Isai Tinoco Gutierrez, Russell Ferrall

Dr. Laila

CSC 364

6 July 2025

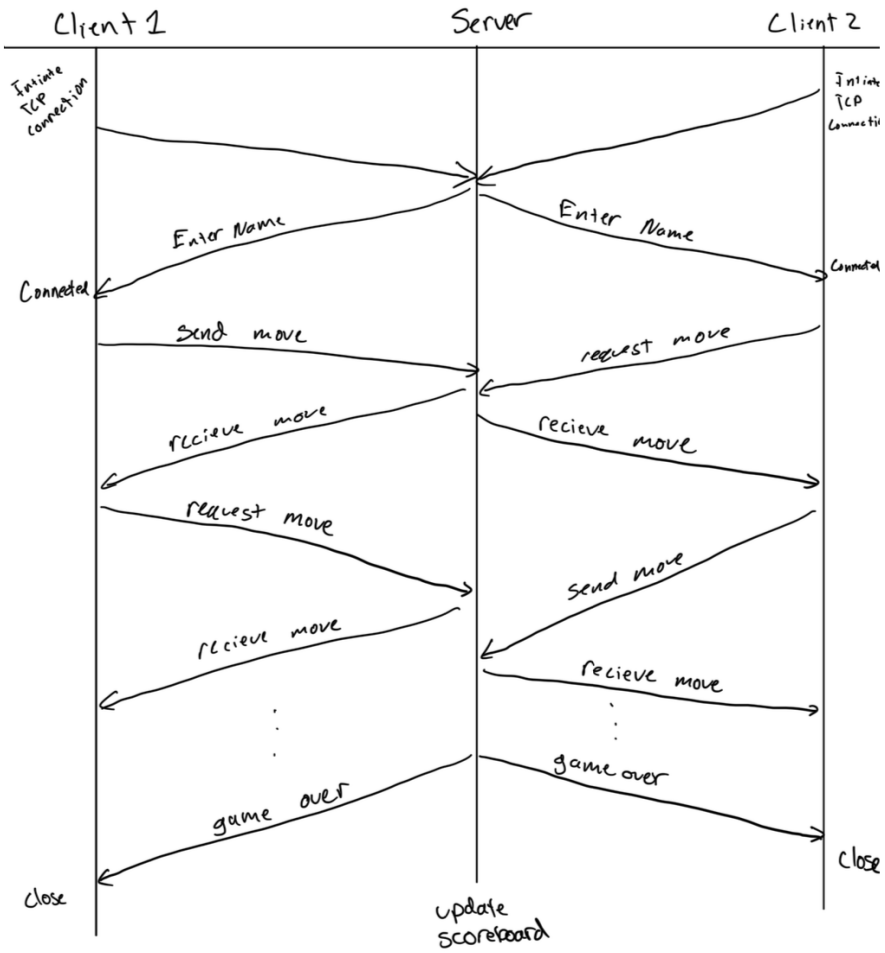
Lab 1 Report

Architecture:

The architecture used in the project is client-server based, where it is a multithreaded server that handles multiple concurrent games. Each client runs a graphical user interface (GUI) that allows users to click on a cell to make a move, rather than typing row and column indices. When two clients connect, the server pairs them and starts a new game thread. Each game session runs independently in its own thread. The scoreboard is saved in a JSON file using a thread-safe lock. Clients are prompted to enter a name at startup, which is stored in the scoreboard file. If there is a disconnect from one of the clients, the game will end, and the remaining client will receive two points. During gameplay, the client sends its move to the server after a box is clicked, and the server updates the board and notifies both clients of the state change.

System Component Diagram:

The system is composed of three core components: two clients and a multithreaded server. Each client connects to the server using a TCP socket of communication via a simple text-based protocol. Once two clients are connected, the server creates a new thread to manage their game session. Within that thread, the server sends game prompts, receives player moves, updates the game board, and determines the outcome. The scoreboard is managed carefully on the server JSON file, and updates are protected with a thread-safe lock to prevent race conditions between concurrent game sessions. This modular structure allows multiple pairs of players to play independently and simultaneously, all while preserving a shared, persistent scoreboard.



Transport-Level Protocol:

TCP is the protocol used because Tic Tac Toe requires a reliable transfer of data, we wanted to ensure that each move is sent to the server in order without the fear of a dropped moves.

Types, Syntax, and Semantics:

The communication between clients and the server follows a structured message format, even though the user interaction is now handled through a graphical interface. Upon launch, the client prompts the user to enter their name via a GUI text input. During gameplay, the user makes moves by clicking directly on the desired grid cell, which automatically sends the corresponding (row, col) coordinates to the server. These messages are simple encoded strings that represent the user's intent without requiring

manual input. The server responds by validating the move, updating the internal game state, and sending back the new board configuration. The GUI on both clients is updated to reflect the current state of the game and whose turn it is. Messages such as invalid move alerts or game results (“You win!”, “Opponent wins”, or Draw”) are also displayed through the GUI, making the experience more intuitive. It will display your name and opponent’s name instead. While maintaining a clean communication protocol underneath.

Rules for Sending/Receiving Messages:

The server initiates all communication by sending prompts and updates, while the client responds only when explicitly asked for input. After two clients are connected, the server asks each for the players’ name and stores these on the scoreboard. During gameplay, the server alternates each turn by prompting the current player, receiving and validating their move, and then updating both clients with the current board state. The server uses blocking receive calls (recv) to wait for responses, ensuring synchronization between game actions. Once the game concludes, the final board state and result are sent to both clients, and the server updates and saves the scoreboard file before closing the connections.