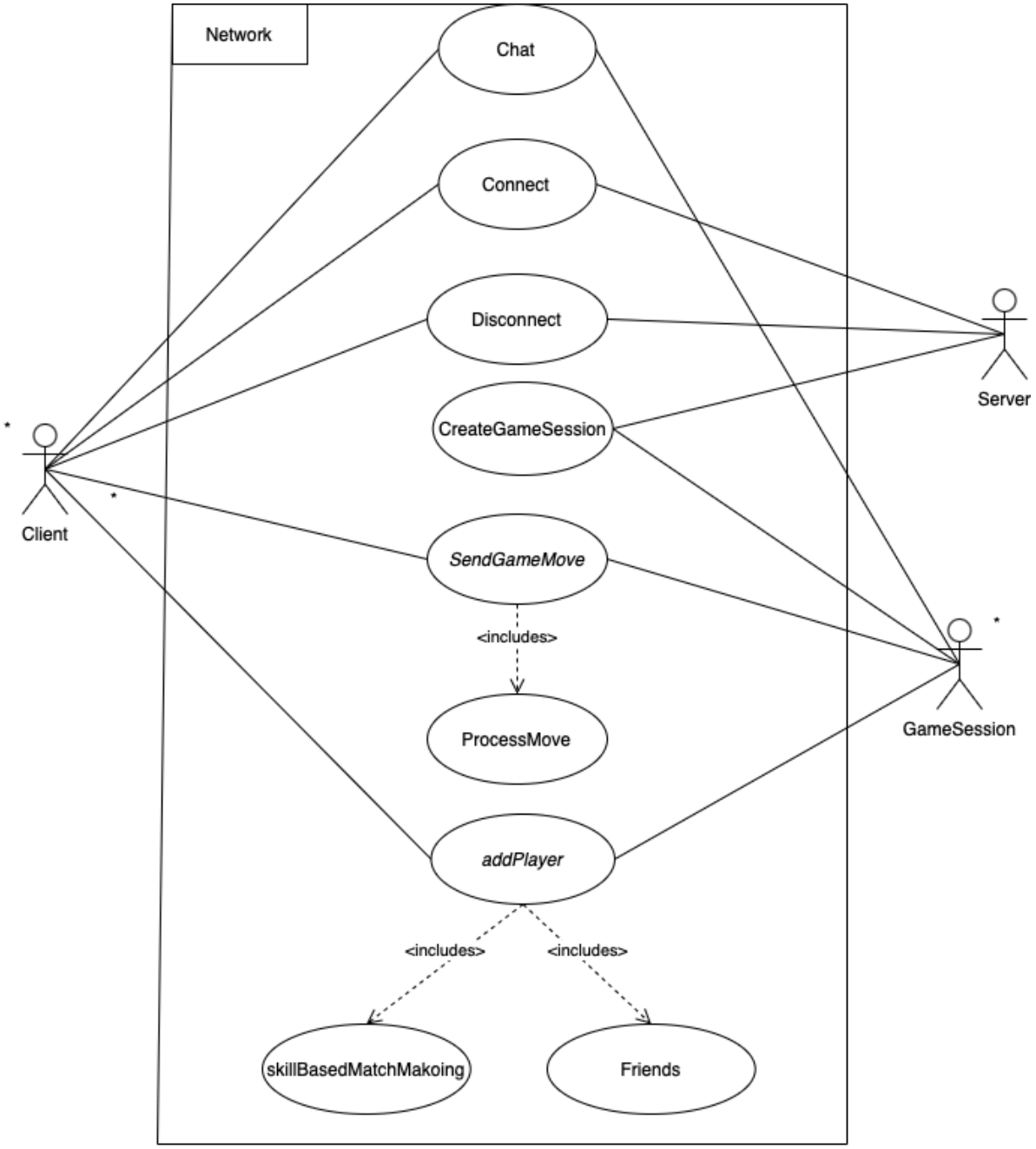
Use Case Diagram (Networking)



Use Case Descriptions (Networking)

Use Case: User connects to game server

Primary actor: Player

Goal in context: Connect player to server to enable gameplay and online interactions.

Preconditions: Player has an internet connection, game (client) is installed and running, server is online and available

Trigger: Player enters the multiplayer portion of the gaming platform.

Scenario:

1. Player selects “connect” or somehow enters the multiplayer section.
2. Game client retrieves the address and connection settings
3. Game client attempts to establish a connection with the server
4. Player 1 gains access to online features

Exceptions:

1. Poor internet and connection is interrupted
2. Server unreachable. Player will be notified if the server is down.

Priority: Essential for online gameplaying.

When available: Third iteration

Frequency of use: High. Used anytime a player wishes to play against other players.

Channel to actor: OMG gaming platform

Secondary Actor: Server

Channels to secondary actors: Server API, network connection

Open issues: High player traffic on server.

Use Case: User authenticates with server

Primary actor: Player

Goal in context: Verify the player's identity and give them access to the game server

Preconditions: Player has an internet connection, player has a registered account

Trigger: Player must click the login button.

Scenario:

1. Player clicks the login button
2. Player is prompted for a username and password
3. Player enters a username and password
4. Game client sends the login request to the server
5. Server verifies credentials
6. The client receives the server's response and the player is either granted access to the account in question or is prompted to try again.

Exceptions:

1. Username does not exist in the database
2. Username exists in the database but the password is incorrect

Priority: Essential, must be implemented to ensure a secure gaming experience

When available: Third iteration

Frequency of use: Medium. Some players may choose to stay logged out. Once a player is logged in, they will not be asked to log in again unless they close the session and reopen it.

Channel to actor: Via OMG gaming platform (requires internet connection)

Secondary actor: Authentication server

Channels to secondary actors: Network connection

Open issues:

1. How many wrong password attempts should it take for a user to get locked out of their account?

Use Case: Create new game session

Primary actor: Player

Goal in context: Player connects to a specific game session after matchmaking or by direct invitation.

Preconditions: Player must be authenticated and connected to the game server, server is online

Trigger: Player clicks “Create New Game”

Scenario:

1. Player clicks “Create New Game”
2. Game client sends a request to the server to create a new game
3. Server verifies request and creates a new game
4. Server gives the game a unique session ID and stores it in a database
5. Server sends back session details to client and the game is displayed on the client’s screen

Exceptions:

1. Network issues. Client fails to communicate with server and prompts the user to retry.

Priority: High. Required for initiating multiplayer games.

When available: Third iteration

Frequency of use: High. Anytime a player wishes to create a game.

Channel to actor: Via OMG gaming platform

Secondary actor: Server, game logic system

Channels to secondary actors: Network connection, database for storing session details

Open issues:

1. Handling ownership of the game if the player who created it disconnects

Use Case: Player joins game

Primary actor: Player 1

Goal in context: Player connects to a specific game session after matchmaking or invitation.

Preconditions: Player has authenticated and connected to the server, a game has been created (for invitation case), the player has received an invitation or has been assigned an opponent by the computer.

Trigger: Player 1 accepts the invitation from player 2 or the matchmaking system assigns player 1 to a game with player 2.

Scenario:

1. Player 1 accepts an invitation from player 2 or the matchmaking system assigns player 1 to a session
2. The game client sends a join request to the server asking to join a game with a specific ID
3. The server validates the request by checking if the session ID exists
4. If validation is successful, player 1 is added to the game
5. Server updates the game state and the client is notified that player 1 had been added.
6. Player 1 can see on the GUI that they have been added to the game

Exceptions:

1. Session is full. Player is notified and prompted to join another session
2. Session not found (invalid session ID). Player is notified and prompted to try again
3. Network error. Player is notified

Priority: High. Required for multiplayer gaming.

When available: Third iteration

Frequency of use: Each time a player joins an online session.

Channel to actor: OMG gaming platform

Secondary actor: Server, game session, player 2

Channels to secondary actors: Network connection, API for session updates, database for session details

Open issues:

1. Handling reconnects if a player is disconnected mid-game

Use Case: Send and receive game moves

Primary actor: Player 1

Goal in context: Enable real-time gaming interaction between players online

Preconditions: Player 1 and player 2 are connected to the game server, game session is active, server is online.

Trigger: Player 1 makes a move on the gaming platform

Scenario:

1. Player 1 makes a move in the game
2. Game client sends the move to the server
3. Server calls the game's logic system
4. Server updates the game state if the move is valid
5. Server sends updated game state to player 2
6. Player 2's client receives the updated move and displays it

Exceptions:

1. Network lag / connection loss
2. Invalid move

Priority: Essential. Used anytime a player wants to play against another player

When available: Third iteration

Frequency of use: High. Used anytime a player wishes to play against other players.

Channel to actor: OMG gaming platform

Secondary Actor: Player 2's client, game logic system, server, database for storing moves

Channels to secondary actors: Network connection, OMG gaming platform, API

Use Case: In-Game chat

Primary actor: Player 1

Goal in context: Players send and receive chat messages within a game session

Preconditions: Both players are connected to the game server

Trigger: Player 1 types and sends a chat to player 2

Scenario:

1. Player 1 types a message in the chat box and sends it
2. Game client sends the message to the server
3. The server sends the message to player 2's client
4. The message appears on player 2's chat

Exceptions:

1. Network lag / connection loss

Priority: Medium. Not necessary to play a game, but standard in online gaming platforms.

When available: Third iteration

Frequency of use: Medium. Not all players will wish to use the chat function.

Channel to actor: OMG gaming platform

Secondary Actor: Player 2's client, server

Channels to secondary actors: Network connection, OMG gaming platform

Open issues:

1. Handling inappropriate messages

Use Case: Disconnect

Primary actor: Player

Goal in context: Break the connection between the player's game client and the server, either on purpose or due to external factors.

Preconditions: Player is connected to the server

Trigger: Player exits the multiplayer section.

Scenario:

5. Player exits the multiplayer section (either by logging out or clicking "single player") OR server detects inactivity.
6. Client sends a disconnect request to the server
7. The server acknowledges the request and removes the client from the server
8. Gaming client returns to main menu

Exceptions:

3. Lost connection with network

Priority: Medium. Important for proper online gaming management.

When available: Third iteration

Frequency of use: High. Used anytime a player ends an online gaming session

Channel to actor: OMG gaming platform

Secondary Actor: Server

Channels to secondary actors: Network connection

Open issues: Handling unexpected disconnects without data loss

Use Case: Game session completion

Primary actor: Players

Goal in context: Gaming session ends, results are recorded, and stats are updated.

Preconditions: Game session is active, one player has met victory conditions or game has ended in a draw.

Trigger: A player has won or the game has ended in a draw

Scenario:

1. The game logic system detects a win or a draw
2. The server finalizes the game state (stops accepting new moves)
3. Server updates rankings according to the result of the game
4. Server sends game completion update to player 1 and 2's clients (containing game stats)
5. Game over message and game stats displayed on the screens of player 1 and 2

Exceptions:

1. Player disconnects before completion
2. Server crashes before winner is declared

Priority: Essential because game must end.

When available: Third iteration

Frequency of use: High. Used anytime a game ends.

Channel to actor: OMG gaming platform

Secondary Actor: Server, leaderboard system

Channels to secondary actors: Network connection

Open issues: How should match history be saved?

Use Case: Connection error recovery

Primary actor: Game client

Goal in context: System attempts to recover from temporary connection issues to maintain game integrity

Preconditions: Game client is actively connected to the server

Trigger: Client detects a connection failure due to network loss, server issues or timeouts

Scenario:

1. Client detects connection failure
2. Client displays an update to the player such as “reconnecting” and a loading circle
3. Client attempts automatic reconnection
4. a) If the client succeeds in reconnecting:
    - The client requests the latest game state from the server
    - The server sends the current game state
    - Client resynchronizes with the server and gameplay resumes

   b) Otherwise:
    - Client displays a disconnection message with options such as retrying manually to connect or quitting the game and returning to the main menu
    - If the client manages to reconnect to the server somehow, the steps in 4.a) will be taken.

Exceptions:

1. Player reconnects too late. They are notified that the session expired and are redirected to the main menu

Priority: High. Connection errors are inevitable and need to be handled

When available: Third iteration

Frequency of use: Medium. Will not happen every game, only when a connection issue occurs.

Channel to actor: client/server communication

Secondary Actor: Server, disconnected player

Channels to secondary actors: OMG gaming platform, network status messages sent to client

Open issues: how many automatic reconnect attempts should the client do before displaying a disconnection message?

Use Case: Matchmaking request processing

Primary actor: Server

Goal in context: Server handles player request for matchmaking and facilitates connection once a match is found.

Preconditions: player is logged into the game, player has selected matchmaking mode

Trigger: Game client sends a matchmaking request to the server

Scenario:

1. Player selects matchmaking mode and clicks “find opponent”
2. Game client sends matchmaking request to the server
3. The matchmaking system searches for a suitable match according to skill level and game type
4. The server sends a match confirmation request to both players who have been matched
5. If both players accept the match:
    - The server assigns the players to a session
    - The server notifies both player’s clients to connect to the assigned game server
    - Both clients connect to the game server and the game begins

Exceptions:

1. 1 or both players reject the match. In this case, the matchmaking process begins again for both players.

Priority: Essential, must be implemented for multiplayer gaming.

When available: third iteration

Frequency of use: High. Used every time a player wishes to match with another player.

Channel to actor: game client to server communication

Secondary Actor: player, matchmaking system

Channels to secondary actors: server, player’s client

Open issues: How should the server balance fair matches vs. faster matchmaking?

Use Case: Spectator mode

Primary actor: Spectator

Goal in context: Non-playing user connects to observe an ongoing game session

Preconditions: Spectator has permission to view the match. Game must be public or spectator must somehow be invited to view.

Trigger: Spectator requests to spectate a game

Scenario:

1. Spectator requests to join a game as an observer
2. Spectator's game client sends a request to the server to observe the game
3. The server assigns the spectator to the requested session
4. The client established a connection to the server
5. The game server streams real-time updates to the spectator's client
6. The spectator disconnects when the game is over or when they choose to leave.

Exceptions:

1. Network issues during spectating

Priority: Non-essential to multiplayer gaming

When available: Third iteration

Frequency of use: Used as needed by spectators. Could be high depending on the importance of the match.

Channel to actor: Via OMG gaming platform (internet required)

Secondary Actor: Game server

Channels to secondary actors: Network connection

Open issues:

1. If a game is public and open for spectators, should players be allowed to block a certain spectator from joining?
2. Should spectators have their own chat?