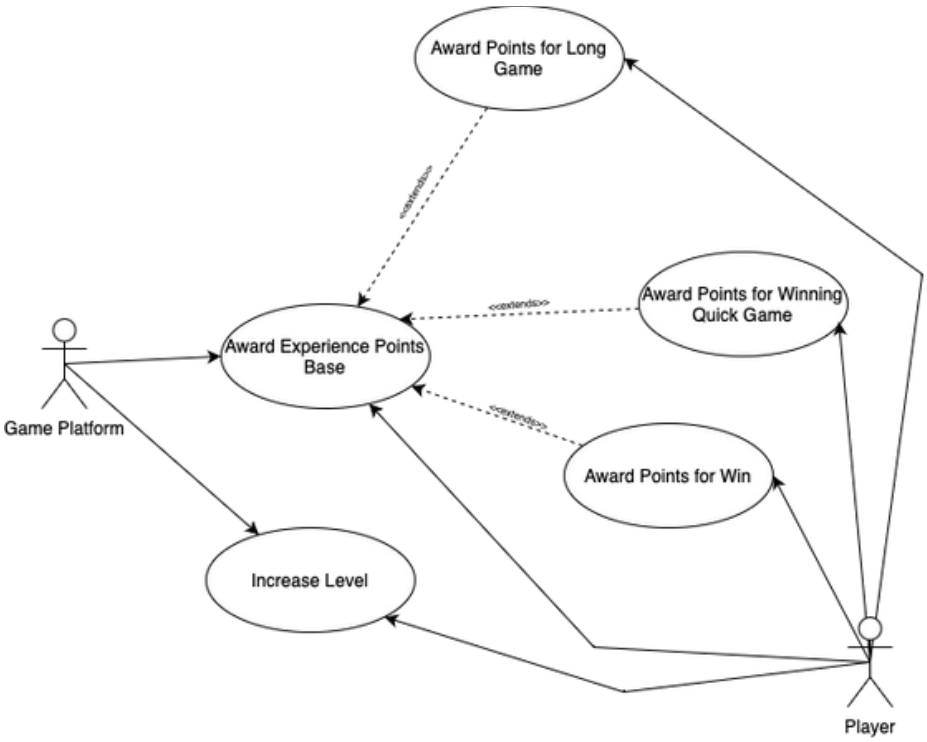
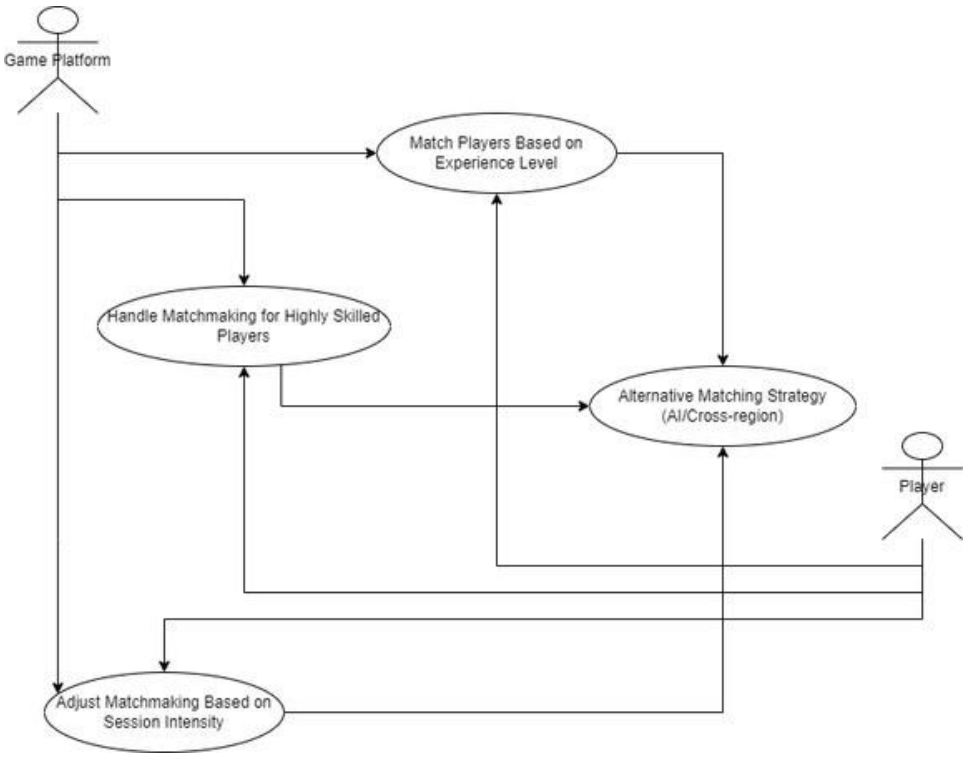
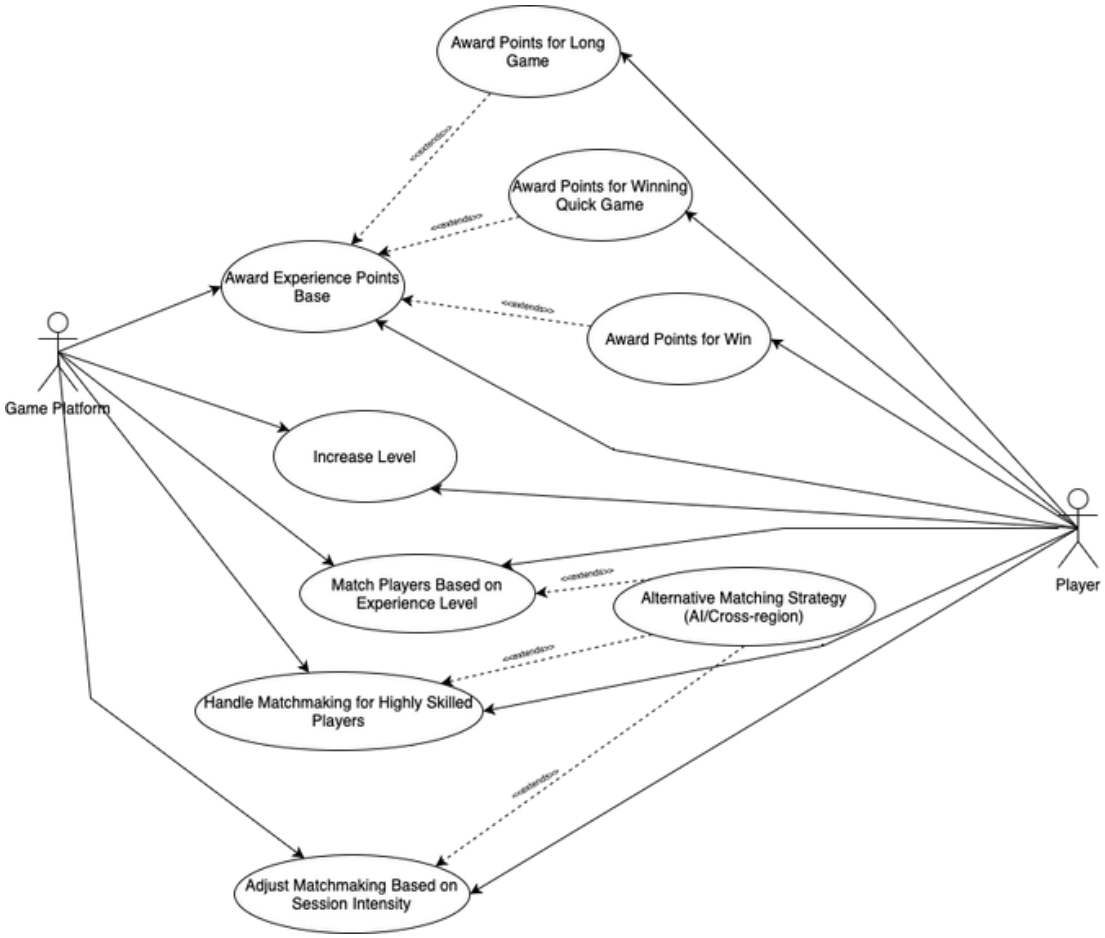
USE CASE DIAGRAM - Leveling



USE CASE DIAGRAM - Matchmaking



USE CASE DIAGRAM - Statistics



Use Case Descriptions: Levelling

Award Points for Win

Iteration: 2, modified March 6

Primary Actor: Game Platform

Goal in Context: For the game to award a user an additional point for winning a game

Preconditions: Platform includes winnable games

Trigger: A player wins and completes a game

Scenario:

1. The player completes a game and is declared the winner
2. The platform awards them 1 point, in addition to the base point given for participation
3. The platform adds the progress to the level experience metric

Exceptions:

- The number of points awarded when added, exceeds the threshold for the current level's experience metric: Refer to Increase Level Use Case

Priority: Essential

When Available: First increment

Frequency of Use: Many times a day

Channel to Actor: Via device

Secondary Actors: The player

Channels to Secondary Actors: Via device and platform

Open Issues: N/A

Award Experience Points Base

Iteration: 2, modified March 6
Primary Actor: Game Platform
Goal in Context: For the game to award a user a point for completing a game
Preconditions: Platform includes games
Trigger: A player completes a game
Scenario:

1. The player completes a game
2. The platform awards them 1 point
3. The platform adds the progress to the level experience metric

Exceptions:

- The number of points awarded when added, exceeds the threshold for the current level's experience metric: Refer to Increase Level Use Case

Priority: Essential
When Available: First increment
Frequency of Use: Many times a day
Channel to Actor: Via device
Secondary Actors: The player
Channels to Secondary Actors: Via device and platform

Open Issues: N/A

Award Points for Long Game

Iteration: 2, modified March 6
Primary Actor: Game Platform
Goal in Context: For the game to award a user a point for completing a long game
Preconditions: Platform includes games, each having a known average length to complete
Trigger: A player completes a game in after several moves or rounds that exceed the average game length **Scenario**:

1. The player completes a game in after several moves or rounds that exceed the average game length
    - For Whist, this is at least 10 rounds
    - For connect four, this is at least 28 turns
    - For checkers, this is at least 32 turns
2. The platform awards them 1 point, in addition to the one base point for participation
3. The platform adds the progress to the level experience metric

Exceptions:

- The number of points awarded when added, exceeds the threshold for the current level's experience metric: Refer to Increase Level Use Case

Priority: Medium-High
When Available: First increment
Frequency of Use: Many times a day
Channel to Actor: Via device
Secondary Actors: The player
Channels to Secondary Actors: Via device and platform
Open Issues: N/A

Award Points for Winning Quick Game

Iteration: 2, modified March 6

Primary Actor: Game Platform

Goal in Context: For the game to award a user more points for winning a game quickly

Preconditions: Platform includes games, each having a known average length to complete

Trigger: A player completes and wins a game in a time below the average number of moves or rounds

Scenario:

1. The player completes a game within a certain number of rounds or turns
    - For Whist, this limit is 3 rounds
    - For connect four, this limit is 9 turns
    - For checkers, this limit is 10 turns
2. The platform awards them 2 points, in addition to the one base point for participation
3. The platform adds the progress to the level experience metric

Exceptions:

- The number of points awarded when added, exceeds the threshold for the current level's experience metric: Refer to Increase Level Use Case

Priority: Medium-High

When Available: First increment

Frequency of Use: Many times a day

Channel to Actor: Via device

Secondary Actors: The player

Channels to Secondary Actors: Via device and platform

Open Issues: N/A

Increase Level

Iteration: 2, modified March 6
Primary Actor: Game Platform
Goal in Context: For the game to increase the user's level after reaching an XP threshold
Preconditions: Platform includes games for which points can be awarded
Trigger: A player has reached the XP threshold to increase their level
Scenario:

1. The player reaches the XP threshold
    - To advance from level 1 to 2, this is 10 points
    - To advance to any levels beyond that, it is the number of points it took to advance from the previous level, multiplied by 1.2
2. The platform awards them 2 points, in addition to the one base point for participation
3. The level experience metric changes to the next level, and calculates the amount of XP in the current level

Exceptions: N/A
Priority: Essential
When Available: First increment
Frequency of Use: Many times
Channel to Actor: Via device
Secondary Actors: The player
Channels to Secondary Actors: Via device and platform
Open Issues: N/A

Use Case Descriptions: Player Matchmaking

Match Players Based on Experience Level

Iteration: 1, modified March 5

Primary Actor: Game Platform

Goal in Context: Match a player with another player whose experience level is close to theirs.

Preconditions:

- The platform includes matchmaking functionality.
- The player has an assigned experience level.
- There are available players with similar experience levels.

Trigger: A player initiates a matchmaking request.

Scenario:

1. The player requests to find a match.
2. The platform searches for available opponents with a similar
3. experience level.
4. If multiple suitable opponents are found, one is selected randomly.
5. The players are placed into a match.

Exceptions:

1. If no suitable match is found, the platform expands the search range or increases wait time before attempting another match.

Priority: Essential
When Available: First increment
Frequency of Use: Many times a day
Channel to Actor: Via Device
Secondary Actors: Players
Channels to Secondary Actors: Via device and platform

Adjust Matchmaking Based on Session Intensity

Iteration: 1, modified March 5

Primary Actor: Game Platform

Goal in Context: Ensure dynamic matchmaking by adjusting matchmaking preferences based on a player's session intensity.

Preconditions:

- The platform includes matchmaking functionality.
- The player has an assigned session intensity score.
- There are available opponents with varying experience levels.

Trigger: A player initiates a matchmaking request.

Scenario:

1. The player requests to find a match.
2. The platform retrieves the player's experience level and session intensity.
3. The platform calculates the effective matchmaking level as (experience level × `session intensity)`.
4. The platform searches for opponents with experience levels within an acceptable range of the effective matchmaking level.
5. A suitable opponent is selected and matched with the player.

Exceptions:

- If no suitable match is found, the platform gradually broadens the search criteria.

Priority: Medium-High

When Available: First increment

Frequency of Use: Many times a day

Channel to Actor: Via device

Secondary Actors: Players

Channels to Secondary Actors: Via device and platform

Handle Matchmaking for Highly Skilled Players

Iteration: 1, modified March 5

Primary Actor: Game Platform

Goal in Context: Ensure highly skilled players can still find matches without excessive wait times.

Preconditions:

- The platform includes matchmaking functionality.
- The player has an assigned experience level and session intensity.
- The player is among the top-ranked players with limited opponents available.

Trigger: A highly skilled player initiates a matchmaking request.

Scenario:

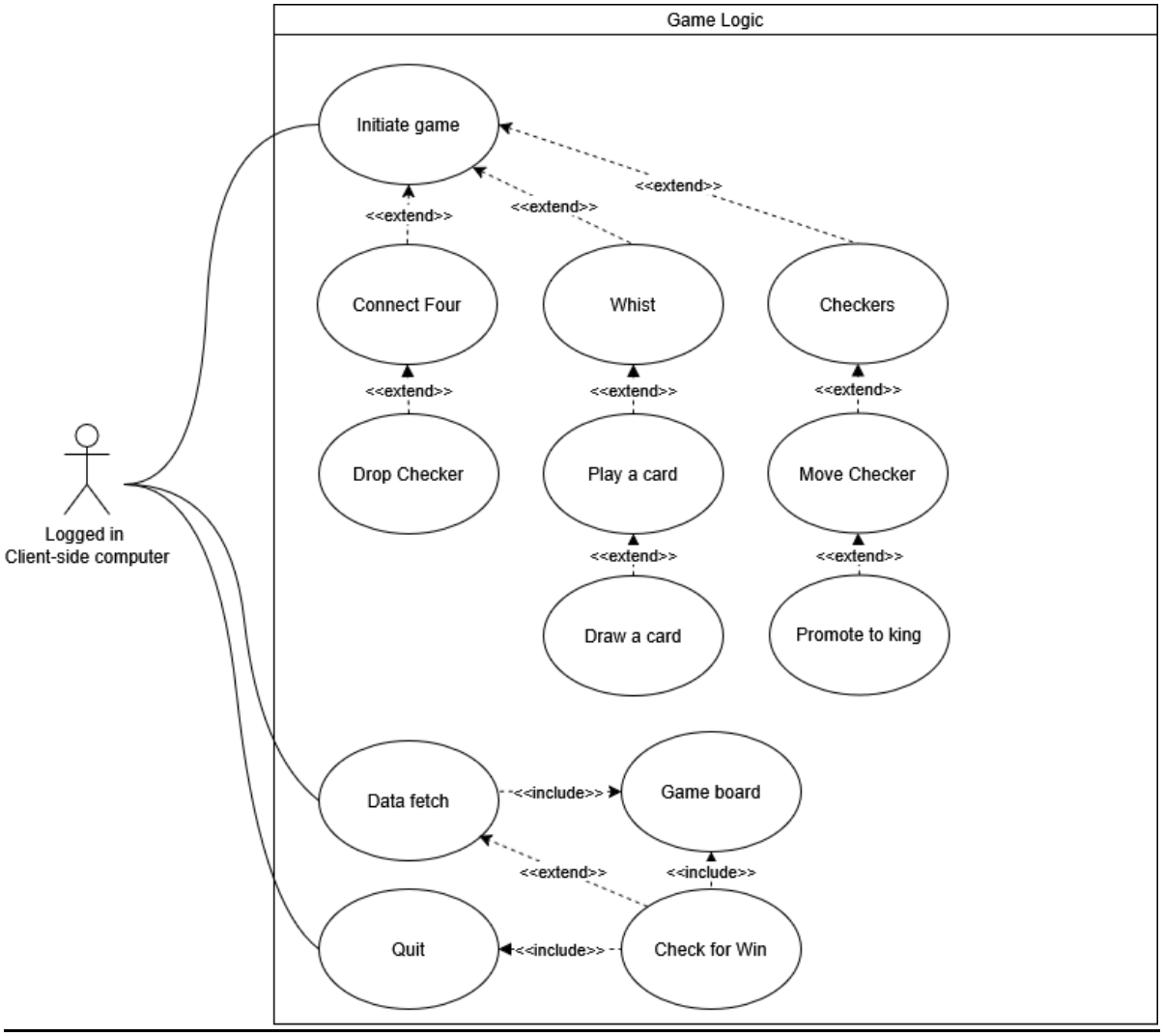
1. The player requests a match.
2. The platform attempts to find an opponent with a similar experience level.
3. If no immediate match is found, the platform gradually expands the matchmaking criteria:
   - Increasing the acceptable range of experience levels.
   - Prioritizing players with high session intensity to maintain a challenge.
4. Once a match is found, the players are paired, and the game begins.

Exceptions:

- If no match is found after an extended period, the platform may offer alternative options, such as AI opponents or matchmaking across regions.

Priority: High
When Available: First increment
Frequency of Use: Multiple times a day
Channel to Actor: Via device
Secondary Actors: Players
Channels to Secondary Actors: Via device and platform

Game Logic Use Case Diagram



Initiating a Game

Iteration: 1

Primary Actor: Player

Goal in Context: Starting a game with another player

Preconditions: The player is logged in

Trigger: The player selects a game and clicks play

Scenario:

1. The game matches another player of similar skill level
2. An object of the game selected is created
3. The UI displays the game board to both players

Postcondition: The selected game is ready to play for both players
Exceptions:
1. The player is matched to themself

Priority: High
When available: First release
Frequency of Use: High, once every game
Channel to actor: Network UI update
Secondary Actors: Game host server
Channel to secondary Actors: Network
Open Issues:

Game Data Fetch by client-side Computer
Iteration: 1
Primary Actor: Client-side UI
Goal in Context: Allow the client to retrieve necessary data for gameplay
Preconditions: The player is logged in and in a game
Trigger: The player initiates a game, the game is refreshed

Scenario:
1. The computer fetches the game data
2. The UI reflects any changes

Postcondition: The user can see the updated game board Exceptions:
1. The client is attempting to fetch unauthorized data

Priority: High
When available: First Release
Frequency of Use: High, every time a game is refreshed
Channel to actor: Network
Secondary Actors: End user, game logic server

Channel to secondary Actors: UI display, network Open
Issues: What should the refresh rate be?

Dropping a Checker (Connect 4)
Iteration: 1
Primary Actor: Active player
Goal in Context: The active player dropping their checker, completing their turn
Preconditions: A valid game is active
Triggers: The beginning of the game and the end of a turn that doesn't result in the end of the game.

Scenario:

1. A game is initiated and validated
2. Player 1 makes the first move

Postcondition: The move reflected in the UI and displayed to both players

Exceptions:

1. The game is not valid, for example, a player is matched with themself
2. A player quits or disconnects, ending the game.

Priority: High, this is the core of the Connect Four gameplay.
When available: First release.
Frequency of Use: High, used multiple times every game.
Channel to actor: Network UI update
Secondary Actors: Passive player, game host server
Channel to secondary Actors: Network Open
Issues:

Moving a Checker (Checkers)
Iteration: 1
Primary Actor: Active Player
Goal in Context: The active player mover their piece to a different square in accordance with the rules of the game
Preconditions: A valid game is active
Trigger: The beginning of the game and the end of a turn that doesn't result in the end of the game.

Scenario:

1. A game is initiated and validated
2. Player 1 makes the first move

Postcondition: The move reflected in the UI and displayed to both players

Exceptions:

1. The game is not valid, for example, a player is matched with themself.
2. A player quits or disconnects, ending the game.
3. Then checker has no legal moves

Priority: High, this is the core of the Checkers gameplay.
When available: First release.
Frequency of Use: High, used multiple times every game.
Channel to actor: Network UI update
Secondary Actors: Passive player, game host server
Channel to secondary Actors: Network Open
Issues:

Promoting to a King (Checkers)
Iteration: 1
Primary Actor: Active player
Goal in Context: The active player promotes one of their men to a King
Preconditions: A valid game is active
Trigger: A man reaches the opposite end of the board

Scenario:

1. Active player moves a man to the opposite end row
2. Active player's piece is changed to a King

Postcondition: The move and promotion reflected in the UI and displayed to both players

Exceptions:

1. The game is not valid, for example, a player is matched with themself.

2. A player quits or disconnects, ending the game.
3. The player has a forced capture

Priority: High, this is a core mechanic Checkers gameplay.
When available: First release.
Frequency of Use: High, used a few times a game.
Channel to actor: Network UI update
Secondary Actors: Passive player, game host server
Channel to secondary Actors: Network Open
Issues:

Play a Card (Whist)
Iteration: 1
Primary Actor: Active Player
Goal in Context: The active player plays a card from their hand
Preconditions: A valid game is active, it is the active player's turn
Trigger: A player clicks a card in their hand on their turn

Scenario:

1. Active player has a turn in the game
2. Active player selects a card to play
3. Active player plays their card

Postcondition: The player has one less card in their hand, their card is on the table
Exceptions:

1. The active player has no cards to play

Priority: High, this is a core feature of the game When available: First release.
Frequency of Use: High, used many times a game.
Channel to actor: Network UI update
Secondary Actors: Passive player, game host server
Channel to secondary Actors: Network Open
Issues:

Draw a Card (Whist)
Iteration: 1
Primary Actor: Active Player
Goal in Context: The active player draws a card from the draw pile
Preconditions: A valid game is active, the game is in the drafting stage
Trigger: A player finishes a trick during the drafting stage

Scenario:

4. Active player has contributed to a trick in the drafting stage
5. The trick is concluded from both parties contributing
6. Active player draws a card from the deck

Postcondition: The player has one more card in their hand, the draw pile is one card smaller Exceptions:

2. The draw pile is empty

Priority: High, this is a core feature of the game When available: First release.
Frequency of Use: High, used many times a game.
Channel to actor: Network UI update
Secondary Actors: Passive player, game host server
Channel to secondary Actors: Network Open
Issues:

Ending a Game
Iteration: 1
Primary Actor: Active player
Goal in Context: Displaying to all players that the game is over, and if they lost, won or drew.
Preconditions: A game is active
Trigger: A wining, losing, or drawing move, or a player quits

Scenario:

1. The server detects that the game is over
2. The players' stats are updated
3. The UI displays to all players whether they won, drew or lost
4. The UI displays the option to leave the game

Postcondition: The game is over
Exceptions: The game will always end

Priority: High
When available: First release
Frequency of Use: High, used every game
Channel to actor: Network
Secondary Actors: Passive players, game logic server Channel to secondary Actors: Network Open
Issues:

Use Case: Signing Up and Setting Up an Account

Iteration: 1, Initial version.

Primary actor: User (New Player)

Goal in context:

- To allow new users to create an account and set up their profile to access the OMG platform.

Preconditions:

- The user has access to the OMG platform via a web browser or application.
- The user is not already registered.

Trigger:

- The user wants to create an account to access the platform.

Scenario:

1. The user navigates to the OMG platform's sign-up page.
2. The system displays a registration form requesting:
    - Username
    - Email address
    - Password
    - Date of birth (for age restrictions)
3. The user fills in the required details and submits the form.
4. The system validates the input fields and checks for:
    - Unique username and email
    - Password strength
    - Proper format for email and date of birth
5. If validation is successful, the system sends a verification email to the user.
6. The user checks their email and clicks the verification link.
7. The system confirms the email and activates the account.
8. The user logs in for the first time using their credentials.
9. The system prompts the user to set up their profile, requesting:
    - Profile picture (optional)
    - Display name
    - Game preferences (optional)
    - Privacy settings
10. The user completes the profile setup and submits the information.
11. The system saves the user's preferences and redirects them to the main dashboard.

Exceptions:

1. Username or email already taken—system prompts the user to choose a different one.
2. Weak password—system suggests a stronger password.
3. Email not verified—system prevents login and reminds the user to verify their email.
4. Forgot password—user can request a reset before completing verification.
5. User skips profile setup—user can proceed to the dashboard and complete setup later.

Priority: Moderate priority, to be implemented in early development.

When available: First increment.

Frequency of use: Frequent.

Channel to actor: Via web browser or application.

Secondary actors:

- System administrator (for account verification issues).
- Customer support (for troubleshooting account creation).

Channels to secondary actors:

- System administration dashboard.
- Customer support ticket system.

Open issues:

1. What mechanisms will be in place to prevent bots and fake accounts?
2. How can the system ensure strong password compliance without frustrating users?
3. Should the platform enforce two-factor authentication (2FA) for added security?
4. Will users have the ability to sign up using third-party authentication (Google, Facebook, etc.)?

Use Case: Logging in with an Existing Account

Iteration: 1, Initial version.

Primary actor: User (Returning Player)

Goal in context:

- To allow returning users to log in and access their OMG platform account.

Preconditions:

- The user has already registered and verified their account.
- The user has access to a web browser or application to log in.

Trigger:

- The user wants to log in to their existing account.

Scenario:

1. The user navigates to the OMG platform's login page.
2. The system displays a login form requesting:
    - Username or email
    - Password
3. The user enters their credentials and submits the form.
4. The system validates the credentials against stored user data.
5. If validation is successful, the system authenticates the user and grants access.
6. The system redirects the user to their dashboard, displaying their profile and game history.
7. If enabled, the system checks for saved preferences and applies them (e.g., dark mode, notifications).

Exceptions:

1. Incorrect username/email or password—system displays an error and prompts the user to try again.
2. Multiple failed login attempts—system may trigger a CAPTCHA or temporarily lock the account.
3. Forgotten password—user can request a password reset via email.
4. Account not verified—system prompts the user to verify their email before logging in.
5. System maintenance—if the platform is under maintenance, a message is displayed with an estimated availability time.

Priority: High priority, required for user access.

When available: First increment.

Frequency of use: Frequent.

Channel to actor: Via web browser or application.

Secondary actors:

- System administrator (for account recovery issues).
- Customer support (for troubleshooting login problems).

Channels to secondary actors:

- System administration dashboard.
- Customer support ticket system.

Open issues:

1. Should the platform support social media logins (Google, Facebook, etc.)?
2. Will two-factor authentication (2FA) be required or optional for enhanced security?
3. How long should login sessions remain active before requiring reauthentication?
4. Should the system notify users of login attempts from new devices or locations?

Use Case: Joining a Match

Iteration: 1, Initial version.

Primary Actor: User (Player)

Goal in Context: To allow users to join a multiplayer match on the OMG platform.

Preconditions:

- The user is logged in to the platform.
- The user has a stable internet connection.
- The user has access to the game library and available matches.
- The match being joined is open and ready for new players.

Trigger:

- The user wants to join an available match either by browsing the game list or accepting an invitation.

Scenario:

1. The user navigates to the match selection screen, either through the game library or a game invite.
2. The system displays available matches, including information such as:
    - Game type
    - Number of players already in the match
    - Current game status (waiting for players, in-progress, etc.)
3. The user selects a match to join.
4. The system checks if the match is still open and if the user meets the necessary criteria (e.g., correct skill level, no restrictions).
5. If the match is available, the system allows the user to join the match and displays a waiting room or the game interface, depending on the game's state.
6. The system notifies the other players in the match that a new player has joined.
7. Once all players have joined, the system begins the game.

Exceptions:

- **Match Full**: If the match is already full, the system notifies the user and suggests other available matches.
- **Match in Progress**: If the match has already started, the system informs the user that the game cannot be joined.
- **User Incompatible**: If the user does not meet the requirements (e.g., skill level mismatch or region restriction), the system will notify the user and suggest an alternative match.
- **Connection Issues**: If the user loses connection while attempting to join, the system will prompt the user to try again or return to the match selection screen.

Priority: High priority, required for multiplayer functionality.

When Available: First increment.

Frequency of Use: Frequent, as joining matches is central to the gameplay.

Channel to Actor: Via web browser or application.

Secondary Actors:

- **Match Host**: The player who created the match.
- **System Administrator**: For managing backend issues, such as matchmaking or server stability.
- **Customer Support**: For troubleshooting connection or account issues related to joining matches.

Channels to Secondary Actors:

- **Match Host**: Notification system or in-game chat for match updates.
- **System Administrator**: Admin dashboard for monitoring match server health and load.
- **Customer Support**: Customer support ticket system for any gameplay-related issues.

Open Issues:

- Should there be a limit on how many matches a player can join in a certain period?
- Will there be matchmaking restrictions (e.g., region-based, skill-based) for certain games?
- Should there be a mechanism for a player to leave a match once they have joined?

Use Case: Tracking Game History

Iteration: 1, Initial version.

Primary Actor: User (Player)

Goal in Context: To allow users to view their past game history, including wins, losses, and other relevant statistics.

Preconditions:

- The user is logged in to the OMG platform.
- The user has participated in one or more matches.
- The system has stored game data such as match results, player statistics, and game types.

Trigger:

- The user wants to view their game history to review their performance or analyze their past games.

Scenario:

1. The user navigates to their profile or dashboard on the OMG platform.
2. The system displays the user's game history section, showing a list of past games including:
    - Game title
    - Match date
    - Opponents
    - Game results (win/loss/draw)
    - Game duration
    - Performance metrics (e.g., score, ranking)
3. The user can filter their game history by date, game type, or performance (e.g., showing only wins or specific games).
4. The user can select a specific game from the list to view more detailed information, such as:
    - A detailed game breakdown (moves, actions, or interactions)
    - Leaderboard positions at the time of the match
    - Any achievements or milestones earned during the game
5. The system allows the user to export or share their game history via social media, email, or within the platform.

Exceptions:

- **No Game History**: If the user has not played any games yet, the system will display a message informing the user that they have no game history available.

- **Data Corruption**: If there's an issue with retrieving the game data, the system will display an error message and suggest the user contact customer support.
- **No Internet Connection**: If the user's connection is lost while trying to load their game history, the system will prompt the user to reconnect.

Priority: High priority, as it is an essential feature for player engagement and tracking progress.

When Available: First increment.

Frequency of Use: Frequent, as players often want to track their progress and review past matches.

Channel to Actor: Via web browser or application.

Secondary Actors:

- **System Administrator**: For ensuring the integrity and availability of game data.
- **Customer Support**: For assisting users with any issues related to accessing their game history.

Channels to Secondary Actors:

- **System Administrator**: Admin dashboard for monitoring data storage and user requests.
- **Customer Support**: Customer support ticket system for troubleshooting issues related to game history retrieval.

Open Issues:

- How long should game history be stored? Should there be an archive or a limit on the number of games shown?
- Should users have the ability to delete or edit their game history?
- Will game history be shared across multiple devices or platforms (cross-platform play)?

Use Case: Sending a Friend Request

Iteration: 1, Initial version.

Primary Actor: User (Player)

Goal in Context: To allow users to send friend requests to other players in order to connect, play together, and interact on the OMG platform.

Preconditions:

- The user is logged into the OMG platform.
- The user has a valid account and is active on the platform.
- The user is able to search for or browse other players' profiles.

Trigger:

- The user wants to add another player to their friends list to connect and play together.

Scenario:

1. The user navigates to the profile or search section to find the player they wish to send a friend request to.
2. The system displays a list of available players or a search function for the user to enter a username or email.
3. The user selects the player they want to send a friend request to.
4. The system displays the selected player's profile, showing their current status (online/offline), games played, and other relevant information.
5. The user clicks on the "Send Friend Request" button on the profile.
6. The system sends the friend request to the selected player and notifies the user that the request has been sent successfully.
7. The system notifies the selected player of the incoming friend request, which they can accept or decline.
8. If the request is accepted, the system adds the users to each other's friends list, and they are able to see each other's online status and send game invites.

Exceptions:

- **Friend Request Already Sent**: If the user has already sent a friend request to the player, the system will display a message stating that the request has already been sent.
- **Player Already in Friends List**: If the user and the selected player are already friends, the system will display a message indicating that they are already connected.
- **User Blocked**: If the selected player has blocked the user, the system will inform the user that they cannot send a friend request.
- **Request Declined**: If the player declines the friend request, the system will notify the user of the declined status.

Priority: Medium priority, as it enhances social interaction but is not essential for the core gameplay.

When Available: First increment.

Frequency of Use: Moderate, as players typically send friend requests to people they want to play with or communicate with more regularly.

Channel to Actor: Via web browser or application.

Secondary Actors:

- **System Administrator**: For monitoring and resolving any issues related to user accounts and the friend request system.
- **Customer Support**: For assisting users with issues related to sending or receiving friend requests (e.g., blocked users, incorrect functionality).

Channels to Secondary Actors:

- **System Administrator**: Admin dashboard for managing system errors and resolving issues with the friend request feature.
- **Customer Support**: Customer support ticket system for troubleshooting and resolving any user complaints or issues related to friend requests.

Open Issues:

- Should there be a limit on how many friend requests a user can send in a day to prevent spam?
- Should friend requests expire if not accepted within a certain period?
- Should the system provide an option to customize friend request messages, or should it be a simple request notification?

Use Case: Quitting a Game

Iteration: 1, Initial version.

Primary Actor: User (Player)

Goal in Context: To allow users to quit a game they are currently playing on the OMG platform, whether due to disconnecting, finishing the game, or choosing to exit early.

Preconditions:

- The user is actively participating in a game on the OMG platform.
- The game is currently running, and the user has control over their game actions.
- The system allows players to leave or quit during gameplay (depends on game type and settings).

Trigger:

- The user wants to quit the game either to stop playing, exit due to a technical issue, or leave for personal reasons.

Scenario:

1. The user decides to quit the game while it is in progress.
2. The system displays a confirmation prompt asking the user if they are sure they want to quit the game.
    - This prompt includes options: **"Yes"** to confirm quitting or **"No"** to cancel and continue playing.
3. If the user selects **"Yes"**, the system proceeds to quit the game.
    - The user's game data (score, progress, etc.) is either saved or discarded, based on game rules and settings.
    - If applicable, the system updates the match status (e.g., marks the game as incomplete, or declares a surrender or forfeiture).
    - The user is returned to the main game menu, dashboard, or match selection screen.
4. If the user selects **"No"**, the game continues, and the user remains in the match.
5. If the user experiences a disconnect (internet loss, crash, etc.), the system attempts to reconnect the player, and if unsuccessful, the game ends, recording the user as a quitter or incomplete participant, depending on game type.

Exceptions:

- **Game State Lock**: If quitting is not allowed due to game rules (e.g., competitive ranked matches), the system will notify the user that quitting is not permitted.
- **Unsaved Progress**: If quitting results in unsaved progress or incomplete matches, the system will warn the user that they may lose any unsaved data.

- **Connection Issues**: If the game quits unexpectedly due to connection problems, the system will try to reconnect the player. If unsuccessful, the game will be marked as incomplete, and the user may be penalized or asked to reconnect.
- **Forced Quit by Server**: If the game server crashes or is shut down, the system will automatically end the game for all players, notifying the users of the issue.

Priority: Medium priority, as quitting is a common part of gameplay but may need to be controlled for certain match types (e.g., competitive modes).

When Available: First increment.

Frequency of Use: Moderate, as players occasionally quit games for various reasons.

Channel to Actor: Via web browser or application.

Secondary Actors:

- **Match Host**: If the user is the host of the game, their quitting may affect the game status for all players.
- **System Administrator**: For monitoring technical issues, including connection problems or server shutdowns that may cause game quitting.
- **Customer Support**: For assisting users with issues related to quitting games or recovering unsaved progress.

Channels to Secondary Actors:

- **Match Host**: In-game chat or notification system for informing other players if the match is ending or disrupted.
- **System Administrator**: Admin dashboard for managing server health, player connection status, and game match integrity.
- **Customer Support**: Customer support ticket system for resolving issues related to quitting games, recovering lost progress, or handling disputes over quitting.

Open Issues:

- Should there be a penalty or consequence for players who quit mid-game in competitive or ranked matches?
- How should the system handle quitting in cooperative games (e.g., does it impact other players)?
- Should there be a confirmation step to prevent accidental quitting during intense gameplay?

Use case: Forgotten Password Reset

Iteration: 1

Primary Actor: User (Player)

Goal in context:

- To allow users to access their account long enough to create a new password.

Preconditions:

- The user has access to the OMG platform via a web browser or an application.
- The user has a registered account with an verified username, email and password.

Trigger:

- The user wants to access their account but can't give the password they registered with.

Scenario:

1. The user navigates to the 'forgot password' screen from the OMG login screen.
2. The screen prompts for:
    a. Email
    b. Username
3. User enters their information.
    a. The system checks both inputs for valid formatting
    b. The system checks the username/email pairing against stored user data.
4. If the input is valid, a temporary password is sent to the provided email address.
    a. The temporary password is deleted from the database 15 minutes after being sent, or right after the user logs in.
5. The system prompts the user to create a new password.
    a. System checks the password's validity.
6. The new password is added to the database.

Post conditions:

- The user has a new password that they can log into their account with.

Exceptions:

1. Username or email not recognized.
    - User is asked to check that they are entering the correct information.
2. Email not associated with username.
    - User is asked to enter a different email.
3. New password isn't formatted in a valid way.
    - System suggests ways to make a stronger password.

Priority: High priority. This feature restores access to the platform in the case that a user forgets their password.

When available: First iteration

Frequency of use: Intermittent

Channel to actor: ‘Forgot password’ screen, via web browser or application.

Secondary actors:

- User data database: database containing usernames, emails and passwords.

Channel to secondary actors:

- User data database: application server for returning database information to the client.

Open issues:

1. Should there be a limit for password reset requests?
2. Should there be a time-out between reset requests?

Use Case: Deleting an Existing Account

Iteration: 1, Initial version.

Primary actor: User (Registered Player)

Goal in context:

- To allow users to permanently delete their OMG platform account along with associated data.

Preconditions:

- The user is logged into their account.
- The user has access to account settings where the deletion option is available.

Trigger:

- The user decides to permanently delete their account.

Scenario:

1. The user navigates to the OMG platform's account settings.
2. The system displays the option to delete the account.
3. The user selects the delete account option.
4. The system prompts the user with a confirmation message warning about data loss.
5. The user confirms the deletion request.
6. The system asks the user to re-enter their password for security verification.
7. The system validates the password and proceeds with the deletion process.
8. The system removes all user data, including profile information, game history, and preferences.
9. The system logs the user out and displays a confirmation message stating the account has been deleted.

Exceptions:

1. Incorrect password entered—system notifies the user and prompts them to try again.
2. User changes their mind—system provides an option to cancel the deletion request before final confirmation.
3. Pending transactions or disputes—if the user has ongoing purchases, refunds, or disputes, the system prevents account deletion until they are resolved.
4. System error—if deletion fails due to technical issues, the system notifies the user and logs the issue for review.

Priority: High priority, necessary for compliance with user privacy rights.

When available: Second increment.

Frequency of use: Infrequent.

Channel to actor: Via web browser or application.

Secondary actors:

- System administrator (for account recovery within a grace period if applicable).
- Customer support (for handling disputes or issues related to account deletion).

Channels to secondary actors:

- System administration dashboard.
- Customer support ticket system.

Open issues:

1. Should there be a grace period (e.g., 30 days) during which users can recover their account before permanent deletion?
2. How should account deletion requests be handled for users with active subscriptions or premium features?
3. Should users receive a final email confirming their account deletion?
4. What level of data anonymization (if any) should be applied instead of full deletion to comply with legal requirements?

Use case: Logging out of account

Iteration: 1

Primary Actor: User (Player)

Goal in context:

- To allow users to log out of their account.

Preconditions:

- The user has a registered and verified account.
- The user is currently logged into their account from a web browser or application.
- The user is not currently in a match.

Trigger:

- The user wants to log out of the account they are currently using.

Scenario:

1. The user initiates logging out through their dashboard.
2. System acts for confirmation from the user.
3. User confirms.
4. The system sends the logout request to the server.
5. The server confirms logout success and session termination.
6. System redirects user to the OMG platform's login page.

Post conditions:

- The user is logged out of their account and taken back to the login page.

Exceptions:

1. Connection interruptions: If the log out attempt is interrupted by connectivity issues, the system will display a message for the user, and log them out.

Priority: High priority. Logging out protects user information and privacy and notifies the system of session termination.

When available: First iteration.

Frequency of use: Frequent

Channel to actor: OMG platform dashboard via web browser or application.

Secondary actors: Server

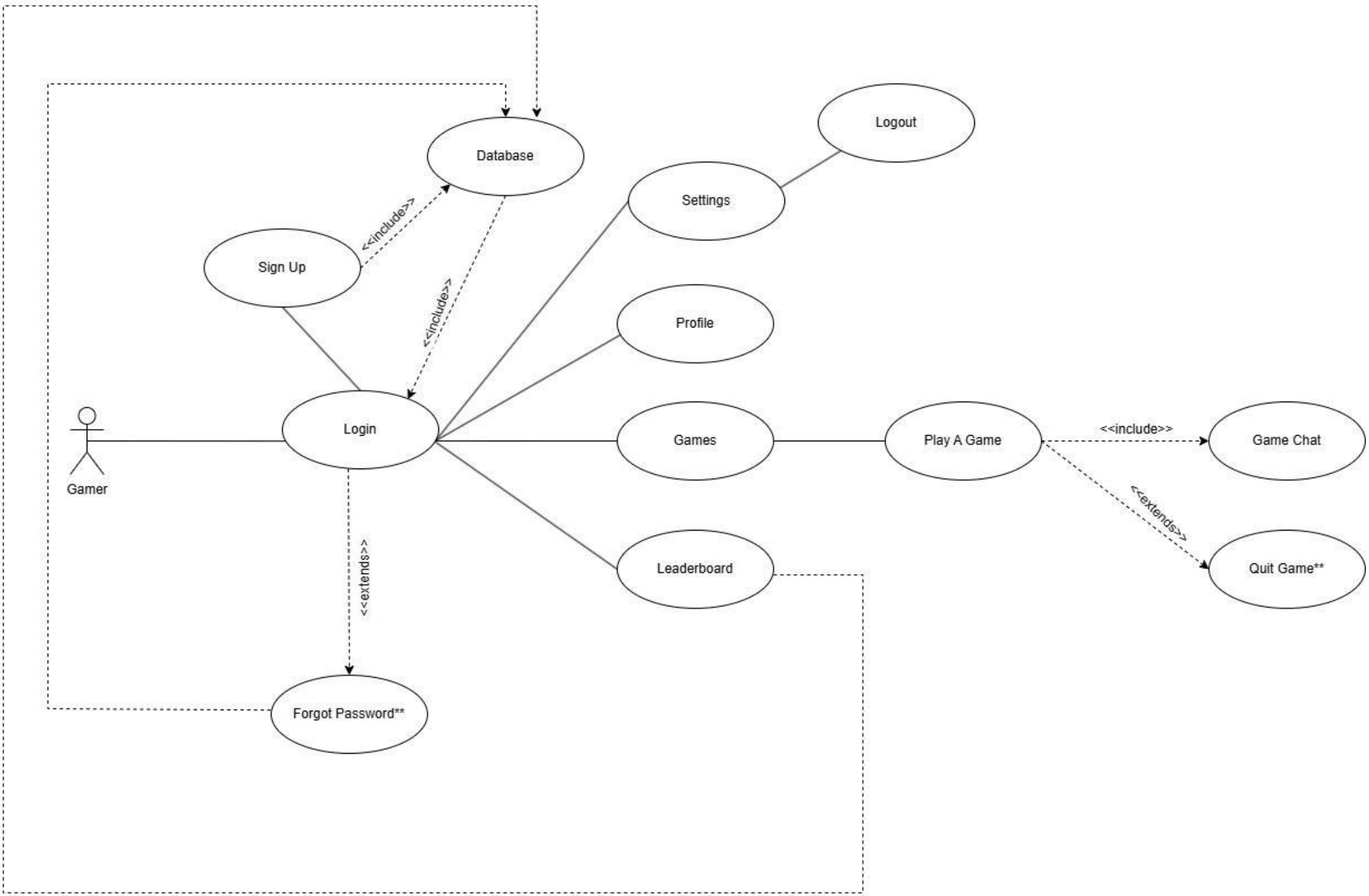
Channel to secondary actors:

- Server: Client-server communication over the network

Open issues:

- N/A

USE CASE DIAGRAM - GUI



Use Case: Login

Iteration: 1

Primary Actor: Existing User

Goal in Context: Allow the existing user to authenticate and gain access to the system's features.

Preconditions:

1. The system is running and accessible.
2. The user has an existing account.

Trigger:

The user attempts to access the system's features.

Scenario:

1. The user enters their username and password.
2. The system verifies the credentials against the database.
3. If the credentials are valid, the user is taken to the main menu.
4. If the credentials are invalid, the system displays an error message and prompts the user to try again.

Post Conditions:

The user is logged in and can access the system's functionalities, or, if the credentials were invalid, an error message is displayed and the user remains logged out.

Exceptions:

1. The user enters an incorrect username or password, prompting an error message and a retry request.
2. The system displays a message indicating that login is unavailable.
3. The user's account does not exist; the system prompts the user to try again, and after successive failures, suggests signing up instead.
4. A system timeout or other technical issues occur.

Priority: High – Logging in is necessary to access other system functionalities.

When Available: Within the first development iteration.

Frequency of Use: Once per session (or as needed if the session times out).

Channel to Actor: The user interacts with the system via the GUI using a keyboard and mouse/touchscreen.

Secondary Actors: The database (for verifying credentials).

Channel to Secondary Actors: The system interacts with the database via database requests.

Open Issues:

1. Should there be a password recovery option?
2. Should multi-factor authentication be implemented?

Use Case: Sign Up

Iteration: 1

Primary Actor: New User

Goal in Context: Allow a new user to create an account.

Preconditions:

1. The system is powered on and accessible.
2. The user is on the sign-in page or has chosen to create a new account.

Trigger:

The user selects the "Sign Up" option.

Scenario:

1. The user selects the "Sign Up" option.
2. The system presents a registration form with fields for username/email, password, and other required information.
3. The user fills in the form.
4. The system validates the entered information (e.g., checks for a unique username and password strength).
5. If the information is valid, the system creates a new user account and stores it in the database.
6. The user is prompted to log in with their new credentials.

Post Conditions:

A new user account is created, and the user can now log in with the newly created credentials.

Exceptions:

1. The username or email already exists.
2. The password does not meet the requirements.
3. There is an error or issue connecting to the database.
4. A system timeout or other technical issues occur.

Priority: High – Sign-up is essential for growing the user base and providing access to new users.

When Available: Within the first development iteration.

Frequency of Use: Once per new user.

Channel to Actor: The user interacts with the system via a graphical user interface (GUI) using a keyboard and mouse/touchscreen.

Secondary Actors: The database (for storing user accounts).

Channel to Secondary Actors: The system interacts with the database via database requests.

Open Issues:

1. Should there be an email verification process?
2. Should a CAPTCHA be included to prevent bots?

Use Case: Forgot Password

Iteration: 1

Primary Actor: Existing User

Goal in Context: Allow the existing user to reset their password.

Preconditions:

1. The system is powered on and accessible.
2. The user has an existing account linked to a valid email address.

Trigger:

The user clicks on the "Forgot Password" link on the login screen.

Scenario:

1. The user selects the "Forgot Password?" option.
2. The system displays a password recovery form requesting the registered email address (or username).
3. The user enters their email address and submits the form.
4. The system verifies whether the email is associated with an account.
5. If the email is valid, the system sends a password reset link to the user's email via the email subsystem stub.
6. The system displays a confirmation message indicating that reset instructions have been sent.

Post Conditions:

The user receives an email containing a password reset link with instructions for creating a new password.

Exceptions:

1. If the entered email is not linked to any account, the system displays an error message prompting the user to check their input.

2. If the email cannot be sent due to network or system issues, the system notifies the user and suggests retrying later.

Priority: Medium – Essential for account recovery and maintaining user access, though not a core gameplay function.

When Available: Within the first development iteration.

Frequency of Use: Infrequent – only used when a user forgets their password.

Channel to Actor: The user interacts with the system via a graphical user interface (GUI) using a keyboard and mouse/touchscreen.

Secondary Actors: The email subsystem (stubbed to simulate sending the reset instructions).

Channel to Secondary Actors: N/A

Open Issues:

1. Should there be a validity period for the reset link?
2. Should additional security measures (e.g., security questions) be implemented before allowing a password reset?

Use Case: Games

Iteration: 1

Primary Actor: User

Goal in Context: Allow the user to view all available games.

Preconditions:

The user is successfully logged in.

Trigger:

The user logs in successfully.

Scenario:

1. The user logs in successfully.
2. The system displays a list of available games.

3. The user browses the list and selects a game to join or test.

Post Conditions:

The user selects a game and can play it.

Exceptions:

1. The user is unable to log in.
2. There are network connection issues.
3. A system timeout or other technical issues prevent the start of the gaming session.

Priority: High – Viewing the game library is essential for providing gameplay.

When Available: Within the first iteration.

Frequency of Use: Multiple times per user, depending on their engagement.

Channel to Actor: The user interacts with the system via a graphical user interface (GUI) using a keyboard and mouse/touchscreen.

Secondary Actors: Game Server (for managing game sessions and matchmaking).

Channel to Secondary Actors: The system interacts with the game server via network communication.

Open Issues: N/A

Use Case: Play A Game

Iteration: 1

Primary Actor: User

Goal in Context: Allow the user to join a game session.

Preconditions:

The user is logged in and has selected a game to play.

Trigger:

The user selects a game to play.

Scenario:

1. The user selects the game to play.
2. The system adds the user to the selected game session.
3. The system matches the player with someone of a similar skill level.
4. The user is taken to the game interface.

Post Conditions:

The user is added to the selected game session and can play the game.

Exceptions:

1. No active game sessions are available.
2. The selected game session is full.
3. There are network connection issues.
4. A system timeout or other technical issues prevent the start of the gaming session.

Priority: High – Joining a game is a core functionality for multiplayer interaction.

When Available: Within the first iteration.

Frequency of Use: Multiple times per user, depending on their engagement.

Channel to Actor: The user interacts with the system via a graphical user interface (GUI) using a keyboard and mouse/touchscreen.

Secondary Actors: Game Server (for managing game sessions and matchmaking).

Channel to Secondary Actors: The system interacts with the game server via network communication.

Open Issues:

1. Should there be requirements (e.g., rank, invite-only) to join certain games?
2. What happens if the game starts while the user is joining?

Use Case: Game Chat

Iteration: 1

Primary Actor: User

Goal in Context: Allow users to communicate with each other during a game session.

Preconditions:

The user is logged in and has successfully joined a game session.

Trigger:

The user wants to send a message to other players.

Scenario:

1. The user selects a game to play from the library.
2. The user successfully joins a game session.
3. The user accesses the chat interface.
4. The user types a message.
5. The user sends the message to their opponent(s).
6. The message is displayed to all players in the game session.

Post Conditions:

The message is sent and displayed to the other players.

Exceptions:

1. Network connection issues prevent the message from being sent.
2. A system timeout or other technical issues occur.

Priority: Medium – Game chat enhances social interaction and communication during gameplay, but it

is not required for gameplay.

When Available: Within the first iteration.

Frequency of Use: Multiple times per user during a game session.

Channel to Actor: The user interacts with the system via a graphical user interface (GUI) using a keyboard and mouse/touchscreen.

Secondary Actors: The game server (for relaying chat messages).

Channel to Secondary Actors: The system interacts with the game server via network communication.

Open Issues:

1. Should private messages or voice chat be implemented in future iterations?
2. Should there be filters for inappropriate or offensive words and terms?

Use Case: Quit Game

Iteration: 1

Primary Actor: User

Goal in Context: Allow users to exit the game application.

Preconditions:

1. The system is running.

Trigger:

The user selects "Quit Game."

Scenario:

1. The user selects "Quit Game" from the menu.
2. The system prompts for confirmation.
3. The user confirms, and the system closes the application.

Post Conditions:

The application exits successfully.

Exceptions:

1. The system fails to close due to an error.
2. The user cancels the quit action before confirming.

Priority: High – Essential for user control.

When Available: Within the first development iteration.

Frequency of Use: Occasionally.

Channel to Actor: The user interacts with the GUI via a keyboard and mouse/touchscreen.

Secondary Actors: None.

Channel to Secondary Actors: None.

Open Issues:

1. Should there be a warning if the user quits with unsaved progress?

Use Case: Leaderboard

Iteration: 1

Primary Actor: User

Goal in Context: Allow the user to view the leaderboard and see player rankings.

Preconditions:

The user is successfully logged in.

Trigger:

The user clicks on the "Leaderboard" button.

Scenario:

1. The user is logged in.
2. The user clicks on the "Leaderboard" button.
3. The system retrieves the leaderboard data from the database.
4. The system displays the leaderboard with player rankings.

Post Conditions:

The user is able to view the leaderboard with up-to-date rankings.

Exceptions:

1. The leaderboard data is not available due to a database connection error, a database error, or a system timeout.

Priority: Medium – Viewing the leaderboard can enhance competition and engagement but is not mandatory for gameplay.

Frequency of Use: Multiple times per user, depending on their interest.

Channel to Actor: The user interacts with the system via a graphical user interface (GUI) using a keyboard and mouse/touchscreen.

Secondary Actors: The database (for player ranking information).

Channel to Secondary Actors: The system interacts with the database via database requests.

Open Issues:

1. Should there be filtering or sorting options for different leaderboard categories?

Use Case: Profile

Iteration: 1

Primary Actor: User

Goal in Context: Allow users to view and update their profile information.

Preconditions:

1. The user is logged in.

Trigger:

The user selects "Profile" from the main menu.

Scenario:

1. The user selects "Profile."
2. The system retrieves profile data from the database.
3. The system displays profile details (e.g., username, avatar, game stats).
4. The user updates the information if desired.
5. The system saves the changes.

Post Conditions:

The user's profile is updated successfully.

Exceptions:

1. The system cannot retrieve or save profile data due to technical issues.
2. The user enters invalid data (e.g., a username that is already taken).

Priority: Medium – Enhances user experience.

When Available: Within the first development iteration.

Frequency of Use: Occasionally.

Channel to Actor: The user interacts with the system via a GUI using a keyboard and mouse/touchscreen.

Secondary Actors: The database (to store user preferences).

Channel to Secondary Actors: The system interacts with the database via queries.

Open Issues:

1. Should users be able to delete their profile?
2. Should profile updates require verification?

Use Case: Settings

Iteration: 1

Primary Actor: User

Goal in Context: Allow users to configure system and game preferences.

Preconditions:

1. The user is logged in.

Trigger:

The user selects "Settings" from the main menu.

Scenario:

1. The user selects "Settings."
2. The system displays available options (audio, controls, graphics, notifications).
3. The user modifies the settings and confirms the changes.
4. The system saves the new settings.

Post Conditions:

The new settings are applied successfully.

Exceptions:

1. The system fails to save changes due to a technical issue.
2. The user cancels the changes before saving.

Priority: Medium – Enhances user experience.

When Available: Within the first development iteration.

Frequency of Use: Occasionally.

Channel to Actor: The user interacts with the GUI via a keyboard and mouse/touchscreen.

Secondary Actors: The database (to store user preferences).

Channel to Secondary Actors: The system interacts with the database via queries.

Open Issues:

1. Should settings be stored locally or in the cloud?

2. Should there be a reset-to-default option?

Use Case: Logout

Iteration: 1

Primary Actor: User

Goal in Context: Allow users to securely end their session.

Preconditions:

1. The user is logged in.

Trigger:

The user selects "Logout."

Scenario:

1. The user selects "Logout."
2. The system asks for confirmation.
3. The user confirms the logout.
4. The system ends the session and redirects the user to the login screen.

Post Conditions:

The user is logged out and must log in again to access features.

Exceptions:

1. The system fails to end the session due to a technical issue.

Priority: High – Ensures security.

When Available: Within the first development iteration.

Frequency of Use: Once per session.

Channel to Actor: The user interacts with the GUI via a keyboard and mouse/touchscreen.

Secondary Actors: The database (to update session status).

Channel to Secondary Actors: The system interacts with the database via queries.

Open Issues:

1. Should there be an option to stay logged in?