NETWORKING

Planning and Design

- Create planning documents for NETWORKING branch
  - Outline tasks, assign responsibilities, and set rough completion dates.
  - DEADLINE: March 6, 2025

- Design use case diagrams
  - Create a draft, review, and finalize use case diagrams for matchmaking.
  - DEADLINE: March 6, 2025

- Design class structure diagrams
  - Create a general class structure diagram for matchmaking-related classes.
  - DEADLINE: March 6, 2025

- Create use case descriptions document
  - Identify and detail use cases for networking (e.g., client control, game session sync, etc.)
  - DEADLINE: March 6, 2025

- Review and submit documents for Iteration 1
  - Team review, proofread, and submit to D2L dropbox.
  - DEADLINE: March 6, 2025

Phase 1: Setup

- Duration: March 8, 2025 - March 12, 2025

- Goal: Establish rough skeleton code and finalize design for NETWORKING

- Task List:
  - Git Repository Setup:
    - Set up Git repository for NETWORKING branch.
    - Start Date: March 4, 2025
    - End Date: March 4, 2025
    - Estimated Time: 1 hour.

  - Finalize class structure design:
    - Adjust class diagram based on feedback.
    - Start Date: March 4, 2025
    - End Date: March 5, 2025
    - Estimated Time: 3 hours.

  - Stub networking logic:
    - Simulate client and server performance within a game session.

- Start Date: March 10,2025
- End Date: March 11, 2025
- Estimated Time: 3 hours.

- Plan GUI integration:
    - Create a sketch for the networking interface.
    - Start Date: March 11, 2025
    - End Date: March 12, 2025
    - Estimated Time: 3 hours.

Milestone: Setup complete by March 12, 2025.

Phase 2: Core Development

Duration: March 13, 2025 - March 25, 2025

Goal: Implement core systems for NETWORKING.

Task List:
- Implement connection logic:
    - Develop logic to handle player connection and disconnection (e.g., ELO)
    - Start Date: March 13, 2025
    - End Date: March 16, 2025

<u>**SARA**</u>

Based on the project description and file structure, I'll explain what the networking component does and suggest methods for the networking files.

What Networking Does in This Project

- *Move sent from the client to the server → server validates the move and updates game session state → updated game state is sent to both players.*

The networking component of your OMG (Online Multiplayer board Game) platform

handles:

• Client-server communication for multiplayer games

• Real-time game state synchronization between players

• Handling player connections and disconnections

• Managing game sessions

• Transmitting game moves and chat messages

• Supporting the matchmaking system's online functionality

According to the project requirements, you don't need to actually implement the online functionality, but need to provide stubs where the server communication logic will reside.

Networking Component in the OMG Platform

The networking component is a crucial part of your Online Multiplayer board Game (OMG) platform, serving as the communication backbone between players. Here's a more detailed explanation:

1. **Client-Server Communication**: Establishes and maintains connections between players (clients) and the central game server.
2. **Game State Synchronization**: Ensures all players see the same game state by transmitting moves, updates, and events in real-time.
3. **Session Management**: Creates, maintains, and terminates game sessions as players join and leave games.
4. **User Authentication Integration**: Works with the authentication module to verify user credentials during login.
5. **Matchmaking Support**: Facilitates the matchmaking process by connecting matched players to the same game session.
6. **Chat Functionality**: Transmits chat messages between players during gameplay.
7. **Data Persistence Stubs**: Provides connection points to the external database team's work for saving game results and player statistics.
8. **Error Handling**: Manages connection issues, timeouts, and unexpected disconnections to maintain gameplay integrity.
9. **Game Move Validation**: Works with game logic to ensure moves are valid before broadcasting to other players.
10. **Multiplayer Coordination**: Handles turn-based gameplay coordination, ensuring players act in the correct order.

<u>VIHANDU</u>

2. Methods and Attributes for Networking Files

Server.java

Attributes:

- private int port - Port number for the server
- private List<GameSession> activeSessions - List of active game sessions
- private boolean isRunning - Server status flag
- private UserDatabaseStub userDatabase - Reference to user database

Methods:

- public void start(int port) → Starts the server on specified port
- public void stop() → Stops the server
- public GameSession createGameSession(String gameType, UserProfile player1, UserProfile player2) → Creates a new game session
- public boolean authenticateUser(String username, String password) → Authenticates a user • public void broadcastMessage(String message) → Broadcasts message to all clients • public List<GameSession> getActiveSessions() → Returns list of active sessions • public void registerMatchmaker(Matchmaker matchmaker) → Registers matchmaking system • private void handleClientConnection(Client client) → Handles new client connections
- public void updateLeaderboard(UserProfile player, int points) → Updates leaderboard after game

Client.java

Attributes:

- private String serverAddress - Server IP address
- private int serverPort - Server port number
- private UserProfile userProfile - Current user profile
- private GameSession currentSession - Current game session
- private boolean isConnected - Connection status flag

Methods:

- public boolean connect(String serverAddress, int port) → Connects to server • public void disconnect() → Disconnects from server
- public boolean login(String username, String password) → Logs in user • public void sendGameMove(String moveData) → Sends game move to server • public void joinGameSession(int sessionId) → Joins an existing game session • public

void leaveGameSession() → Leaves current game session

• public void sendChatMessage(String message) → Sends chat message • public

void requestMatchmaking(String gameType) → Requests matchmaking • public

boolean isConnected() → Returns connection status

• private void handleServerMessage(String message) → Handles incoming server messages •

public void updateUserProfile() → Updates user profile data from server **GameSession.java**

Attributes:

• private int sessionId - Unique session identifier

• private String gameType - Type of game being played

• private List<Client> players - Players in the session

• private Object gameState - Current game state

• private boolean isActive - Session status

• private ChatManager chatManager - Chat manager for this session

Methods:

• public GameSession(int sessionId, String gameType) → Constructor

• public boolean addPlayer(Client player) → Adds player to session

• public boolean removePlayer(Client player) → Removes player from session • public

void broadcastGameState() → Broadcasts game state to all players • public void

processMove(Client player, String moveData) → Processes a player's move • public

boolean isSessionFull() → Checks if session has maximum players • public void

endSession(Client winner) → Ends the session with a winner • public int getSessionId()

→ Returns session ID

• public String getGameType() → Returns game type

• public List<Client> getPlayers() → Returns players in session

• public void sendChatMessage(Client sender, String message) → Sends chat message to all players

• public void notifyPlayerDisconnect(Client player) → Notifies when a player disconnects

These methods would provide the necessary functionality for implementing the networking component of your multiplayer board game platform while using stubs where needed, as mentioned in the project description.

ELOISA

Use Case Descriptions for Networking

1. User Connection

- **Title**: User Connects to Game Server
- **Actors**: Player, Server
- **Description**: A user launches the game client and connects to the central game server

2. User Authentication

- **Title**: User Authenticates with Server
- **Actors**: Player, Server, Authentication System
- **Description**: Connected user provides credentials which are verified against the user database

3. Game Session Creation

- **Title**: Create New Game Session
- **Actors**: Server, Game Logic System
- **Description**: Server creates a new game session when players are matched or a custom game is requested

4. Player Joining Game

- **Title**: Player Joins Game Session
- **Actors**: Player, Server, Game Session
- **Description**: Player connects to a specific game session after matchmaking or by direct invitation

5. Game Move Transmission

- **Title**: Game Move Processing
- **Actors**: Player, Server, Game Logic System
- **Description**: Player makes a move which is transmitted to the server, validated, and broadcast to other players

6. Real-time State Synchronization

- **Title**: Game State Update
- **Actors**: Server, All Players in Session

- **Description**: Server ensures all players have synchronized game state after each game event

7. Chat Communication

- **Title**: In-Game Chat
- **Actors**: Players in Session, Server
- **Description**: Players send and receive chat messages within a game session

8. Player Disconnection Handling

- **Title**: Handle Player Disconnection
- **Actors**: Server, Remaining Players, Game Logic
- **Description**: Server detects player disconnection and manages the game state accordingly

9. Game Completion

- **Title**: Game Session Completion
- **Actors**: Server, Players, Leaderboard System
- **Description**: Game session ends, results are recorded, and statistics are updated

10. Server-Client Error Recovery

- **Title**: Connection Error Recovery
- **Actors**: Server, Disconnected Player
- **Description**: System attempts to recover from temporary connection issues to maintain game integrity

11. Matchmaking Integration

- **Title**: Matchmaking Request Processing
- **Actors**: Player, Server, Matchmaking System
- **Description**: Server handles player request for matchmaking and facilitates connection once a match is found

12. Game Spectating

- **Title**: Spectator Mode
- **Actors**: Spectator, Server, Active Game Session
- **Description**: Non-playing user connects to observe an ongoing game session

These use cases provide a comprehensive overview of how the networking component would function in your multiplayer game platform, covering the key interactions between players, the server, and various system components.