Aidan Doruska, Kyle Balk, Braden Miller

Multiplayer Connect 4 Project - Developer's Guide

Overview:

This Connect 4 implementation uses a client-server architecture with Java Sockets and multithreading. Each client represents a player and manages its own UI, while the server manages the turn-taking, broadcasting, and chat communication for up to 4 players.

Compile and Run:

- Compile: javac *.java
- Run Server: java Server
- Run Client: java ClientTest

Key Classes:

- Server: Manages the connections of adding clients to the game.
- Player: Each thread handles the client interaction and runs the entire game loop logic. Implements Runnable().
- Client: Manages the connections and inputs from the server.
- LobbyController: Lobby GUI, displays the lobby with live read of number of players and extra functionality for the first player (leader) to manage the grid dimensions and to start the game.
- GameController: Displays the connect 4 grid, handles user clicks to place chips using animations, handles the animations, and implements the turn logic.
- PostGameController: Manages the leaderboard and chat UI.
- BackgroundController: Displays a random background image for the lobby.

Server/Client Communication:

Key Communication Logic: The core of the game's communication protocol is built around string-based messages passed between the clients and server. The client processConnection() method and the server's Player.run() method contain the majority of the logic to decode these message strings. Below are the main message handlers:

Client-Side Messages (Client.java)

- "You are Player X" → sets the player's number
- "Your turn!" → activates the player's turn
- "Turn|<playerNumber>" → updates the client to reflect whose turn it is

- “GAMEOVER” → Transitions the client UI to post-game mode
- “CHAT|<message>|<playerNumber>” → Displays a chat message in the post-game chat
- “LEADER” → indicates the given client has been given lobby leader rights
- “CONNECTED|<playerCount>” → Updates the number of currently connected players
- “PLAYERWON|<playerNumber>” → Announces a given player’s win and updates the rankings
- “NOTIFY|<message>” → Displays a pop-up notification to the client with the given message
- “START|<rows>|<cols>|<win>” → Initializes the gameboard and win condition
- “TERMINATE” → Signals the end of the session

Server-Side Messages (Player.java)

- “START|<boardSize>|<win>” → (this comes only from the leader) Broadcasts aboard setup information to all of the players and the win condition
- “CHAT|<message>|<playerNumber>” → Broadcasts a post-game chat message to all of the players
- “WON” → Broadcasts the player’s win to all of the clients, triggers game over if 1 player remains
- “<col>” (integer) → Represents the column where the player made a move

Turn Synchronization: The server uses **ReentrantLock turnLock** and **Condition turnCondition** to manage the turn-taking and to ensure no interference. **playerNumber** ensures that only the correct player is making their turn and the **isDone** flag tracks players that have finished or quit and should be skipped.

Threading Model: Each client runs their own GUI thread. The server launches one thread per Player (each implementing **Runnable**). All threads are managed by the **ExecutorService**.

Game Flow:

1. First player to join the lobby becomes the leader.
2. Leader waits for at least 1 other player to join the lobby.
3. Leader can select the board dimensions, the number of chips in a row required for a win, and when to start the game.
4. The turn orders are enforced using ReentrantLock and Condition as well as variables tracking the currentPlayerTurn and nextPlayer.
5. Each move is broadcasted to all players to update GUI’s respectively.
6. When a player wins, their win is sent to the server.
7. Game continues until 1 or 0 players remain.
8. Final page containing a leaderboard and chatroom are shown to all.

Aidan Doruska, Kyle Balk, Braden Miller