
# SOFE 4790U: Distributed Systems (Fall 2023)

## Assignment #2

**Honour code:**

By submitting this assignment, I affirm this is my own work, and I have not asked any of my fellow students or others for their source code or solutions to complete this assignment, and I have not offered my source code or solutions for this assignment to any of my fellow students.

**Name:** OluwaJomiloju Ijose

**Banner ID#:** 100819367

## 1. Application Idea

My application is a distributed multiplayer Tic-Tac-Toe game that lets players play against each other or an AI. It supports both a text-based client and a GUI client built using JavaFX. The game runs on an RMI-based distributed system where the server manages game sessions, game states, and communication between clients.

## 2. Describe the Core Functionalities (5 Unique Functions)

- Create or Join a Game

Players can create a new game with custom settings (board size, win condition, AI or player opponent) or join an existing game. The server matches players based on their game settings.

```java
public class GameClient {
    public static void main(String[] args) {
        try {
            GameInterface server = (GameInterface) Naming.lookup( name: "rmi://localhost/GameServer");
            Scanner scanner = new Scanner(System.in);

            System.out.print("Enter your name: ");
            String playerName = scanner.nextLine();

            System.out.println("Do you want to play against another player or AI? (Type 'player' or 'AI'):");
            String opponentChoice = scanner.nextLine();

            int boardSize = 3;
            int winCondition = 3;
            System.out.print("Enter board size (default 3): ");
            String boardSizeInput = scanner.nextLine();
            if (!boardSizeInput.isEmpty()) {
                boardSize = Integer.parseInt(boardSizeInput);
            }
            System.out.print("Enter win condition (default 3): ");
            String winConditionInput = scanner.nextLine();
            if (!winConditionInput.isEmpty()) {
                winCondition = Integer.parseInt(winConditionInput);
            }

            String gameId;
            if (opponentChoice.equalsIgnoreCase( anotherString: "AI")) {
                gameId = server.createGame(playerName, boardSize, winCondition,  playWithAI: true);
                System.out.println("Starting a game against AI.");
            } else {
                gameId = server.findGame(playerName, boardSize, winCondition);
                System.out.println("Joined game: " + gameId);
            }
```

```
public synchronized String createGame(String playerName, int boardSize, int winCondition, boolean playWithAI) throws RemoteException {
    String gameId = (playWithAI ? "AIGame" : "Game") + (gameSessions.size() + 1);
    GameSession newSession = new GameSession(gameId, playerName, boardSize, winCondition);
    if (playWithAI) {
        newSession.player2 = "AI";
        newSession.isAI = true;
        newSession.currentTurn = newSession.player1;
    }
    gameSessions.put(gameId, newSession);
    return gameId;
}
```

---

- Real-Time Gameplay Management

The server maintains the state of the game and enforces turn-based gameplay. Players can make moves, and the game updates the board in real-time. This feature ensures fair play and consistency across clients.

```
public synchronized String makeMove(String gameId, String playerName, int row, int col) throws RemoteException {
    GameSession session = gameSessions.get(gameId);
    if (session == null) {
        return "Invalid game ID!";
    }
    if (session.winner != null) {
        return "The game has already ended!";
    }
    if (!playerName.equals(session.currentTurn)) {
        return "It's not your turn!";
    }
    if (row < 0 || row >= session.boardSize || col < 0 || col >= session.boardSize || !session.board[row][col].isEmpty()) {
        return "Invalid move!";
    }
    session.board[row][col] = playerName.equals(session.player1) ? "X" : "O";
    if (checkWinner(session, row, col)) {
        session.winner = session.currentTurn;
        // Record the game result
        recordGameResult(session.winner, won: true);
        String loser = session.winner.equals(session.player1) ? session.player2 : session.player1;
        if (loser != null && !loser.equals("AI")) {
            recordGameResult(loser, won: false);
        }
    } else {
        if (session.isBoardFull()) {
            session.winner = "Draw";
        } else {
            session.currentTurn = session.currentTurn.equals(session.player1) ? session.player2 : session.player1;
        }
    }

    // If playing against AI and it's AI's turn
    if (session.isAI && session.currentTurn.equals("AI") && session.winner == null) {
        aiMakeMove(session);
    }
    return "Move accepted!";
}
```

```java
public synchronized String getGameState(String gameId) throws RemoteException {
    GameSession session = gameSessions.get(gameId);
    if (session == null) {
        return "Invalid game ID!";
    }
    StringBuilder state = new StringBuilder();
    for (int row = 0; row < session.boardSize; row++) {
        for (int col = 0; col < session.boardSize; col++) {
            String cell = session.board[row][col];
            state.append(cell.isEmpty() ? "-" : cell).append(" ");
        }
        state.append("\n");
    }
    return state.toString();
}
```

---

- Chat Functionality

Players can send and receive messages during the game. The server handles the message delivery between players, ensuring seamless communication.

```java
@Override  2 usages
public synchronized void sendMessage(String gameId, String playerName, String message) throws RemoteException {
    GameSession session = gameSessions.get(gameId);
    if (session == null) return;

    String opponent = playerName.equals(session.player1) ? session.player2 : session.player1;
    if (opponent != null && !opponent.equals("AI")) {
        session.messageQueues.computeIfAbsent(opponent, k -> new ArrayList<>()).add(playerName + ": " + message);
    }
}

@Override  2 usages
public synchronized List<String> receiveMessages(String gameId, String playerName) throws RemoteException {
    GameSession session = gameSessions.get(gameId);
    if (session == null) return new ArrayList<>();

    List<String> messages = session.messageQueues.getOrDefault(playerName, new ArrayList<>());
    session.messageQueues.put(playerName, new ArrayList<>()); // Clear messages after retrieval
    return messages;
}
```

---

- Leaderboard Tracking

The application records player statistics (wins and losses) and displays them in the leaderboard. This functionality provides competitive tracking and encourages players to improve their performance.

```
@Override  3 usages
public synchronized void recordGameResult(String playerName, boolean won) throws RemoteException {
    if (playerName.equals("AI")) return;
    int[] stats = playerStats.getOrDefault(playerName, new int[]{0, 0});
    if (won) {
        stats[0]++; // Increment wins
    } else {
        stats[1]++; // Increment losses
    }
    playerStats.put(playerName, stats);
}

@Override  1 usage
public synchronized Map<String, int[]> getLeaderboard() throws RemoteException {
    return playerStats;
}
```

---

- Variable Game Settings

Players can select different board sizes and winning conditions, allowing for more challenging and varied gameplay. The game logic adapts dynamically to the chosen settings.

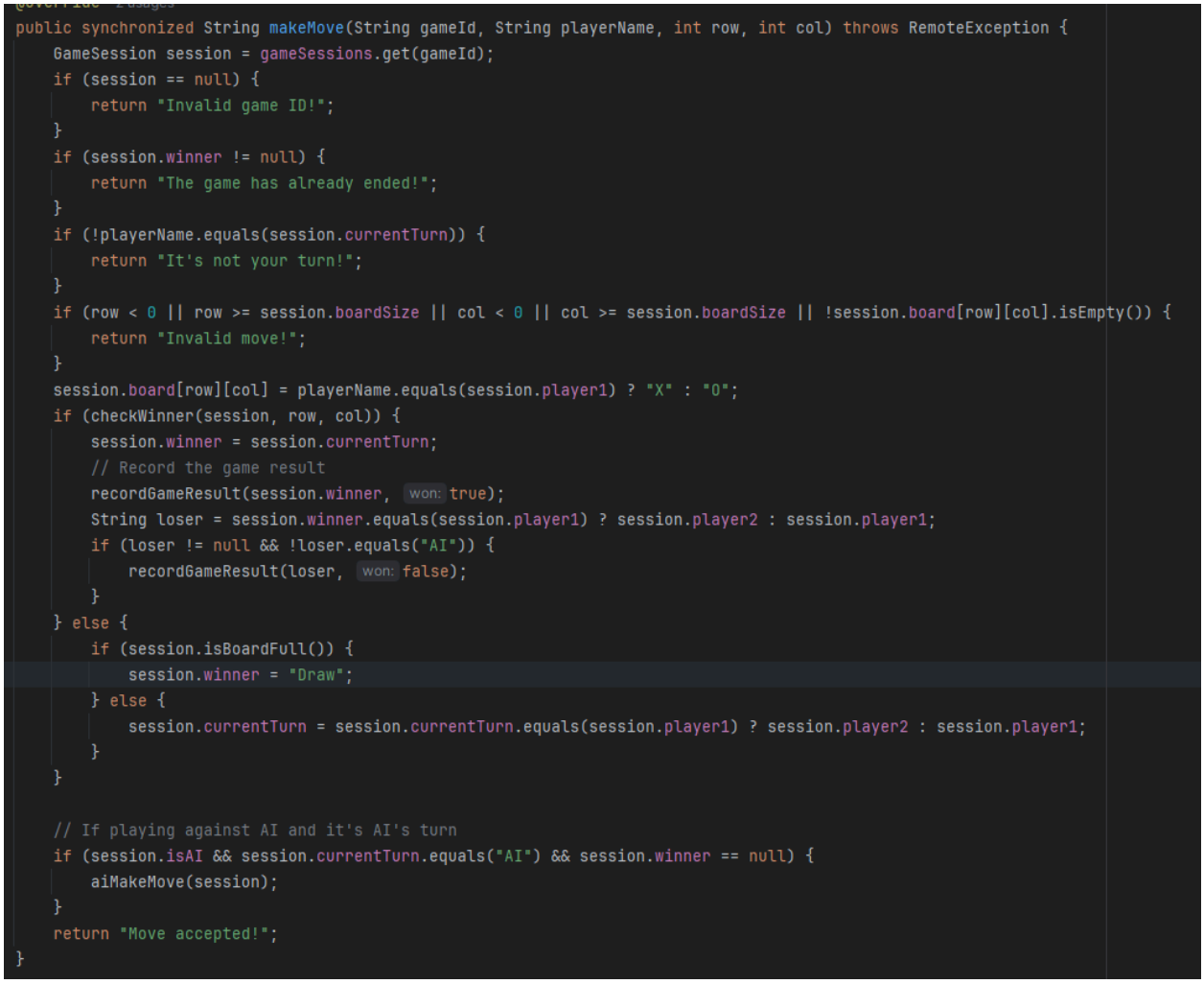
```
@Override  1 usage
public synchronized int getBoardSize(String gameId) throws RemoteException {
    GameSession session = gameSessions.get(gameId);
    return session != null ? session.boardSize : -1;
}
```

```
int boardSize = 3;
int winCondition = 3;
System.out.print("Enter board size (default 3): ");
String boardSizeInput = scanner.nextLine();
if (!boardSizeInput.isEmpty()) {
    boardSize = Integer.parseInt(boardSizeInput);
}
```

## 3. Challenges and Solutions

Challenge 1: Implementing Customizable Game Logic

- Problem: Supporting variable board sizes and win conditions required significant changes to the core game logic, including handling different win conditions in rows, columns, and diagonals.
- Solution: The logic was generalized using loops to check all possible winning scenarios dynamically based on the chosen board size and win condition.



```
public synchronized String makeMove(String gameId, String playerName, int row, int col) throws RemoteException {
    GameSession session = gameSessions.get(gameId);
    if (session == null) {
        return "Invalid game ID!";
    }
    if (session.winner != null) {
        return "The game has already ended!";
    }
    if (!playerName.equals(session.currentTurn)) {
        return "It's not your turn!";
    }
    if (row < 0 || row >= session.boardSize || col < 0 || col >= session.boardSize || !session.board[row][col].isEmpty()) {
        return "Invalid move!";
    }
    session.board[row][col] = playerName.equals(session.player1) ? "X" : "O";
    if (checkWinner(session, row, col)) {
        session.winner = session.currentTurn;
        // Record the game result
        recordGameResult(session.winner, won: true);
        String loser = session.winner.equals(session.player1) ? session.player2 : session.player1;
        if (loser != null && !loser.equals("AI")) {
            recordGameResult(loser, won: false);
        }
    } else {
        if (session.isBoardFull()) {
            session.winner = "Draw";
        } else {
            session.currentTurn = session.currentTurn.equals(session.player1) ? session.player2 : session.player1;
        }
    }

    // If playing against AI and it's AI's turn
    if (session.isAI && session.currentTurn.equals("AI") && session.winner == null) {
        aiMakeMove(session);
    }
    return "Move accepted!";
}
```

```java
private void aiMakeMove(GameSession session) {  1 usage
    Random rand = new Random();
    int row, col;
    do {
        row = rand.nextInt(session.boardSize);
        col = rand.nextInt(session.boardSize);
    } while (!session.board[row][col].isEmpty());
    session.board[row][col] = "O"; // AI uses 'O'

    // Check if AI wins
    if (checkWinner(session, row, col)) {
        session.winner = "AI";
        try {
            recordGameResult(session.player1, won: false);
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    } else if (session.isBoardFull()) {
        session.winner = "Draw";
    } else {
        session.currentTurn = session.player1;
    }
}
```

```java
    private boolean checkWinner(GameSession session, int lastRow, int lastCol) {  2 usages
        String[][] board = session.board;
        String symbol = board[lastRow][lastCol];
        int boardSize = session.boardSize;
        int winCondition = session.winCondition;

        // Check row
        int count = 0;
        for (int col = 0; col < boardSize; col++) {
            count = board[lastRow][col].equals(symbol) ? count + 1 : 0;
            if (count == winCondition) return true;
        }

        // Check column
        count = 0;
        for (int row = 0; row < boardSize; row++) {
            count = board[row][lastCol].equals(symbol) ? count + 1 : 0;
            if (count == winCondition) return true;
        }

        // Check diagonal (\)
        count = 0;
        int startRow = lastRow - Math.min(lastRow, lastCol);
        int startCol = lastCol - Math.min(lastRow, lastCol);
        while (startRow < boardSize && startCol < boardSize) {
            count = board[startRow][startCol].equals(symbol) ? count + 1 : 0;
            if (count == winCondition) return true;
            startRow++;
            startCol++;
        }

        // Check anti-diagonal (/)
        count = 0;
        startRow = lastRow + Math.min(boardSize - 1 - lastRow, lastCol);
        startCol = lastCol - Math.min(boardSize - 1 - lastRow, lastCol);
        while (startRow >= 0 && startCol < boardSize) {
            count = board[startRow][startCol].equals(symbol) ? count + 1 : 0;
            if (count == winCondition) return true;
            startRow--;
            startCol++;
        }

        return false;
    }

    public static void main(String[] args) {
        try {
            Naming.rebind( name: "GameServer", new GameServer());
            System.out.println("Game Server is running...");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Challenge 2: Cross-Client Compatibility

- Problem: Ensuring that both text-based and GUI clients could seamlessly interact with the server and each other required consistent data handling and synchronization.
- Solution: The server-side code was designed to abstract game state updates and communication.

```
public synchronized String createGame(String playerName, int boardSize, int winCondition, boolean playWithAI) throws RemoteException {
    String gameId = (playWithAI ? "AIGame" : "Game") + (gameSessions.size() + 1);
    GameSession newSession = new GameSession(gameId, playerName, boardSize, winCondition);
    if (playWithAI) {
        newSession.player2 = "AI";
        newSession.isAI = true;
        newSession.currentTurn = newSession.player1;
    }
    gameSessions.put(gameId, newSession);
    return gameId;
}

@Override 2 usages
public synchronized String findGame(String playerName, int boardSize, int winCondition) throws RemoteException {
    // Find a game waiting for a second player with matching settings
    for (GameSession session : gameSessions.values()) {
        if (session.player2 == null && !session.isAI && session.boardSize == boardSize && session.winCondition == winCondition) {
            session.player2 = playerName;
            session.currentTurn = session.player1; // Start the game
            return session.gameId;
        }
    }
    // Create a new game
    return createGame(playerName, boardSize, winCondition, playWithAI: false);
}
```

---

Challenge 3: JavaFX Integration

- Problem: Configuring JavaFX with the newer JDK versions and ensuring compatibility during runtime was challenging.
- Solution: The JavaFX SDK was downloaded and configured with appropriate module paths during compilation and execution.

---

Challenge 4: AI Opponent Logic

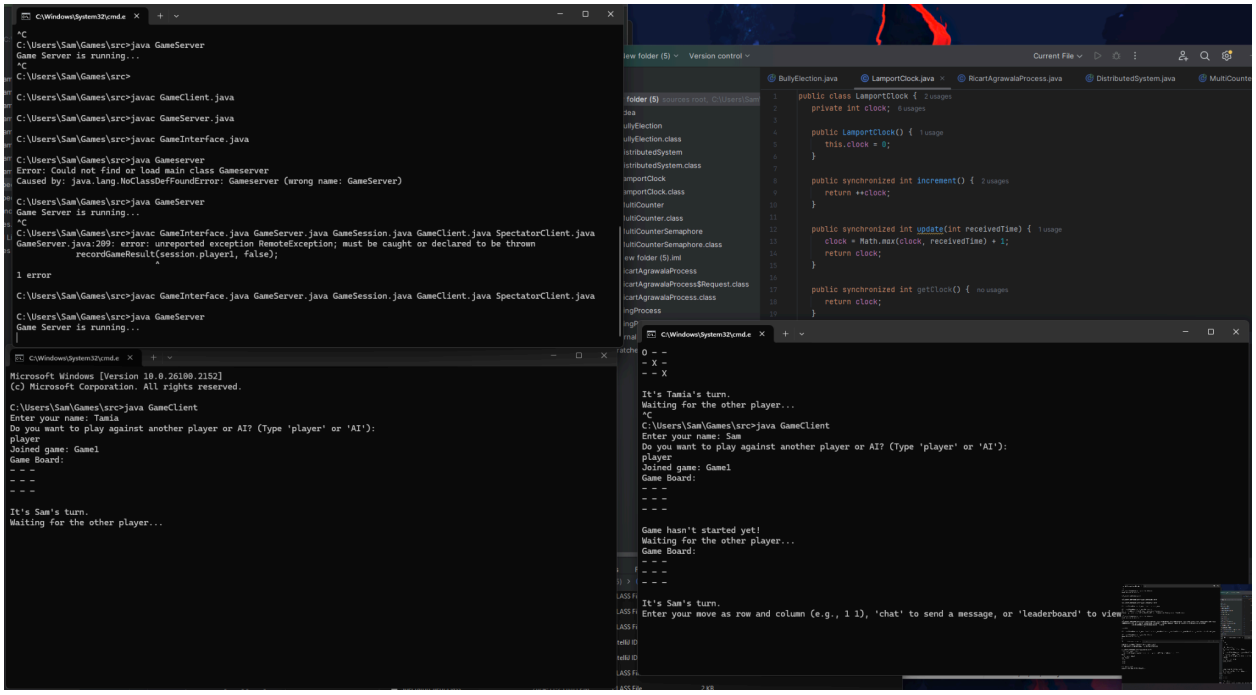
- Problem: Designing an AI that could make valid moves while adhering to game rules was difficult at first.
- Solution: A simple random-move AI was implemented, ensuring valid moves were selected, and it could adapt to different board sizes.

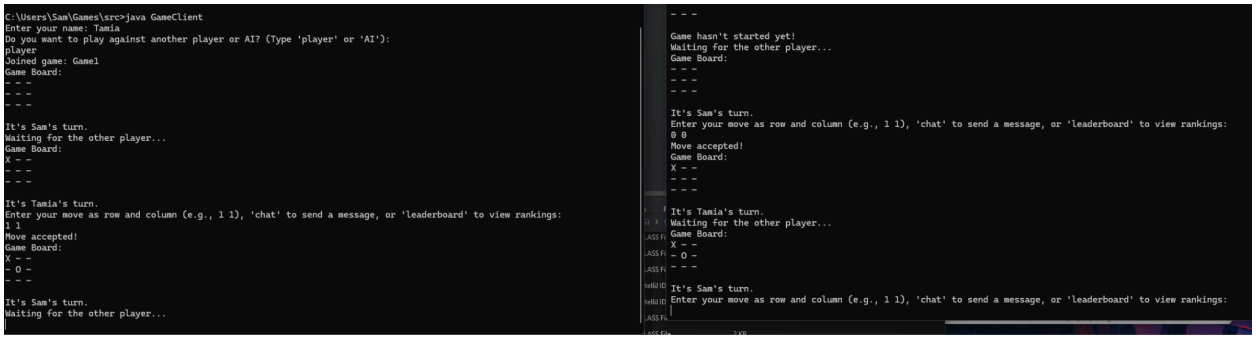
```java
private void aiMakeMove(GameSession session) {  1 usage
    Random rand = new Random();
    int row, col;
    do {
        row = rand.nextInt(session.boardSize);
        col = rand.nextInt(session.boardSize);
    } while (!session.board[row][col].isEmpty());
    session.board[row][col] = "O"; // AI uses 'O'

    // Check if AI wins
    if (checkWinner(session, row, col)) {
        session.winner = "AI";
        try {
            recordGameResult(session.player1, won: false);
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    } else if (session.isBoardFull()) {
        session.winner = "Draw";
    } else {
        session.currentTurn = session.player1;
    }
}
```

---

## 4. Testing

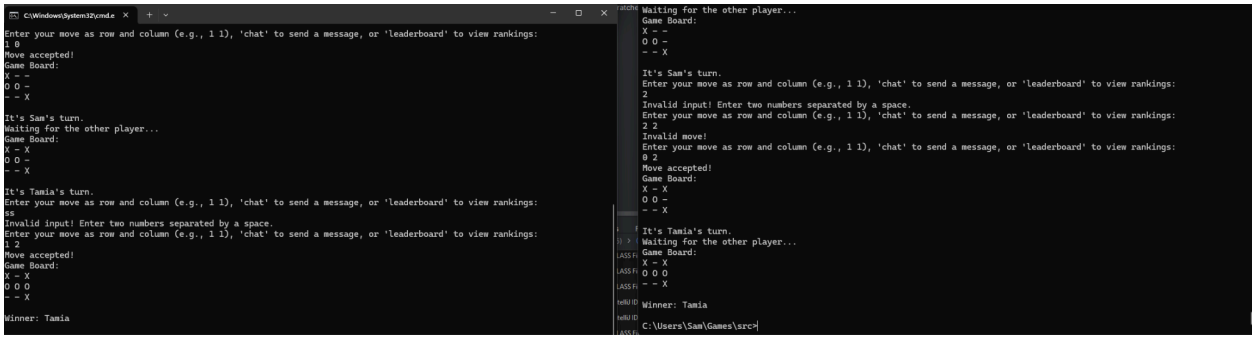
Gameplay (player):



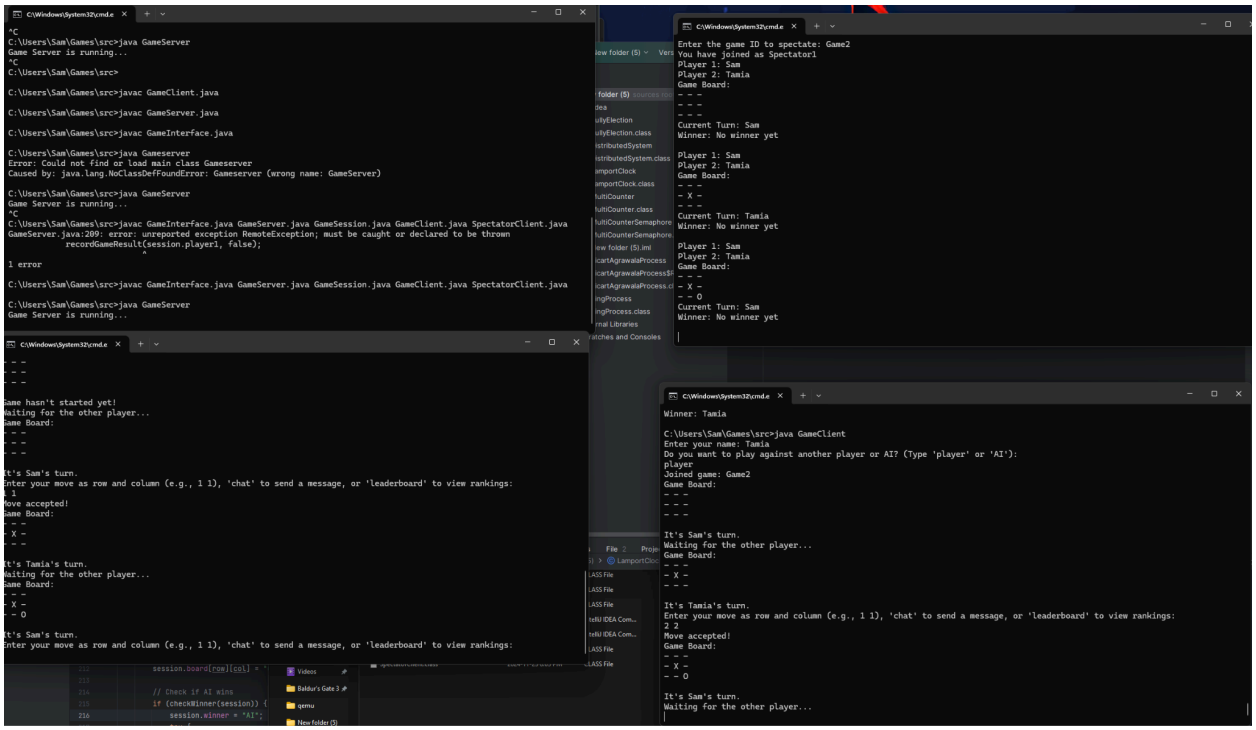




Invalid move:

Win Condition:



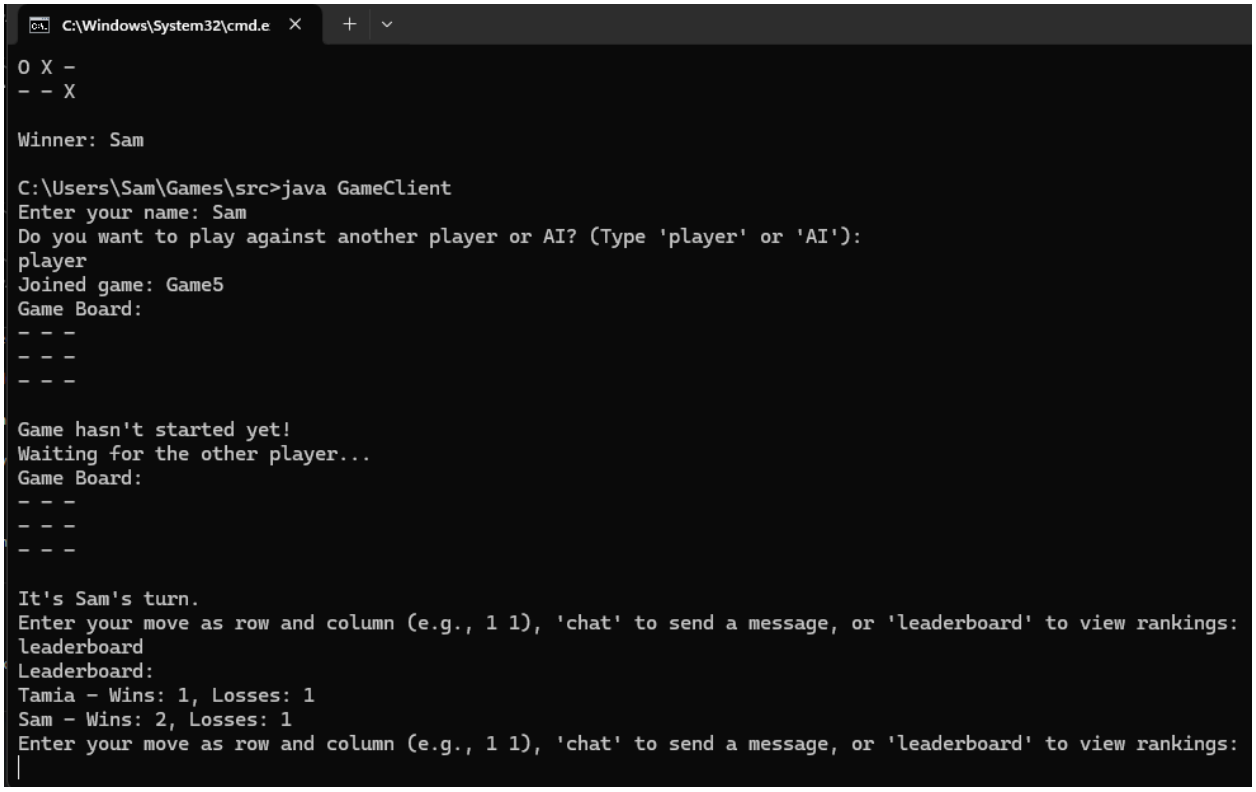

Spectating:



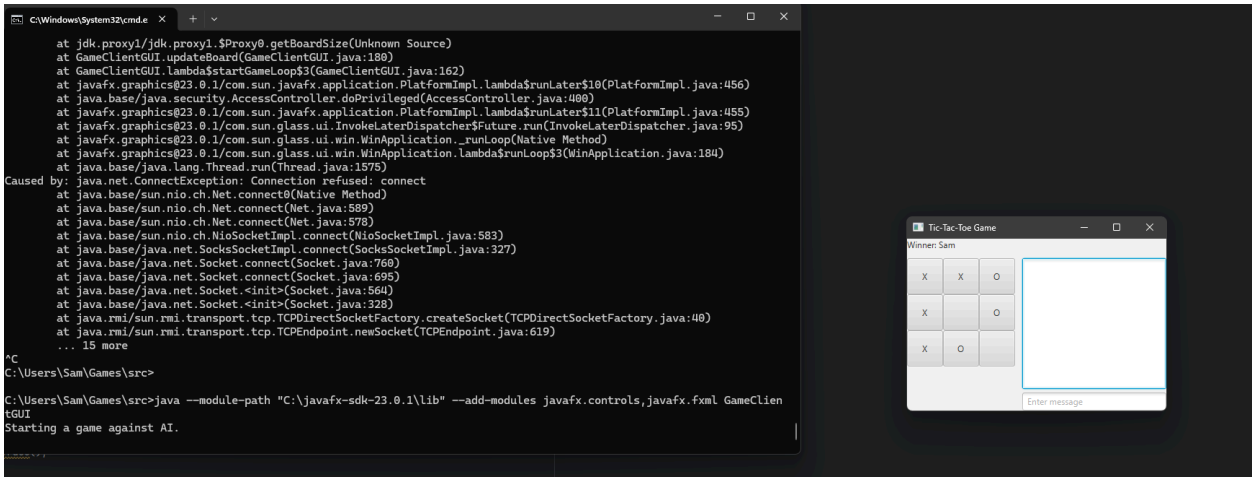

SinglePlayer VS AI:

```
Enter your name: Sam
Do you want to play against another player or AI? (Type 'player' or 'AI'):
AI
Starting a game against AI.
Game Board:
- - -
- - -
- - -

It's Sam's turn.
Enter your move as row and column (e.g., 1 1), 'chat' to send a message, or 'leaderboard' to view rankings:
1 1
Move accepted!
Game Board:
- - -
- X O
- - -

It's Sam's turn.
Enter your move as row and column (e.g., 1 1), 'chat' to send a message, or 'leaderboard' to view rankings:
2 2
Move accepted!
Game Board:
O - -
- X O
- - X

It's Sam's turn.
Enter your move as row and column (e.g., 1 1), 'chat' to send a message, or 'leaderboard' to view rankings:
0 1
Move accepted!
Game Board:
O X -
O X O
- - X

It's Sam's turn.
Enter your move as row and column (e.g., 1 1), 'chat' to send a message, or 'leaderboard' to view rankings:
2 1
Move accepted!
Game Board:
O X -
O X O
- X X

Winner: Sam

C:\Users\Sam\Games\src>
```

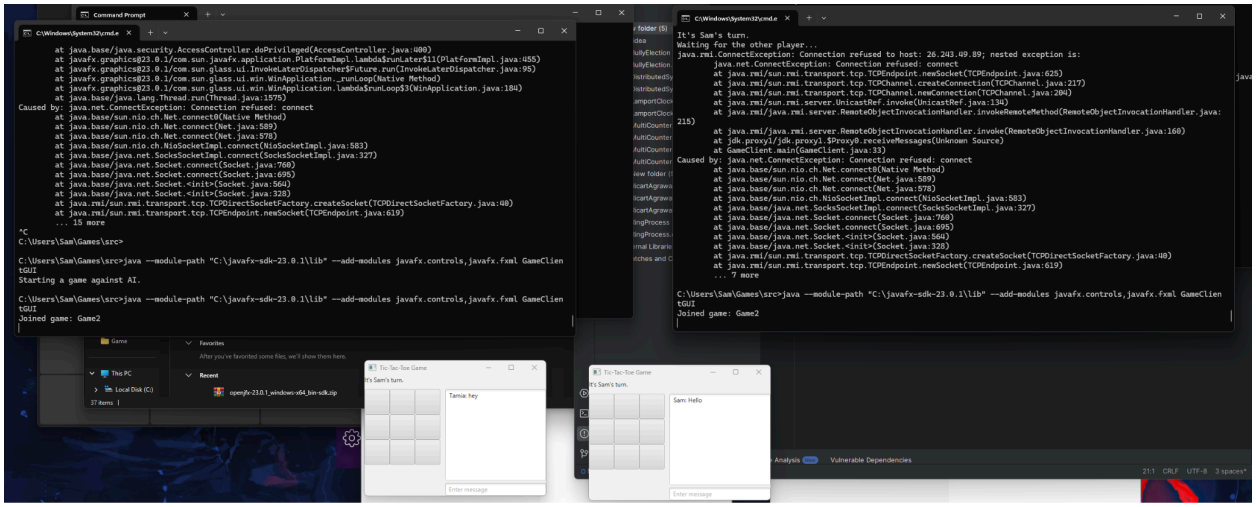
Game Leaderboard feature:



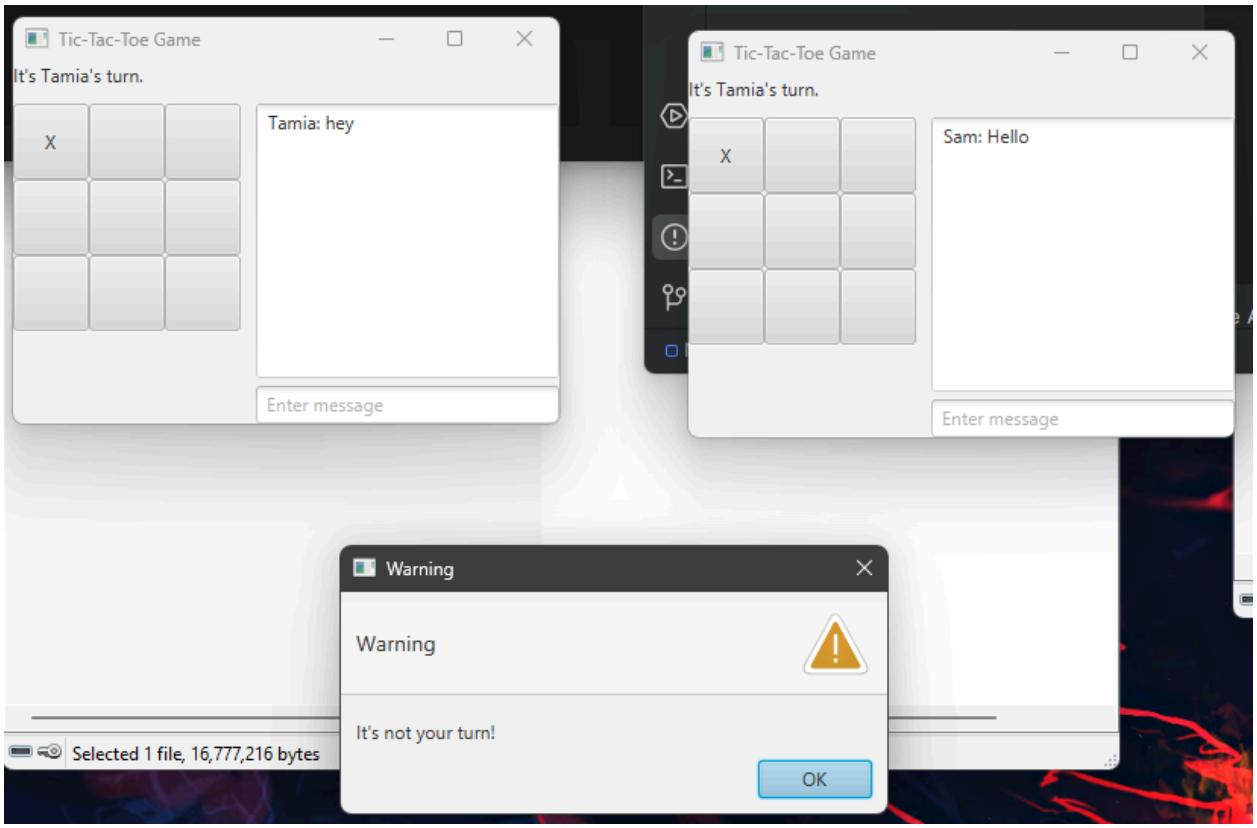

Game GUI (VS AI):



Chat in GUI & Player VS Player in GUI




GUI Wrong Turn Notification:



Board size Changing: