the main game session

Open issues:

1. Should private messaging between specific groups (e.g., team members) be allowed in team-based game modes?

2. How should chat history be saved between game sessions for regular opponents?

3. Should chat have rich text formatting or emoji reactions?

4. How to balance chat visibility with space constraints in small screens

5. Should voice chat be considered a future enhancement?

6. How to manage international player translations?

7. What level of administrative control is needed for chat moderation?

8. Should chat support auto-send game event notifications?

9. How will the "chat" feature scale between different game types (e.g.,

   turn-based vs. real-time)?

Dashboard

Look up Player

Friends List

Settiings

Log Out

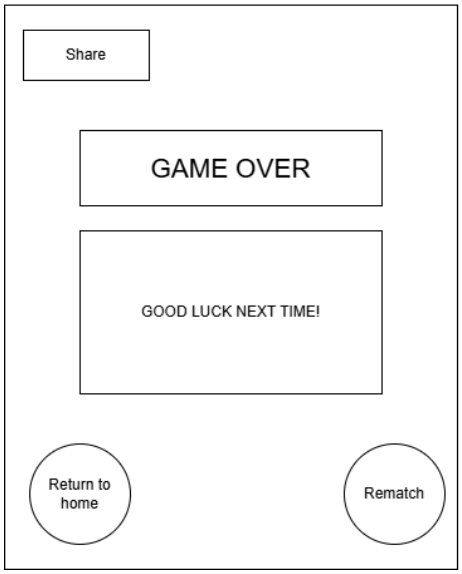
Info

Player Name

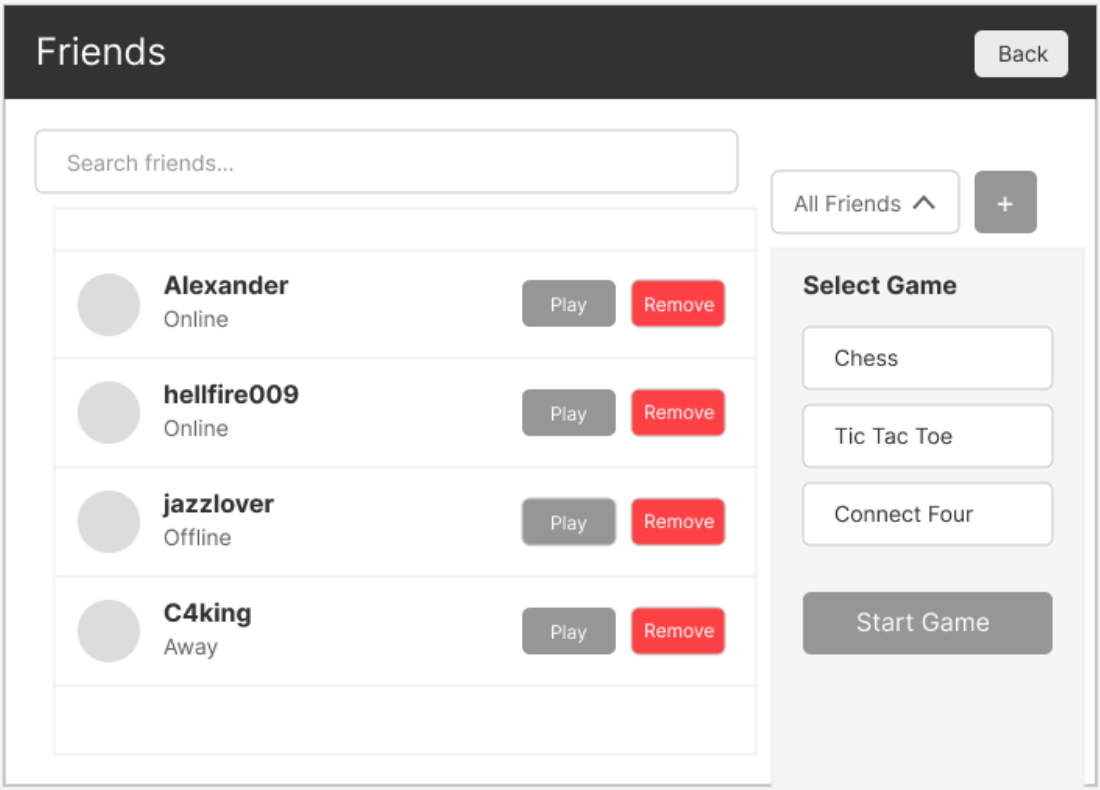
View Profile

Games

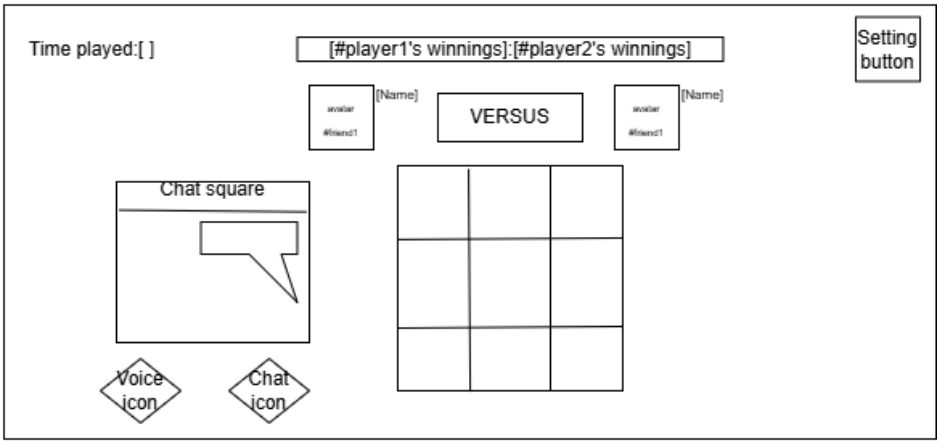
Top Players


Share
GAME OVER
GOOD LUCK NEXT TIME!
Return to home
Rematch

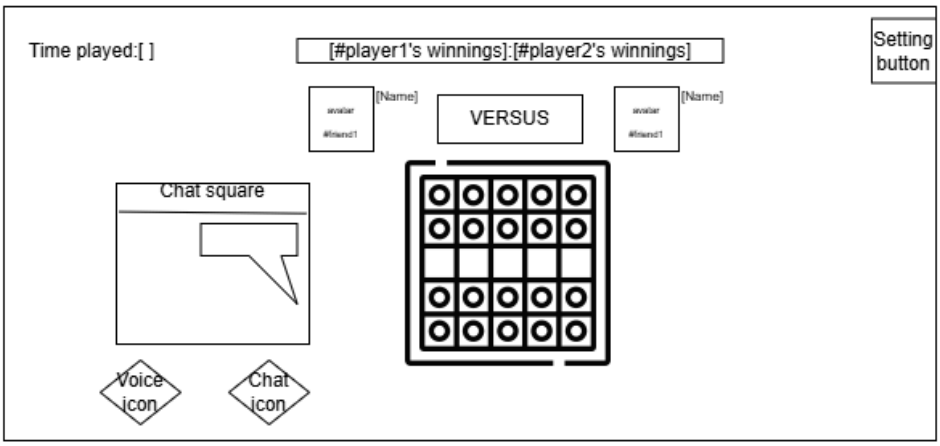

Share
GAME OVER
CONGRATULATIONS!
YOU HAVE WON THE GAME
Return to home
Rematch


Friends
Back
Search friends...
All Friends
+
Alexander
Online
Play
Remove
hellfire009
Online
Play
Remove
jazzlover
Offline
Play
Remove
C4king
Away
Play
Remove
Select Game
Chess
Tic Tac Toe
Connect Four
Start Game

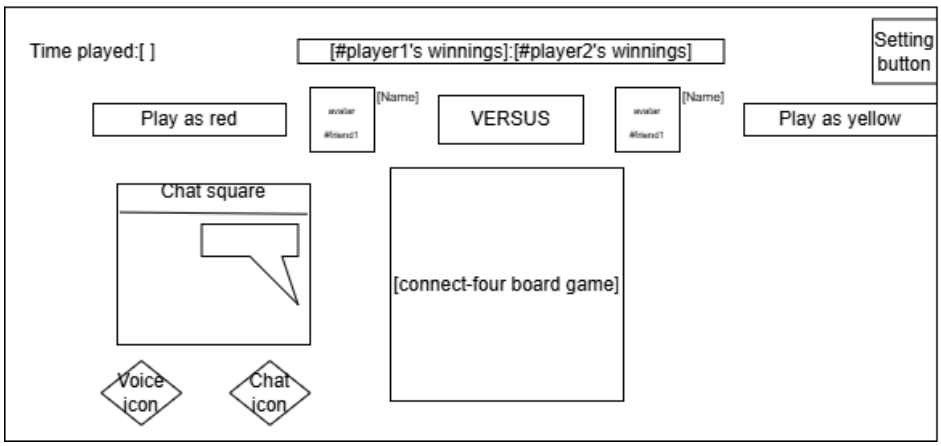
TIC TAC TOE



Chess



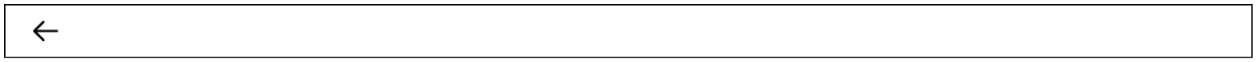
Connect four



ABOUT FAQ SUPPORT

ONLINE MULTIPLAYER PLATFORM

Login Sign Up



Login

Don't have an account yet ? Sign up

Username

Username

Password

******

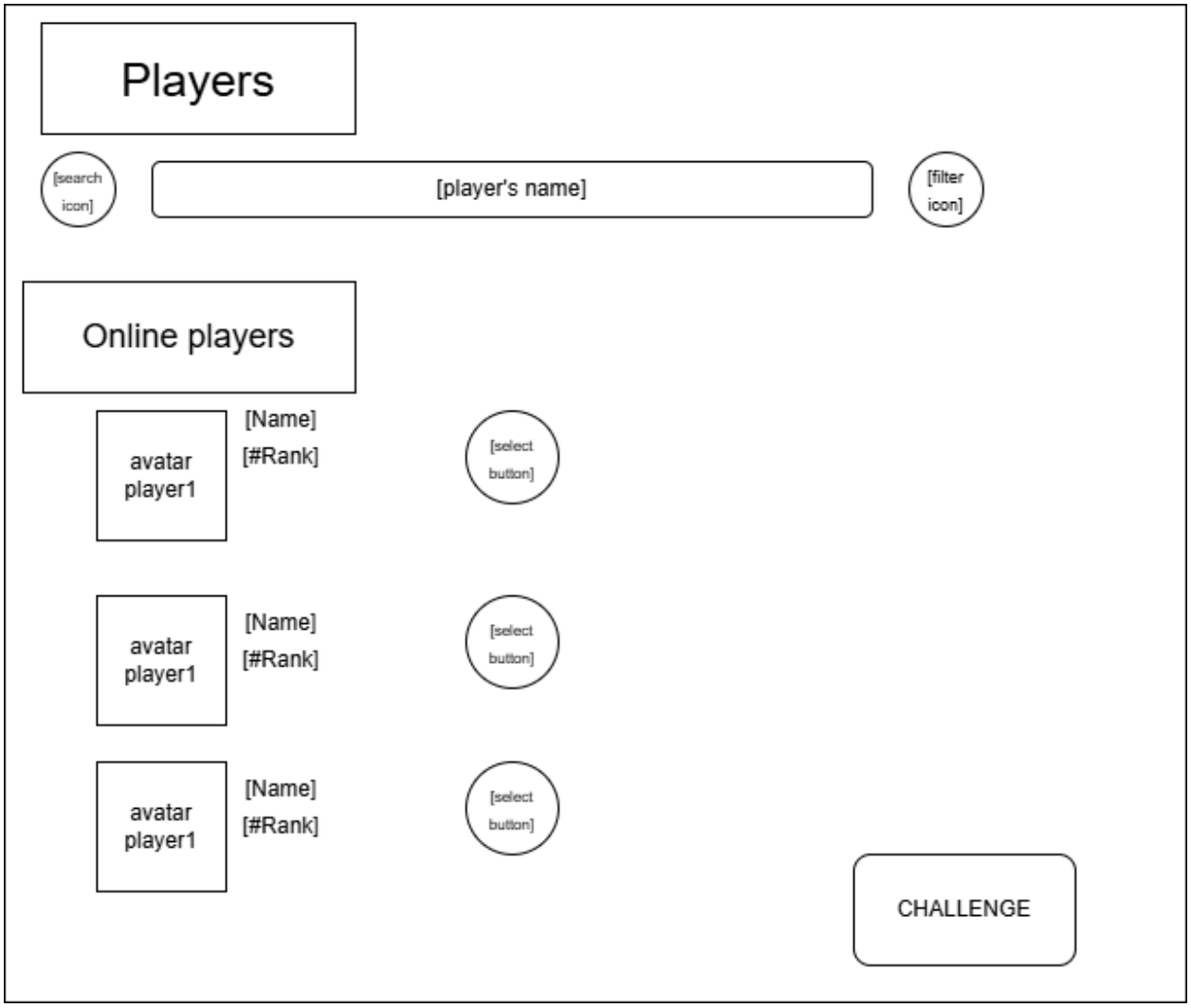
Login

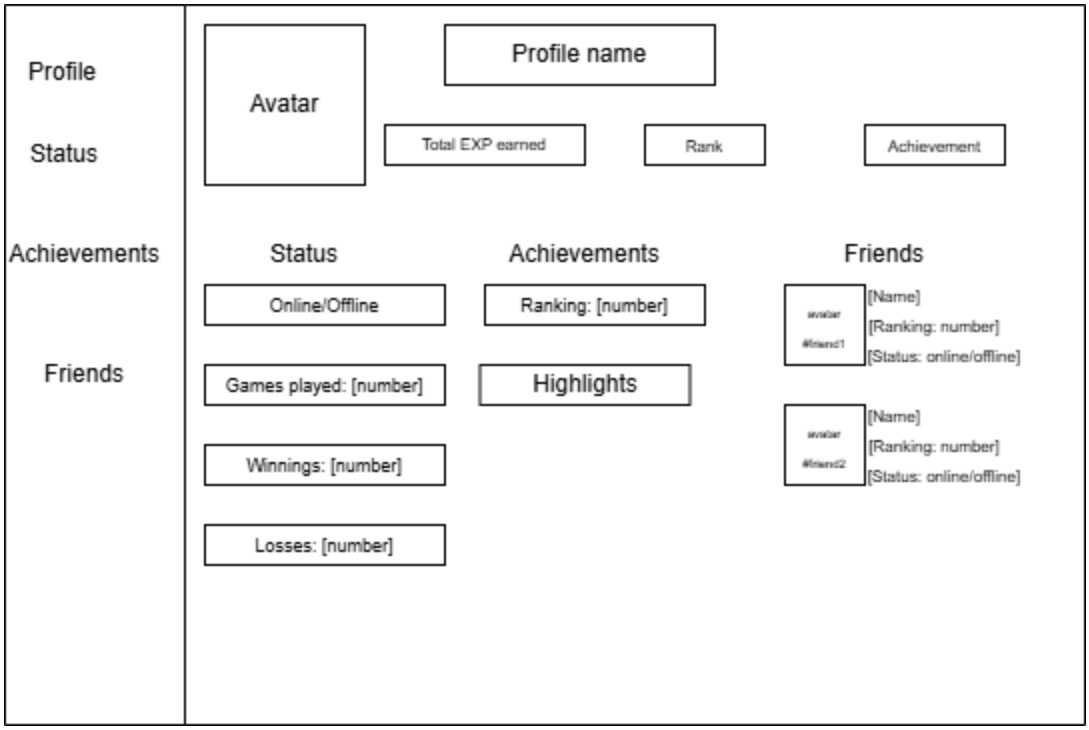
Guest Login

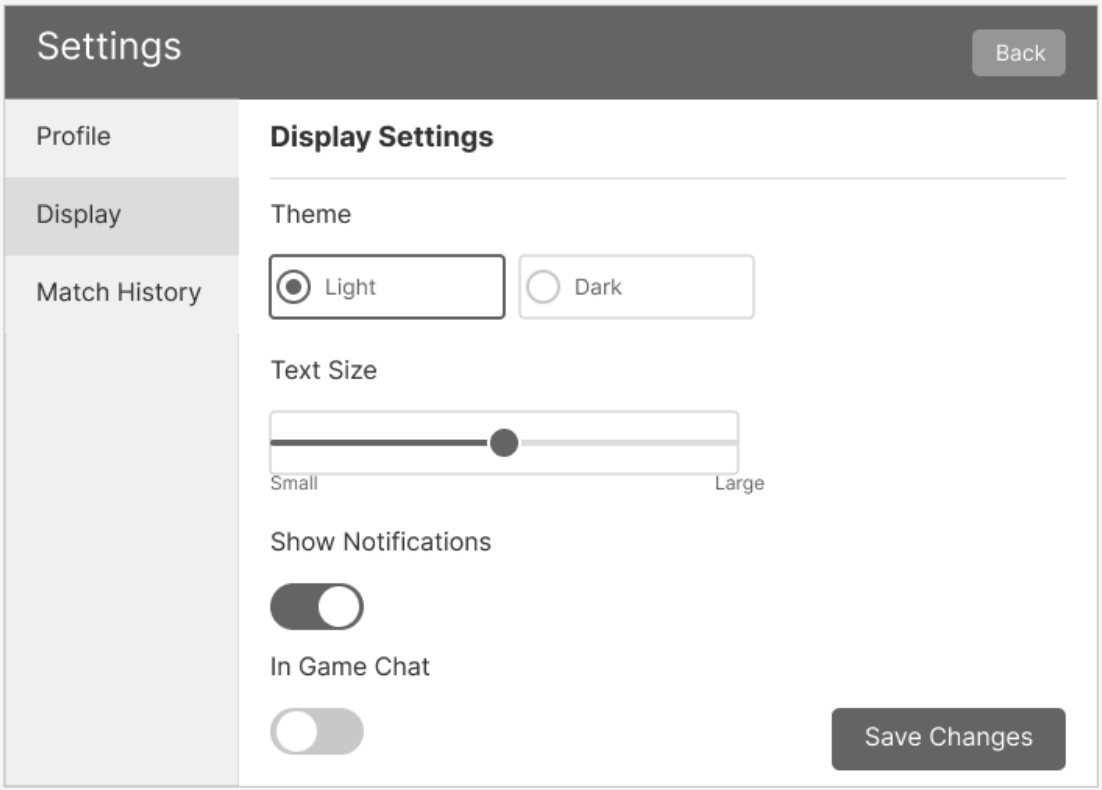
Username

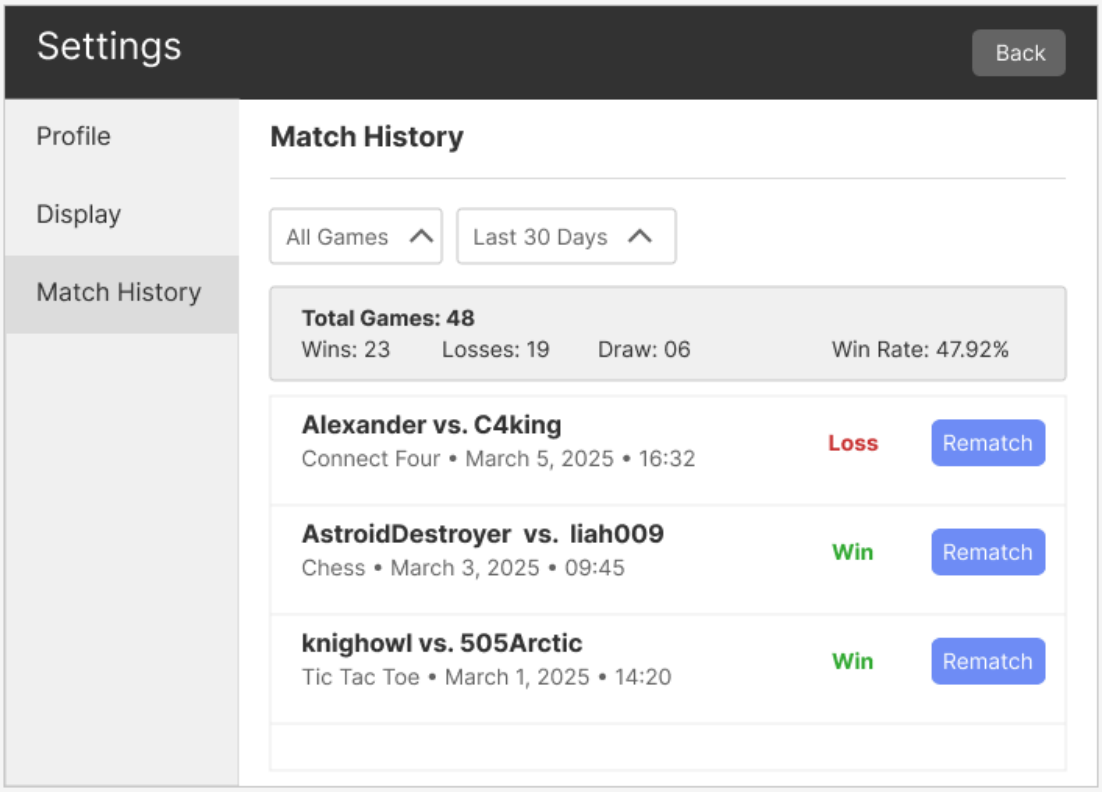
Username

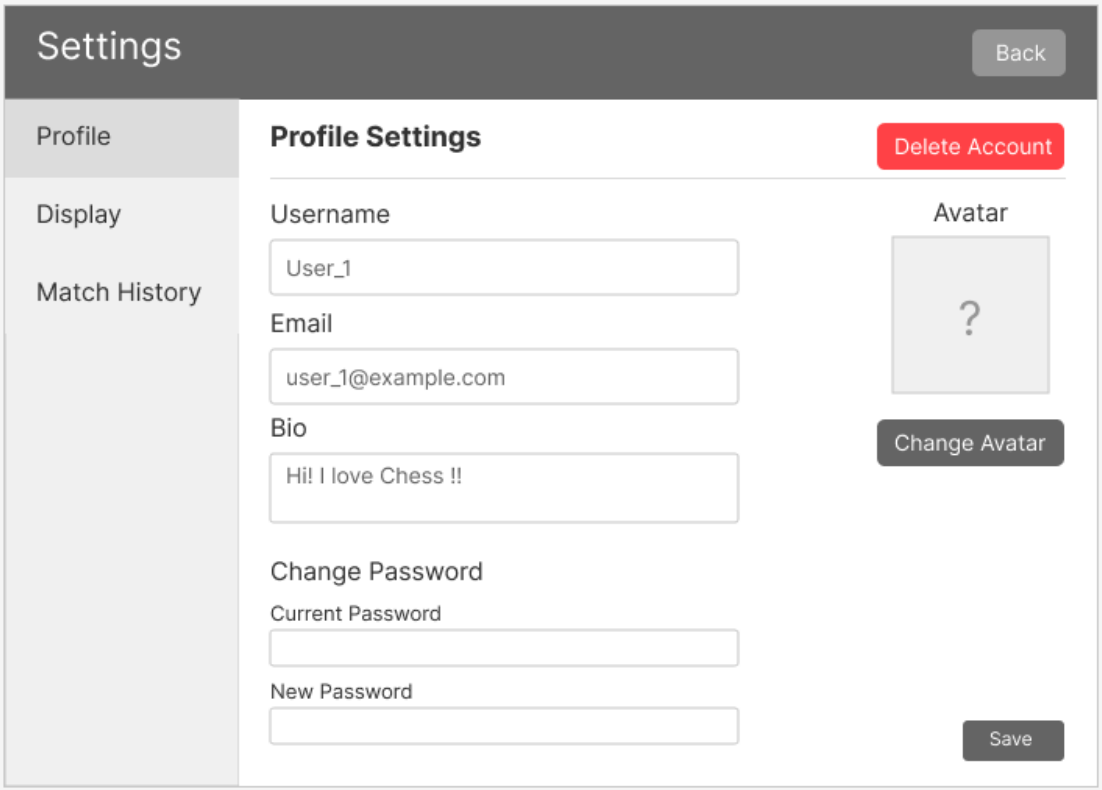
Login as Guest

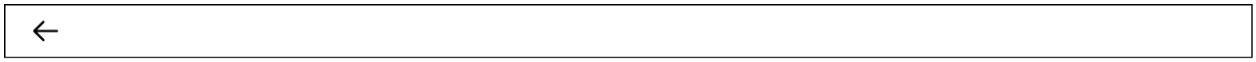

Players
[search icon]
[player's name]
[filter icon]
Online players
avatar player1
[Name]
[#Rank]
[select button]
avatar player1
[Name]
[#Rank]
[select button]
avatar player1
[Name]
[#Rank]
[select button]
CHALLENGE


Profile
Status
Achievements
Friends
Avatar
Profile name
Total EXP earned
Rank
Achievement
Status
Online/Offline
Games played: [number]
Winnings: [number]
Losses: [number]
Achievements
Ranking: [number]
Highlights
Friends
avatar
#friend1
[Name]
[Ranking: number]
[Status: online/offline]
avatar
#friend2
[Name]
[Ranking: number]
[Status: online/offline]


Settings
Back
Profile
Display
Match History
Display Settings
Theme
Light
Dark
Text Size
Small
Large
Show Notifications
In Game Chat
Save Changes


Settings
Back
Profile
Display
Match History
Match History
All Games
Last 30 Days
Total Games: 48
Wins: 23
Losses: 19
Draw: 06
Win Rate: 47.92%
Alexander vs. C4king
Connect Four • March 5, 2025 • 16:32
Loss
Rematch
AstroidDestroyer vs. liah009
Chess • March 3, 2025 • 09:45
Win
Rematch
knighowl vs. 505Arctic
Tic Tac Toe • March 1, 2025 • 14:20
Win
Rematch


Settings
Back
Profile
Display
Match History
Profile Settings
Delete Account
Username
User_1
Email
user_1@example.com
Bio
Hi! I love Chess !!
Avatar
?
Change Avatar
Change Password
Current Password
New Password
Save



Sign Up

Already have an account? Login

Username

Username

Password

******

Re-type Password

******

Sign Up

Dashboard

Look up Player

Friends List

Settiings

Log Out

Info

Tic Tac Toe

Join a Match

Find an Opponent

Leaderboard