Networking Team - Project Iteration 1 Agenda

Project: Online Multiplayer Board Game Platform (OMG)

To successfully integrate networking into the system, our first iteration will focus on designing the necessary classes and core functionality, drafting the required use case descriptions and diagrams, and structuring our class relationships in a way that ensures efficient implementation in future iterations.

Classes and Components Needed for Networking Implementation

- **ClientHandler** – Manages individual player connections and handles incoming/outgoing messages.
    - Accepts new connections from players.
    - Reads and writes messages to and from the server.
    - Keeps track of active players and their session states.
- **NetworkManager** – Oversees all network communications between clients and the game server.
    - Acts as a high-level controller for networking logic.
    - Routes messages between ClientHandler, GameSessionManager, and other components.
    - Manages protocol-specific details (e.g., WebSockets or TCP).
- **GameSessionManager** – Maintains active game sessions, syncing state across players.
    - Keeps track of ongoing matches and active players per session.
    - Ensures that game actions are properly processed and broadcasted to all participants.
    - Handles reconnections by restoring the player's session state.
- **MessageParser** – Handles serialization/deserialization of messages.
    - Encodes outgoing data into **JSON** or another format.
    - Decodes incoming player actions and system messages.
    - Ensures messages follow a predefined schema.

- **ServerController –** Central control for the server-side logic.
    - Accepts and processes incoming requests from NetworkManager.
    - Delegates game-related requests to GameSessionManager.
    - Maintains a global list of active sessions and players.
- **GameStateSynchronizer –** Ensures that all clients maintain a consistent game state.
    - Sends periodic updates to clients about the game state.
    - Handles state rollback in case of inconsistencies.
- **AuthHandler (Interface with Authentication Team) –** Processes login/logout requests and maintains user sessions.
    - Receives authentication requests from the client.
    - Interfaces with the authentication team's system for validation.
    - Manages session tokens for persistent player authentication.

Database Components

- DatabaseConnector - Handles the connection between the application and the PostgreSQL/MySQL database.
    - Establishes a connection using JDBC.
    - Executes queries for authentication, player records, and match history.
- PlayerManager - Manages player-related database interactions.
    - Registers new players.
    - Verifies login credentials.
    - Updates player stats after matches.
- MatchHistoryManager - Stores and retrieves post-match records.
    - Saves completed games.
    - Retrieves leaderboard rankings.
    - Provides player match history.

Use Case Descriptions & Diagrams

1. **(Cristian) Player Joins a Game**
    - When a player selects a game, they send a request to join a session.
    - The system verifies if an existing session is available. If so, the player is added; otherwise, a new session is created.
    - The use case diagram will illustrate the interaction between the **player, server, and game session manager**.
2. **(Josiah)Game State Update**

- Each time a player takes an action (e.g., moving a piece in chess), the action is sent to the server.
- The server updates the game state and relays the new state to all connected players.
- This will be visualized in a use case diagram showing the flow of game state updates between the player, game session manager, and game state synchronizer.

3. **(Cristian) Player Disconnection & Reconnection**
   - If a player disconnects, the server temporarily retains their game state.
   - If the player reconnects within a set timeframe, they rejoin the session with their last known state.
   - The use case diagram will show interactions between the player, client handler, and game session manager.
4. **(Josiah)Chat Message Handling (If Implemented)**
   - A player sends a message in a game session, which is routed through the server and broadcasted to other session participants.
   - This use case diagram will depict the flow of chat messages between the client, server, and session members.
5. **(Cristian) Authentication & Session Handling**
   - When a player logs in, their credentials are verified by the authentication system.
   - Upon success, a session token is issued and stored.
   - This use case diagram will highlight the interaction between the player, authentication system, and network layer.
6. **(Josiah)Player Creates Account & Credentials Stored on Server**
   - The player launches the game and has access to the registration screen.
   - The game server is online and reachable
   - The database is available to store player credentials
7. **(Cristian) Turn Timeout Handling**
   - Server starts a countdown timer when a player's turn begins.
   - If the player makes a move within the allotted time period, the move is processed normally.
   - If the timer expires
     i. Assigns a default move?
     ii. Skips the player's turn?

Class Structure and Relationships

- **(Martin) NetworkManager and ClientHandler** will work together to establish and maintain player connections. The ClientHandler will manage communication with individual clients, while NetworkManager will route data across the system.
- **(Colby) GameSessionManager and GameStateSynchronizer** will coordinate game state updates. The session manager will ensure each session is properly tracked, and the synchronizer will handle broadcasting updates to players.
- **(Martin) ServerController** will serve as the main coordinator, managing incoming requests and delegating tasks to the appropriate handlers.
- **(Martin) MessageParser** will act as an interface that allows for flexibility in message encoding (e.g., JSON or binary formats). This design will enable future adaptability without changing core networking logic.
- **(Colby) AuthHandler** will interface with the authentication team's implementation, ensuring seamless authentication and session management.
- **(Colby) DatabaseConnector** established the connection to the database.
- **PlayerManager** retrieves player profiles and handles authentication.
- **MatchHistoryManager** records game results and retrieves leaderboard data.

Networking Implementation Strategy

- **Communication Model**
    - We will use WebSockets to enable real-time bidirectional communication between clients and the server.
    - As a fallback, TCP Sockets may be implemented for structured request-response interactions.
- **Data Serialization**
    - Messages will initially be serialized in JSON for readability and debugging purposes.
    - We may later explore binary serialization to optimize performance.
- **Game State Management**
    - Player actions will be sent to GameSessionManager, which will process and validate them.
    - The updated game state will be broadcasted to all players via GameStateSynchronizer.
    - A rollback mechanism will be designed to handle desynchronization issues.
- **Session Handling & Error Recovery**

    - Disconnections will be detected by ClientHandler, and players will be allowed to reconnect within a time window before being removed from a session.
    - Fallback measures will be in place to prevent lost game progress.
- **Security Considerations**
    - Incoming messages will be validated to prevent unauthorized actions.
    - Basic encryption may be used to ensure secure transmission of authentication credentials.
    - Passwords will be hashed before entry into the database.
    - Session tokens in the database will be used to track logged-in users.
    - Incoming messages should be validated to prevent SQL injection.
- **Database Integration**
    - AuthHandler will validate players against the database (Players table?).
    - GameSessionManager will record match results in a match history table.
    - Leaderboards will be queried dynamically from the leaderboard table/view.

Action Items

- **Class Implementations**
    - (name) will implement the ClientHandler class.
    - (name) will develop the NetworkManager class.
    - (name) will work on GameSessionManager.
    - (name) will design the MessageParser interface.
    - (name) will create the base structure for ServerController.
- **Diagrams & Documentation**
    - Christian & Josiah will draft the use case descriptions and diagrams.
    - Colby & Martin will design the class structure diagram.
- **Initial Implementation Steps**
    - The first networking prototype will establish basic WebSocket/TCP connections.
    - A message structure will be defined and tested using MessageParser.
    - The server will be set up to accept and process client connections.