Chess Use Cases:

Use case: Play Chess

Iteration: 1

Primary actor: Player

Goal in context: Allows player to start a game of chess

Preconditions:

- The Game app has been opened
- The app is on the game menu

Trigger: The user selects the start game button

Scenario:

- The game enters play mode

Exceptions:

Postconditions:

- The game is now in play mode and the user can begin playing the game

Channel to actor: The user input via GUI or command line

Open Issues:

- How is the opponent chosen, will each game implement a way of handling the opponent selection or will this be implemented by some other system done before the start game is selected
- How do we determine who plays first?

Use case: End Game

Iteration: 1

Primary actor: Player

Goal in context: Quit the game

Preconditions:

- the game is currently in the play mode
- the game is not finished (neither player has won)

Trigger: The player presses the Quit Game button

Scenario:

- The game stops
- The app returns to the game menu

Exceptions:

Postconditions:

- The app in game menu mode

Channel to actor:

- user input via GUI

Open Issues:

- If a player hits the quit game button should it show a summary of the game so far before it returns to the main menu , should it display one of those annoying "are you sure you want to quit" pop ups.

Use case: Check Win (Checkmate)

Iteration: 1

Primary actor: Player

Goal in context: Display game stats and show who won

Preconditions:

- The game is in the play mode
- The last move resulted in a checkmate

Trigger: Last move resulted in checkmate

Scenario:

- The game should display stats to the players via the GUI

- The game should make visible the Exit button

Exceptions:

Postconditions:

- the Exit button is visible to the player

Channel to actor: GUI

Open Issues:

- how will the stats be displayed when the game ends, If it is through an overlay popup or by going to a different GUI scene then there would need to be an exit button, If not then the quit game button can function as the exit button

Use case: Exit Game on Finish

Iteration: 1

Primary actor: Player

Goal in context: Exit the game

Preconditions:

- the game is currently in the game won mode

Trigger: The player presses the Exit Game button

Scenario:

- The app returns to the game menu

Exceptions:

Postconditions:

- The app in game menu mode

Channel to actor:

- user input via GUI

Open Issues:

Use case: Check Draw

Iteration: 1

Primary actor: Player

Goal in context: Display game stats and say it is a draw

Preconditions:

- The game is in the play mode
- The last move resulted in a draw

Trigger: Last move resulted in a dead position (draw)

Scenario:

- The game should display stats to the players via the GUI
- The game should make visible the Exit button

Exceptions:

Postconditions:

- the Exit button is visible to the player

Channel to actor: GUI

Open Issues:

- how will the stats be displayed when the game ends, If it is through an overlay popup or by going to a different GUI scene then there would need to be an exit button, If not then the quit game button can function as the exit button

Use Case: Capture

Iteration: 1
Primary Actor: Player
Goal in context: Show that a chess piece has been captured **Preconditions**:

- The game is in play mode
- The player is able to move a piece into a square occupied by an enemy piece during their turn

Trigger: The player moves one of their pieces into a square occupied by an enemy piece
Scenario:

- The piece occupying the space should be replaced by the piece being moved
- The piece removed should be tallied
- Game should check the game state

Exceptions: Post Conditions:

- The piece that previously existed in the space has been removed from the board and been replaced by the moved piece

Channel to actor: Graphical Representation of the game board **Open Issues:** N/A

Checkers Use Cases:

Use Case: Play Checkers

Primary Actor: Player and Opponent

Iteration: 1

Goal in Context: Start a new game of checkers

Precondition: Both players have opened the game, and start a game with the matchmaking system, or rematch button

Trigger: The matchmaking system creates a match with the player and opponent

Scenario:

- The player and opponent with similar skill levels both look for a new match in the matchmaking system
- The matchmaking system see's that have a similar skill ranking, and pairs them together
- The matchmaking system creates a new game between the player and opponent

Post Condition: The player and opponent can play checkers against each other

Exceptions: N/A

Priority: High

When Available: Iteration 1, will likely be the 1st thing implemented

Frequency of Use: once per game

Channel to Actor: GUI/Network code

Secondary Actors: N/A

Open Issues: N/A

Use Case: Exit Game

Primary Actor: Player

Iteration: 2

Goal in Context: Let the player leave the once the game is over

Precondition: The game is finished and the results screen is being shown (see Game Complete)

Trigger: The player selects the exit game button

Scenario:

- The game of checkers ends, and the results screen is shown
- The player clicks the exit game button
- The opponents stops being allowed to rematch
- The game is closed

Post Condition: The game closes

Exceptions: N/A

Priority: Medium - High

When Available: Once results screen is implemented

Frequency of Use: Once per session

Channel to Actor: GUI

Secondary Actors: Opponent

Open Issues: N/A

Use Case: End Game

Primary Actor: Player

Iteration: 1

Goal in Context: Show the player the results of the game, and give the option to rematch or exit

Precondition: The player and opponent are in the middle of a game

Trigger: The game has ended from either player winning or forfeiting

Scenario:

- A player wins the game, and signals to show the ending screen
- The end screen is shown, with the results of the game
- The player is able to rematch or quit the game

Post Condition: The player see their results and can rematch or quit the game

Exceptions: N/A

Priority: High

When Available: Iteration 1, after game rules are implemented

Frequency of Use: Once per game

Channel to Actor: GUI

Secondary Actors: N/A

Open Issues: N/A

Use Case: Check Win

Primary Actor: Player

Iteration: 1

Goal in Context: Let the player know they won the game, and update their rating/statistics

Precondition: The player finished their move, and it is the opponent's turn

Trigger: The opponent has no pieces they can move on their turn

Scenario:

- The player jumps the opponent's last piece, or blocks all their moves
- The opponent has no pieces they can move on their turn
- The players win rating gets updated
- The game complete screen is shown, letting them know they won

Post Condition: The results screen is shown (see game complete)

Exceptions: N/A

Priority: High

When Available: Iteration 1, with other game rules implementation

Frequency of Use: Up to once a game

Channel to Actor: GUI

Secondary Actors: N/A

Open Issues: N/A

Use Case: Check Loss

Primary Actor: Opponent

Iteration: 1

Goal in Context: Let the player know they lost the game, and update their rating/statistics

Precondition: The opponent has finished their move, and it is the player's turn

Trigger: The player has no pieces they can move on their turn

Scenario:

- The opponent jumps the player's last piece, or blocks all their moves

    - The player has no pieces they can move on their turn
    - The players win rating gets updated
    - The game results screen is shown, letting them know they won

Post Condition: The results screen is shown (see game complete)

Exceptions: N/A

Priority: High

When Available: Iteration 1, with other game rules implementation

Frequency of Use: Up to once a game

Channel to Actor: GUI

Secondary Actors: Player

Open Issues: N/A

TTT Use Cases:

Use Case: Play TTT

Primary Actor: Player 1 or Player 2
Goal: initiate a new TicTacToe game

Preconditions:

- The players must be authenticated in the OMG system
- Both players must be available
- System must be ready to host a new game

Trigger: a player selects to start a new game

Scenario:

- Player starts a new game
- System verifies both players are ready
- System randomly assigns X/O symbols to players
- System initializes an empty board
- System sets first players turn

Exceptions:

- Issue with player authentication
- Issues initializing game board
- Can't find a second player after player 1 selects to start a new game
- Network connection issues

Postconditions:

- New game starts with an empty board, and 2 players with assigned symbols.

Priority: High

When Available: When both players are authenticated OR when 2 different authenticated players select to start a game and neither are in an active game

Frequency of use: once per game session

Channel to actor: GUI interface

Open Issues:

- Should players choose their own symbol or be randomly assigned?
- How do 2 players get matched for a game?
- Should there be a time limit to initiate the game after a play selects to start a new game.

Use Case: Check Win

Primary Actor: N/A (triggered by Make Move)
Goal: To determine if a winning condition has been met

Preconditions:

- A valid move has just been made
- A game is in progress

Trigger: A move has just been completed

Scenario:

- System checks all rows for 3 matching symbols
- System checks all columns for 3 matching symbols
- System checks both diagonals for 3 matching symbols
- If a win is found, the system moves to the end game use case
- If no win is found, the system moves to the check draw use case

Exceptions:

- System fails to check the win condition

Postconditions:

- Either the game is ended or the system checks for a draw

Priority: High

When Available: After a move is made

Frequency of use: Multiple times per game

Channel to actor: Internal system process

Secondary Actors: N/A

Open Issues: N/A

Use Case: Check Draw

Primary Actor: N/A (triggered by Check Win)
Goal: To determine if the game has reached a draw (cat's game)

Preconditions:

- No win has occurred
- A valid move has just been made

Trigger: No win after a move has been made

Scenario:

- System checks if all of the board positions have been filled
- If this is true, the system moves to the end game use case
- If there are still empty positions, the system moves to the switch turn use case

Exceptions:

- System fails at verifying board state

Postconditions:

- Either the game has ended as a draw or it's the next player's turn

Priority: High

When Available: After checking for a win condition after a move has been made

Frequency of use: Multiple times per game session

Channel to actor: Internal system process

Secondary Actors: N/A

Open Issues:

- Should a draw be determined before all the positions are full?

Use Case: End Game

Primary Actor: N/A (triggered by Check Win, Check Draw, or Forfeit)
Goal: Properly conclude a game session that has concluded without any errors

Preconditions: Win, draw, or forfeit occurs in an active game

Trigger: A game has either met the win or draw condition, or a player decides to forfeit

Scenario:

- System verifies the active game has either met a win or draw condition, or that a player has selected to forfeit
- The system stops accepting new moves
- The system notifies both players that the game has ended
- The game is deactivated

Exceptions:

- System fails to deactivate the game

Postconditions:

- Game concluded and results given to each player (either one of them has won or it's a draw)

Priority: High

When Available: After a win or draw condition has been met, or a player forfeits

Frequency of use: Once per game session

Channel to actor: GUI interface (if forfeit) or internal system process (if win or draw)

Open Issues: N/A

Use Case: Disconnect Player

Primary Actor: System
Goal: Handle unexpected player disconnections

Preconditions:

- Game is active
- Connection monitoring is active

Trigger: Player connection is lost

Scenario:

- System detects player disconnection
- System attempts reconnection
- If reconnection fails, initiate game deletion

Exceptions:

- False disconnection detected
- System inability to detect disconnection

Postconditions:

- Game will move on to the delete game use case

Priority: Medium

When Available: During an active game

Frequency of use: Rarely

Channel to actor: Internal system process

Secondary Actors: Players? Network?

Open Issues:

- What should the reconnection time window be?

Use Case: Update Player Data

Primary Actor: N/A (triggered by End Game or Delete Game)
Goal: To maintain accurate player statistics

Preconditions: Game has concluded or been deleted, player records are accessible

Trigger: Game has ended or been deleted

Scenario:

- System identifies game outcome (win, loss, draw, deletion)
- System retrieves player records
- System updates relevant statistics according to game outcome
- System saves updated records

Exceptions:

- Database update failure

Postconditions: Player statistics updated in the database

Priority: High

When Available: Once an active game becomes deactivated/has been concluded

Frequency of use: Once per game

Channel to actor: Internal system process

Secondary Actors: Database

Open Issues:

- For failed games, should we just use this to ensure their player statistics are the same as they were before the game started?
- What are the relevant statistics that should be provided? Do player stats include time spent on each game, number of wins, losses, draws, etc?

Leaderboard System Use Cases

Use Case: View Leaderboard Rankings

Primary Actor: Player
Goal: Player can view and navigate through different leaderboard rankings and filters
Precondition:

- Player is logged into the game
- Player has stable internet connection
- Leaderboard data is available

Trigger: Player selects "Leaderboards" option from the game menu
Scenarios:

- System displays the default leaderboard view (typically current season, all modes)
- Player can:
    - Scroll through rankings showing player names, ranks, scores, and relevant stats
    - Toggle between time periods (daily/weekly/monthly/all-time)
    - Filter by game modes/categories
    - See their own ranking highlighted in the list
    - View detailed statistics for each entry
- System automatically updates rankings in real-time
- Player can refresh the leaderboard manually

Accessibility Features:

- Color-blind friendly ranking indicators
- Screen reader compatibility
- Keyboard navigation support
- Adjustable text size

Exceptions:

- If network connection fails:
    - System displays cached data with timestamp
    - Shows network error message
    - Provides retry option
- If no data available for selected filter:
    - Display "No rankings available" message
    - Suggest alternative filters

Postcondition: Player can view and understand their ranking position relative to others
Priority: High
Frequency of use: Multiple times per session
Channel to actor: Game interface, touch/mouse input
Secondary Actors: Game Server
Channel To Secondary Actor: API calls, WebSocket connections

Use Case: View Leaderboard Rankings
Primary Actor: Player
Goal: Player can view worldwide rankings and compare their performance globally

Precondition:
- Player is logged in
- Global ranking system is operational
- Player has completed at least one ranked game/match

Trigger: Player selects "Global Rankings" tab in leaderboard interface
Scenarios:
- System loads global ranking data
- Player can:
    - View top players worldwide
    - See country/region indicators for each player
    - Compare their score with global averages
    - View global ranking distribution

- System shows player's percentile in global rankings
- Player can bookmark specific players to track

Accessibility Features:
- Region-specific formatting for numbers and dates
- Multi-language support
- High contrast mode

Exceptions:
- If region data unavailable:
    - Display simplified ranking without region info
    - Show notice about missing data
- If player hasn't qualified for global ranking:
    - Show requirements to qualify
    - Display projected ranking based on current score

Postcondition: Player understands their global standing
Priority: Medium
Frequency of use: Daily
Channel to actor: Game interface
Secondary Actors: Regional Servers
Channel To Secondary Actor: Cross-server synchronization

Use Case: View Friend Ranking
Primary Actor: Player
Goal: Player can view and compare rankings among their friend list
Precondition:
- Player is logged in
- Player has at least one friend added
- Friends system is operational

Trigger: Player selects "Friend Rankings" tab in leaderboard interface
Scenarios:
- System loads friend list and their ranking data

- Player can:
    - View rankings of all friends
    - See online/offline status
    - Compare scores directly
    - Challenge friends
    - Share scores with friends
    - Sort by different metrics (total score, recent activity, etc.)
- System highlights recent changes in friend rankings
- Player can send congratulatory messages for achievements

Accessibility Features:

- Customizable friend nicknames
- Activity status indicators
- Quick action shortcuts

Exceptions:

- If friend list empty:
    - Show friend recommendation system
    - Provide quick add friend option
- If friend's data is private:
    - Show privacy indicator
    - Option to request viewing permission

Postcondition: Player can effectively track and compare performance with friends
Priority: Medium-High
Frequency of use: Multiple times per session
Channel to actor: Game interface, social features
Secondary Actors: Friend System Server
Channel To Secondary Actor: Social API calls

Use Case: Search Profiles
Iteration: 1
Primary Actor: Player
Goal: Players should be able to search for other players and view their profile/statistics. Also be able to challenge other players.
Preconditions: Player has logged onto the game and is connected to network
Trigger: Player initiates a search by selecting the search feature
Scenario:

- Player enters search input (ex: username)
- System retrieves and displays all players matching the search text input
- Player selects the specific player they want to view from the search result list

- System displays the selected player’s profile and statistics
- Optional: Player clicks the “Challenge” button on the selected player’s profile
- System sends a challenge notification to the chosen player
- Optional: selected player receives a game notification and can accept or decline the challenge

Post Condition: Player successfully views searched player’s profile/statistics / successfully sends a challenge invitation
Exceptions:

- “No Results Found” - no players match the search input
- Network issues prevent searching
- If challenge is sent, the recipient may not respond

Priority: High
When Available: Always
Frequency of Use: Everytime players open the game and as often as they want to challenge
Channel to Actor: Text Input Search feature (user interface)
Secondary Actor: Challenged player
Channel to Secondary Actor: Challenge notification
Open Issues: privacy option for profiles

Use Case: Reset Leaderboard
Iteration: 1
Primary Actor: Admin
Goal: Clear the leaderboard to start a new season or remove specific categories.
Preconditions: Admin has logged into the system with appropriate permissions
Trigger: Admin selects the "Reset Leaderboard" option.
Scenario:

- Admin selects the "Reset Leaderboard" interface.
- System displays options for resetting seasonal rankings or specific leaderboard categories.
- Admin chooses to reset either the seasonal rankings or a specific category.
- System clears the chosen leaderboard data and prepares it for new rankings.

Post Condition: The selected leaderboard is successfully reset.
Exceptions:

- Database error: If the system is unable to reset due to data corruption or access issues,
- Network error: If the network connection fails, the reset is delayed until connectivity is restored.

Priority: High
When Available: At the end of a season or upon request by Admin.
Frequency of Use: Occasional, typically at season start or for special resets.
Channel to Actor: Admin dashboard/settings menu.
Secondary Actor: N/A

Channel to Secondary Actor: N/A
Open Issues: N/A

Use Case: Configure Leaderboard Settings
Iteration: 1
Primary Actor: Admin
Goal: Set rules and categories for player rankings on the leaderboard.
Preconditions: Admin has logged into the system with appropriate permissions.
Trigger: Admin selects the "Configure Leaderboard Settings" option.
Scenario:

- Admin opens the "Configure Leaderboard Settings" interface.
- System displays options to set up scoring rules, define ranking periods, and manage categories.
- Admin configures scoring rules (e.g., points per win, win-loss ratio).
- Admin sets the ranking period (e.g., daily, weekly, seasonal).
- Admin manages categories by adding, editing, or removing categories.
- System saves and applies the settings across all leaderboards.

Post Condition: Leaderboard configurations are updated, with new rules and categories in effect.
Exceptions:

- Validation error: If settings violate existing rules (e.g., overlapping periods), the system displays an error.
- Database error: If settings cannot be saved, the system informs the admin and retries automatically.

Priority: Medium
When Available: Always
Frequency of Use: Occasional, typically at the beginning of each season or when new leaderboard rules are required.
Channel to Actor: Admin dashboard/settings menu.
Secondary Actor: N/A
Channel to Secondary Actor: N/A
Open Issues: N/A

Networking Use Cases

Use Case : Online Chat
Primary Actor: Player
Goal: The player can send chat messages to their opponent during a game session, with the messages stored in the database for future reference.
Preconditions: The players are in an active game session and connected to the server.
Trigger: The player initiates a chat message.
Scenario:

- Player types a chat message.
- System sends the message
- System forwards the message to the opponent.
- Opponent receives the message in their chat interface.

Postcondition: The message is stored in the database and visible to the opponent.
Exceptions:

- Player disconnects during the message send.
- Database fails to store the message.

Priority: Medium
Frequency of Use: Frequent (during gameplay)
Channel to Actor: In-game chat window
Secondary Actor: Server
Channels to Secondary Actors: Network connection to server
Open Issues: Determine how to handle message storage if the database is unavailable.

Use Case: Connect Player
Primary Actor: Player
Goal: Connect the player to the server to access online features.
Preconditions: Player is logged in.
Trigger: Player initiates a connection.
Scenario:

- Player attempts to connect.
- System calls for player details.
- Server adds a player to the active user list.
- Connection is confirmed for the player.

Postcondition: Player is connected to the server.
Exceptions: Server connection issue.
Priority: High
Frequency of Use: Every login

Channel to Actor: Platform interface
Secondary Actor: Server
Channels to Secondary Actors: Network connection
Open Issues: Define reconnection policy on connection loss.

Use Case: ReceiveMessage
Primary Actor: Player
Goal: Receive a chat message from an opponent.
Preconditions: Players are in the same game session.
Trigger: Opponent sends a chat message.
Scenario:

- Opponent sends a message.
- Server forwards the message to the player.
- System displays the message in the chat interface.

Postcondition: Message is displayed to the player.
Exceptions: Server connection error.
Priority: Medium
Frequency of Use: Frequent
Channel to Actor: In-game chat interface
Secondary Actor: Server
Channels to Secondary Actors: Network connection
Open Issues: None.

Use Case 8: Shutdown
Primary Actor: Server Administrator
Goal: Shutdown the server, disconnecting all players.
Preconditions: None.
Trigger: Administrator initiates shutdown.
Scenario:

- Administrator initiates shutdown.
- Server disconnects all players.
- Server shuts down.

Postcondition: All players are disconnected; server resources are freed.
Exceptions: Players not disconnected correctly.
Priority: Low
Frequency of Use: Occasional (maintenance)
Channel to Actor: Server console
Secondary Actor: Players
Channels to Secondary Actors: Network connection
Open Issues: Define notifications for unexpected shutdowns.

Authentication Use Cases

Use case: Registration

Iteration: 1

Primary Actor: User

Goal in context: Allow a user to create a player profile to access the Online Multiplayer Game (OMG) platform.

Preconditions: Platform has booted up, user is at the home screen, user wants to create a personal profile to enter the platform.

Trigger: User presses the ‘Register’ button.

Scenario:

1. Registration screen pops-up on the program.
2. User enters a username, email and password to associate with their profile
3. A verification code is sent to the user’s email
4. User opens their email to find the code
5. User enters the code in the program
6. User profile creation is complete.
7. Registration screen closes and the program returns to the home screen.

Post conditions: User can now log-in to the platform using their personal profile.

Exceptions:

8. Invalid username is entered
9. Invalid email is entered therefore the user can never locate a verification code
10. Password does not have a minimum of 8 characters, 1 capital, 1 number and 1 punctuation letter.
11. Invalid verification code is entered

Priority: High

When available: 1st Iteration.

Frequency of use: High

Channel to Actor: Opening program on computer via application.

Secondary Actor: N/A.

Channels to secondary actors: N/A.

Open Issues: User profile does not get saved in the database, verification code is not sent to the user's email.

Use case: Delete Profile

Iteration: 1

Primary Actor: User

Goal in context: Allow a use to delete a player profile from the Online Multiplayer Game platform

Preconditions: Platform has booted up, a user with the username and password exists in the system, user wants to delete the profile

Trigger: User presses the delete button, might also have to press a confirmation button to delete

Scenario:

12. Delete user screen pops up on the program
13. User enters the username and password for the user to be deleted
14. A verification code is sent to the user's email
15. User opens their email to find the code
16. User enters the code into the program
17. User has been deleted successfully
18. Delete user screen closes and the program goes to the home screen

Post conditions: User no longer exists in the program and cannot login using the same credentials again

Exceptions:

1. Invalid username is entered
2. Invalid password entered for that username
3. Invalid verification code entered from email

Priority: High

When available: Always

Frequency of use: Medium

Channel to Actor: User accesses the Online Multiplayer Game platform through the computer application interface, specifically the profile deletion screen.

Secondary Actor: N/A

Channels to secondary actors: N/A

Open Issues: User profile does not exist in the database, verification code not successfully sent to the user's email

Use case: Edit Profile (Username)

Iteration: 1

Primary Actor: User

Goal in context: Allow user to edit existing profiles username

Preconditions: The platform must be booted, the user must be logged in, the user must exist in the database, user wants to edit profile's username

Trigger: User clicks the "Confirm" button to change username

Scenario:

1. User goes to profile edit screen
2. User clicks edit username
3. Username change screen pops up
4. User types in new username
5. User clicks confirm button
6. Username has been successfully changed message pops up
7. Program returns to the profile screen where the updated username is shown

Post conditions: Username has been successfully changed in the database, user can no longer login using the old username

Exceptions:

1. Invalid username entered for the specific profile in the database

Priority: High

When available: Always

Frequency of use: Low

Channel to Actor: User accesses the Online Multiplayer Game platform through the computer application interface, specifically the profile edit screen for username updates.

Secondary Actor: N/A

Channels to secondary actors: N/A

Open Issues: N/A

Use case: Edit Profile (Password)

Iteration: 1

Primary Actor: User

Goal in context: Allow user to edit existing profiles password

Preconditions: The platform must be booted, the user must be logged in, the user must exist in the database, user wants to edit profile's username

Trigger: User clicks the "Confirm" button to change password

Scenario:

1. User goes to profile edit screen
2. User clicks edit password
3. Password change screen pops up
4. User types in new password
5. User clicks confirm button
6. Password has been successfully changed message pops up
7. Program returns to the profile screen

Post conditions: Password has been successfully changed in the database, user can no longer login using the old password

Exceptions:

1. Invalid password entered for the specific profile in the database

Priority: High

When available: Always

Frequency of use: Low

Channel to Actor: User accesses the Online Multiplayer Game platform through the computer application interface, specifically the profile edit screen for password updates.

Secondary Actor: N/A

Channels to secondary actors: N/A

Open Issues: N/A

Use case: Edit Profile (Email)

Iteration: 1

Primary Actor: User

Goal in context: Allow user to edit existing profiles email

Preconditions: The platform must be booted, the user must be logged in, the user must exist in the database, user wants to edit profile's username

Trigger: User clicks the "Confirm" button to change email

Scenario:

1. User goes to profile edit screen
2. User clicks edit email
3. Email change screen pops up
4. User types in new password
5. Verification code is sent to registered email
6. User types in the verification code
7. Verification email sent to new email
8. User types in verification code from new email
9. User clicks confirm button
10. Email has been successfully changed message pops up
11. Program returns to the profile screen

Post conditions: Email has been successfully changed in the database, user now receives news and verification codes on new email

Exceptions:

1. Invalid new email entered
2. Invalid verification code entered

Priority: High

When available: Always

Frequency of use: Low

Channel to Actor: User accesses the Online Multiplayer Game platform through the computer application interface, specifically the profile edit screen for updating their email address.

Secondary Actor: N/A

Channels to secondary actors: N/A

Open Issues: Verification code never received on new email or old email

Use case: Edit Profile (Update Bio)

Iteration: 1

Primary Actor: User

Goal in context: Allow user to edit an existing profile's bio.

Preconditions: The platform must be booted, the user must be logged in, the user must exist in the database, user wants to edit profile's bio

Trigger: User clicks the "Confirm" button to change bio

Scenario:

1. User navigates to the Edit Profile screen
2. User clicks the "Edit Bio" option
3. A pop-up for editing the bio opens up
4. User types in the new bio
5. User clicks the "Confirm" button
6. The system saves the updated bio to the database
7. The program returns to the profile screen, displaying the updated bio

Post conditions: The user's bio has been successfully updated in the database and is shown on the user's profile

Exceptions:

8. The user enters prohibited content into the bio (e.g. inappropriate language)
9. The database fails to connect, preventing the bio from being updated
10. The bio exceeds the allowable field length

Priority: Medium

When available: 2nd iteration

Frequency of use: Moderate; users may occasionally update their bio

Channel to Actor: User accesses the Online Multiplayer Game platform through the computer application interface, specifically the profile edit screen for updating their bio.

Secondary Actor: N/A

Channels to secondary actors: N/A

Open Issues: N/A

Use case: Edit Profile (Update Avatar)

Iteration: 1

Primary Actor: User

Goal in context: Allow user to edit an existing profile's avatar.

Preconditions: The platform must be booted, the user must be logged in, the user must exist in the database, user wants to edit profile's avatar

Trigger: User clicks the "Confirm" button to change avatar

Scenario:

1. User navigates to the Edit Profile screen
2. User clicks the "Edit Avatar" option
3. A pop-up screen allowing the user to either "Upload New Avatar" or "Delete Avatar" opens up
4. User either uploads a new image to use as their avatar, or deletes their current avatar
5. User clicks the "Confirm" button
6. The system saves the updated avatar to the database
7. The program returns to the profile screen, displaying the updated avatar

Post conditions: The user's avatar has been successfully updated in the database and is shown on the user's profile. If the user chose to delete their avatar, their avatar is replaced with a default avatar.

Exceptions:

1. The user uploads a prohibited image into the avatar

2. The database fails to connect, preventing the avatar from being updated

Priority: Medium

When available: 2nd iteration

Frequency of use: Moderate; user may occasionally update their avatar

Channel to Actor: User accesses the Online Multiplayer Game platform through the computer application interface, specifically the profile edit screen for updating their avatar.

Secondary Actor: N/A

Channels to secondary actors: N/A

Open Issues: N/A

Use case: Edit Profile (Change Status)

Iteration: 1

Primary Actor: User

Goal in context: To update the profile's display status.

Preconditions: The user has a valid profile and is logged in.

Trigger: User clicks the button to change status

Scenario:

1. The user is logged in and selects option to view their profile.
2. User selects option to update status.
3. From drop-down menu, selects either public (anyone can view), private (friends only), or anonymous (no one can view).

Post conditions: The system updates the user profile to the new preferences set by the user.

Exceptions:

1. User selects already preferred display status.

Priority: Low

When available: Always

Frequency of use: Low

Channel to Actor: Button/drop-down menu while displaying user profile.

Secondary Actor: N/A

Channels to secondary actors: N/A

Open Issues:

- How will private and anonymous profiles be stored?
    - Will they show up in search results when users look for other users?

Use case: Login

Iteration: 1

Primary Actor: User

Goal in context: To let the User be able to log into their account to access the games on the platform.

Preconditions: User must have already successfully created a profile.

Trigger: User wants to log in.

Scenario:

1. User inputs username and password.
2. User clicks ‘Sign In”.
3. Display is changed to main home screen.

Post conditions: User profile data and settings will be updated to reflect the User’s account for use and access, such as editing their profile and logging out. User will also be able to access other features of the platform, like selecting games to play and checking out rankings.

Exceptions:

1. Username or password is invalid.
2. User has not created a profile yet.

Priority: High

When available: First iteration.

Frequency of use: High.

Channel to Actor: Clicking the “Sign In” button.

Secondary Actor: None.

Channels to secondary actors: N/A

Open Issues: Cannot locate or verify username and password in database.

Use case: Log Out

Iteration: 1

Primary Actor: User

Goal in context: To let the User be able to log out of the system.

Preconditions: User must already be logged in.

Trigger: User wants to log out of platform.

Scenario:

1. User clicks on the “Sign Out” button.
2. Prompt appears confirming whether the User is sure they want to log out.
3. User selects “Yes.”
4. System logs User out.
5. Screen is redirected back to the Sign In/Register screen.

Post conditions: User has been signed out of the system. The screen shows the main sign in/register page, and the system is ready to accept another username and password to be entered.

Exceptions:

1. User selects “No” after the prompt confirming the sign out appears. The prompt will then disappear, showing whatever screen the User was viewing before pressing the “Sign Out” button.

Priority: High.

When available: First iteration

Frequency of use: High.

Channel to Actor: Clicking the “SIgn Out” button..

Secondary Actor: None.

Channels to secondary actors: N/A

Open Issues: None.

Use case: Add Friend

Iteration: 1

Primary Actor: User

Goal in context: To let the User be able to add other players as part of their friends list.

Preconditions: User must already be signed in, and the User has successfully searched for the desired player.

Trigger: User wants to befriend another player on the platform.

Scenario:

1. User selects the “Add Friend” button next to the desired player’s username in the list of search results.
2. Friend request is sent to the desired player.

Post conditions: Player is notified of the new request. If the friend request is accepted, see use case for Accept Friend Request; otherwise, see use case for Reject Friend Request.

Exceptions:

1. If a friend request has already been sent to the same player, and it hasn’t been responded to yet, the “Add Friend” button should not be clickable.

Priority: Medium

When available: First iteration

Frequency of use: Moderate.

Channel to Actor: Clicking the “Add Friend” button after successfully searching up the desired player.

Secondary Actor: None.

Channels to secondary actors: N/A

Open Issues: None.

Use case: Remove Friend

Iteration: 1

Primary Actor: User

Goal in context: To let the User be able to remove a player from their friends list.

Preconditions: The User must already be signed in, and the player must already be part of the User's friends list. User must also have their friends list open showing the desired player.

Trigger: User wants to remove a player from their friends list.

Scenario:

1. User selects desired player from their friends list.
2. User selects "Remove Friend" button.

Post conditions: The player is removed from the User's friends list, and the User is removed from the player's friends list. Either User or player is able to search for the other account and send a new friend request after this has been completed.

Exceptions:

1. None.

Priority: Low

When available: Second iteration.

Frequency of use: Low/moderate.

Channel to Actor: Clicking "Remove Friend" button after selecting the player from the User's friends list.

Secondary Actor: None.

Channels to secondary actors: N/A

Open Issues: None.

Use case: Accept Friend Request

Iteration: 1

Primary Actor: User

Goal in context: To allow a user to accept a friend request received from another user, adding them to their friend list.

Preconditions: The user is logged into the application, the user has received a pending friend request from another user.

Trigger: The user navigates to the “Friend Requests” section.

Scenario:

2. The user navigates to the “Friend Requests” section.
3. The program displays a list of pending friend requests.
4. The user selects the friend request they wish to accept.
5. The user accepts a specific friend request.
6. The system displays a confirmation message.
7. The program removes the pending friend request from both user’s Friend Requests.
8. The program updates the friend lists of both users.

Post conditions: The friend request is removed from the pending friend requests list, both users are now added to each other’s friends lists, both users are notified of the new friendship status, the database is updated with the new friends list for both users.

Exceptions:

9. The friend request is withdrawn before the user can accept it
10. The database may fail to update with the acceptance of the friend request

Priority: High

When available: 1st iteration

Frequency of use: Moderate

Channel to Actor: User accesses the Online Multiplayer Game platform through the computer application interface, specifically the Friend Requests section for viewing received requests.

Secondary Actor: Second user (Potential Friend)

Channels to secondary actors: Sending a friend request through button press on a user's profile.

Open Issues: N/A

Use case: Reject Friend Request

Iteration: 1

Primary Actor: User

Goal in context: To allow a user to reject a friend request received from another user, thereby not adding them to their friend list.

Preconditions: The user is logged into the application, the user has received a pending friend request from another user.

Trigger: The user navigates to the "Friend Requests" section.

Scenario:

11. The user navigates to the "Friend Requests" section.
12. The program displays a list of pending friend requests.
13. The user selects the friend request they wish to decline.
14. The user declines a specific friend request.
15. The system displays a confirmation message.
16. The program removes the pending friend request from both user's Friend Requests.
17. The program does not update the friend lists of both users.

Post conditions: The friend request is removed from the pending friend requests list, the friends lists of both users are unchanged, the second user is not notified of the rejection of their friend request.

Exceptions:

1. The friend request is withdrawn before the user can accept it
2. The database may fail to update with the rejection of the friend request, keeping the request in the user's list of pending friend requests.

Priority: High

When available: 1st iteration

Frequency of use: Moderate

Channel to Actor: User accesses the Online Multiplayer Game platform through the computer application interface, specifically the Friend Requests section for viewing received requests.

Secondary Actor: Second user (Potential Friend)

Channels to secondary actors: Sending a friend request through button press on a user's profile.

Open Issues: N/A

GUI Use Cases

Use case: View Profile
Primary Actor: User
Goal in context: User wants to view their profile information, check their stats, and records.
Preconditions: User is logged into their account.
Trigger: User clicks on “view profile” option from navigation menu.
Scenario:

- User clicks on the “view profile” option.
- Their profile is retrieved from the database.
- The system displays all of their login credentials and gaming statistics (in categories).
- Users view their profile.

Post conditions: User successful views their profile.
Exceptions:

- Network error: asks user to try again.

Priority: Medium - high.
When available: This would be available in the second iteration.
Frequency of use: Depends on the player.
Channel to Actor: User interface.
Secondary Actor: Database (retrieves info).
Channels to secondary actors: System sends request to database to retrieve user’s information.
Open Issues:

- Should private info like your password be hidden?

Use case: Browse and Join Games
Iteration: 1
Primary Actor: user
Goal in context: allow the user to browse available games and join a selected game
Preconditions:

- User is logged in
- Platform has a library of available games

Trigger: user navigates to the game library screen
Scenario:

- User opens the game library screen to view available games
- GameLibraryScreen displays a list of available games

- User scrolls through the list and selects a game from the list to join
- GameLibraryScreen calls backend services to initiate joining
- GameLibraryScreen navigates the user to the GameScreen

Post conditions: user joined the game and is directed to the GameScreen
Exceptions:

- Network or Connection Error

Priority: High
When available: first increment
Frequency of use: frequent - used each time a user wants to join a game
Channel to Actor: touchscreen or click-based interface
Secondary Actor: backend service
Channels to secondary actors: calls to initiate game joining
Open Issues:

- Should there be an option to view additional game details before joining?

Use case: Play Game
Iteration: 1
Primary Actor: User
Goal in context: Enable the user to interact with the game board, make moves, receive feedback on invalid moves, and exit the game, returning to the game library.
Preconditions:

- User must be logged in.
- Game type is selected
- The system is ready to initiate the game

Trigger: The user enters the GameScreen after selecting a game from the game library.
Scenario:

- User selects "Play Game" and chooses a specific game
- GameScreen displays the selected game board
- User initiates a move
- GameScreen sends the move to the backend for validation
- GameScreen receives the backend response
    - If the move is valid, GameScreen updates the board to reflect the move
    - If the move is invalid, display an error message. Explaining why the move is invalid

- GameScreen checks if the game has reached an end state (win, loss, or draw) (extends specific game rules for winning, draw conditions)
- If the game is not over, the next turn begins

- User wants to exit the game
    - User clicks the “Exit Game” Button
    - GameScreen triggers a navigation action, returning the user to the game library screen

Post conditions:
- Game board reflects the user’s last valid move.
- If the user exits, they are returned to the game library.

Exceptions:
- Invalid Move
- Network or connection error

Priority: High - core function for gameplay
When available: First increment
Frequency of use: frequently - every time a user plays a game
Channel to Actor: Touchscreen or click-based UI for game interactions
Secondary Actor: backend server
Channels to secondary actors: validate moves, check game state, retrieve responses
Open Issues:
- Should there be an option to automatically save game progress if the user exits mid-game?

Use case: Play TicTacToe (extends Play Game)
Iteration: 1
Primary Actor: User
Goal in Context: Allow the user to play TicTacToe
Preconditions:
- User must be logged in.
- User selected to play TicTacToe

Trigger: user selects “TicTacToe” in the game menu
Scenario:
- User selects to play “TicTacToe”
- GameScreen displays 3x3 grid for TicTacToe with an indicator showing what the player is (X or O)
- User selects an empty cell on the grid to place their symbol
- GameScreen sends the selected move to the backend for validation
    - If the move is invalid, GameScreen displays an error message prompting the user to select a different cell
    - If the move is valid, GameScreen updates the grid

- GameScreen displays the opponent’s turn indicator
- GameScreen receives and displays the opponent’s move on the grid

- Steps 3-6 repeat for each player until
    - Win: GameScreen displays win message
    - Draw: GameScreen displays draw message
    - Lose: GameScreen displays lost message

- User selects “Exit Game”
- Game Screen returns the user to the GameLibrary screen

Post conditions:
- Game board reflects the user’s last valid move.
- If the user exits, they are returned to the game library.

Exceptions:
- Invalid Move
- Network or connection error

Priority: High - core function for gameplay
When available: First increment
Frequency of use: frequently - every time a user plays a game
Channel to Actor: Touchscreen or click-based UI for game interactions
Secondary Actor: backend server
Channels to secondary actors: validate moves, check game state, retrieve responses
Open Issues:N/A

Use case: Play Chess (extends Play Game)
Iteration: 1
Primary Actor: User
Goal in Context: Allow the user to play Chess
Preconditions:
- User must be logged in.
- User selected to play Chess

Trigger: user selects “Chess” in the game menu
Scenario:
- User selects Chess from the game option
- GameScreen displays an 8x8 chessboard with pieces arranged in their standard starting positions. Each player is assigned a color (white or black)
- User selects a piece on the chessboard and chooses a destination to move it
- GameScreen sends the selected move to the backend for validation
    - If the move is invalid, GameScreen displays an error message prompting the user to select a valid move
    - If the move is valid, GameScreen updates the board

- GameScreen displays the opponent’s turn
- GameScreen receives and displays the opponent’s move on the chessboard
- Steps 3-6 repeat until one of these conditions are met:
  - Win: GameScreen displays win message
  - Draw: GameScreen displays draw message
  - Lose: GameScreen displays lost message

- User selects “Exit Game” option
- GameScreen returns the user to the GameLibrary Screen

Post conditions:
- Game board reflects the user’s last valid move.
- If the user exits, they are returned to the game library.

Exceptions:
- Invalid Move
- Network or connection error

Priority: High
When available: First increment
Frequency of use: frequently - every time a user plays a game
Channel to Actor: Touchscreen or click-based UI for game interactions
Secondary Actor: backend server
Channels to secondary actors: validate moves, check game state, retrieve responses
Open Issues:N/A

Use case: Play Checkers (extends Play Game)
Iteration: 1
Primary Actor: User
Goal in Context: Allow the user to play Checkers
Preconditions:
Trigger: user selects “Checkers” in the game menu
Scenario:
- User selects Checkers from the game option
- GameScreen displays an 8x8 checker board with pieces in their starting positions. Assign player color
- User selects a piece and initiates a move by choosing a destination square
- GameScreen sends the move to the backend for validation
  - If the move is invalid, GameScreen displays an error message, prompting the user to try a different move
  - If the move is valid, GameScreen updates the board

- GameScreen displays the opponent’s turn
- GameScreen receives and displays the opponent’s move on the board
- Steps 3-6 repeat until one of these condition is met:
  - Win: GameScreen displays win message

    - Draw: GameScreen displays draw message
    - Lose: GameScreen displays lost message

- User selects “Exit Game”
    - GameScreen returns the user to the GameLibrary screen

Post conditions:
- Game board reflects the user’s last valid move.
- If the user exits, they are returned to the game library.

Exceptions:
- Invalid Move
- Network or connection error

Priority: High
When available: First increment
Frequency of use: frequently - every time a user plays a game
Channel to Actor: Touchscreen or click-based UI for game interactions
Secondary Actor: backend server
Channels to secondary actors: validate moves, check game state, retrieve responses
Open Issues:N/A

Use case: In-Game Chat
Iteration: 1
Primary Actor: User
Goal in context: To allow the user to type a message into the chat box, and then accurately transmit that message to a universal text box that other users can see.
Preconditions:
• User must be logged in
• User has joined a game through the game library
Trigger: The enter button has been pressed after the message has been typed in the message box.

Scenario:
- User is already actively playing with someone in a game (specific game doesn’t matter).
- User enters a message into the text box, and presses Enter to submit the message.
- The message is transmitted via the internet to the other user’s chat box, while also printing the same message on the sender’s own box.

Post conditions:
- The message that has been entered appears on both user’s chat boxes.

Exceptions:

- If the internet disconnects and the message doesn't register on the other user's chat box.

Priority: Low
When available: First Increment
Frequency of use: Depends on the people playing the game.
Channel to Actor: Click-based UI
Secondary Actor: Backend service
Channels to secondary actors: Calls to send message
Open Issues:

- Is it necessary to have chat rooms for the type of games that will be implemented?

Use case: View Leaderboard Rankings
Iteration: 1
Primary Actor: User
Goal in context: To allow the user to see an ordered list of the players with the most successes.

- **Preconditions**:
- The user is connected to the internet
- User must be logged in

Trigger: The user pressed the button indicating the leaderboard rankings on the main page.
Scenario:

- The user selects the Leaderboard Rankings button.
- The user can choose between the three games available (Chess, TTT, and Checkers)
- A table is made for each game showing the top users and how many wins they have compared to their losses.

Post conditions:

- A table is made from the system database showing the best players for each game if chosen.

Exceptions:

- If the game doesn't have any players, a table can't be made.

Priority: Low - Only for informational purposes and doesn't affect the system.
When available: First Iteration
Frequency of use: Low - Isn't necessary for using the system.
Channel to Actor: Click-based UI
Secondary Actor: Database
Channels to secondary actors: System calls to database to retrieve general player database
Open Issues:

- What would be the most optimal way to update player base data?

Use case: View Player Profile from Leaderboard
Iteration: 1
Primary Actor: User

Goal in context: To allow the user to see the statistics of a specific player from the Leaderboard
Preconditions:

- User has logged in
- User has prompted the leaderboard to open and a table has been formed

Trigger: The player selects a specific user highlighted on the Leaderboard
Scenario:

- The user selects the Leaderboard Rankings button.
- The user chooses between the three games available (Chess, TTT, and Checkers)
- The user selects a specific player, showing their statistics specific for that game (ELO for chess, win/loss for TTT and Checkers, etc.)

Post conditions:

- The data collected by the system is presented as statistics and shown on the user's UI.

Exceptions:

- If the selected user has deleted their account and their data can't be traced back.

Priority: Low - Only for informational purposes and doesn't affect the system.
When available: First Iteration
Frequency of use: Low - Isn't necessary for using the system.
Channel to Actor: Click-based UI
Secondary Actor: Database
Channels to secondary actors: System calls to database to retrieve specific data of a user from the general playerbase
Open Issues: See issue from prior use case
Use case: Modify Settings
Iteration: 1
Primary Actor: User
Goal in context: To allow the user change and modify the game and UI to their preference using provided settings.
Preconditions:

- User has logged in

Trigger: The settings button has been selected, and the settings are presented on the UI.
Scenario:

- The user logged into the system
- The user selects the settings button
- The user chooses between different categories of settings (graphic, UI, sound, etc.)
- The user changes the settings which affects the system, and saves their preferences

Post conditions:

- The settings that the user chooses is saved to the user's local system until changed again.

Exceptions:

- The settings interferes with the user's hardware and can't be changed

Priority: High
When available: First/Second Iteration
Frequency of use: Depends on the user, but generally low since the user will initially find what works for them.
Channel to Actor: Click-based UI

Secondary Actor: Backend system
Channels to secondary actors: Calls and changes data as instructed by the user.
Open Issues: What settings are the most necessary for this system to have?

Use case: View Notifications
Iteration: 2
Primary Actor: User
Goal in context: To allow users to view important notifications about their account, ranking or games they're playing.
Preconditions:

- User is logged in.
- User has notifications enabled.

Trigger: Something has changed with the user's account, ranking or game they're playing.
Scenario:

- For each notification, a new line of text is added to a section on the main screen.
- Account Notification - Notifies the user which account they're logged in with.
- Ranking Notification - Notifies the user if they're leaderboard ranking has increased, decreased or stayed the same.
- Game Notification - Notifies the user if their position in a queue has changed or the outcome of a completed game; or a disconnect message if the user disconnects partway through.

Post conditions: The line stays in the notification section until the user decides to clear it.
Exceptions: N/A
Priority: Moderate
When available: Second Increment
Frequency of use: High if notifications are enabled, None if notifications are disabled.
Channel to Actor: Screen
Secondary Actor: N/A
Channels to secondary actors: N/A
Open Issues:

- Should notifications be able to be disabled mid-game?
- Does opening a game obstruct the notification section?

Use case: Clear Notifications
Iteration: 2
Primary Actor: User
Goal in context: To allow the user to clear the notification section on the main screen.
Preconditions:

- User is logged in
- User has notifications enabled
- There are lines of text in the notification section.

Trigger: The user decides to clear the text in the notification section.

Scenario:
- The user presses a button labeled “Clear Notifications”
- The system gives the user a confirmation message.
- The user selects “Yes”.

Post conditions: All lines of text are cleared from the notification section.
Exceptions:
- The user selects “No” when asked for confirmation.
- The notification section is already empty.

Priority: Moderate
When available: Second Increment
Frequency of use: Moderate, None if user has notifications disabled.
Channel to Actor: Button
Secondary Actor: N/A
Channels to secondary actors: N/A
Open Issues: N/A

Use case: Access Support
Iteration: 1
Primary Actor: User
Goal in context: Allow the user to access resources for troubleshooting, FAQs and contacting support.
Preconditions: There is something wrong with the user’s account, screen, or commands that they are unsure how to fix.
Trigger: The user presses the “Access Help and Support” button.
Scenario:
- The dropdown menu provides options for “Troubleshooting”, “FAQs” and “Contact Support”.
- Troubleshooting - Provides a list of common issues and ways they could potentially be solved.
- FAQs - Provides a list of questions asked by users that admin have seen and answered often.
- Contact Support - Provides an email or phone number that the user can use to contact administration regarding their issue if it is not covered in “Troubleshooting” or “FAQs”.

Post conditions: The user acquires the resources to try and resolve the issue themself or contact. administration for support.
Exceptions:
- The issue is something out of admin’s control (user’s internet or hardware)

Priority: High
When available: First Increment
Frequency of use: Moderate
Channel to Actor: Button
Secondary Actor: Administration
Channels to secondary actors: Administration’s screen

Open Issues: N/A