COMP90015: Distributed Systems – Assignment 2 Distributed Tic-Tac-Toe System

Zening Zhang 1078374

October 13, 2023

Contents

1 Introduction

This project involves designing and implementing a classic two-player game Tic Tac Toe game by using a using a client-server architecture.This system should handle the suituation when client concurrently requests to make moves, send chat messages, and update game states in real-time. To reach this standard, this gaming system built on Java's Remote Method Invocation (RMI) framework. Besides, the system facilitates online matches between two players and comes equipped with various features such as player ranking, game state preservation and restoration, and reconnection after unintentional disconnections. This report will delve into the rationale behind the design choices and the critical performance aspects of the system, and emphasizes object-oriented design principles, event-driven programming, and network communication to craft an engaging and competitive gaming experience.

2 System Components

The Tic Tac Toe gaming system is intricately divided into several components, each designed to perform specific functions, ensuring smooth gameplay and efficient communication.

2.1 Client

The client side of the Tic Tac Toe game system serves as the primary interface for players. Its responsibilities include:

2.1.1 Client

- Processing player inputs and sending appropriate requests to the server.
- Receiving updates from the server and relaying them to the GUI for real-time reflection.
- Handling scenarios like disconnection and reconnection, ensuring players can re-join games.

2.1.2 ClientGUI

- Providing an intuitive graphical user interface that allows players to interact with the game.
- Displaying the game board, player moves, chat messages, and more.
- Showing notifications such as game results, player rankings, and other game-related alerts.

Figure 1: Client End Class Diagram

2.2 Server

The server side of the Tic Tac Toe system orchestrates game sessions, player, and overall game state management (TicTacToeServer).

2.2.1 TicTacToeServer

As the core component of server, it can handle:

- Managing multiple game sessions concurrently.
- Broadcasting game state updates to the respective clients.
- Implementing fault-tolerance mechanisms to handle client disconnections.
- Providing a ranking system for players based on their game performance.

2.2.2 GameSession

The GameSession is responsible for each game with two players, all the things happend in game will be handled, recorded and controlled by GameSession. Each GameSession represents an individual game being played between two players. It is responsible for:

- Maintaining the game state, including board state, player scores, and turn management.
- Ensuring synchronization between two players within a session.

2.2.3 Player

The Player class on the server side represents each player. It keeps track of:

- Player details like name, game symbol (X or O), and current score.
- Communication interface for the player to interact with the server.

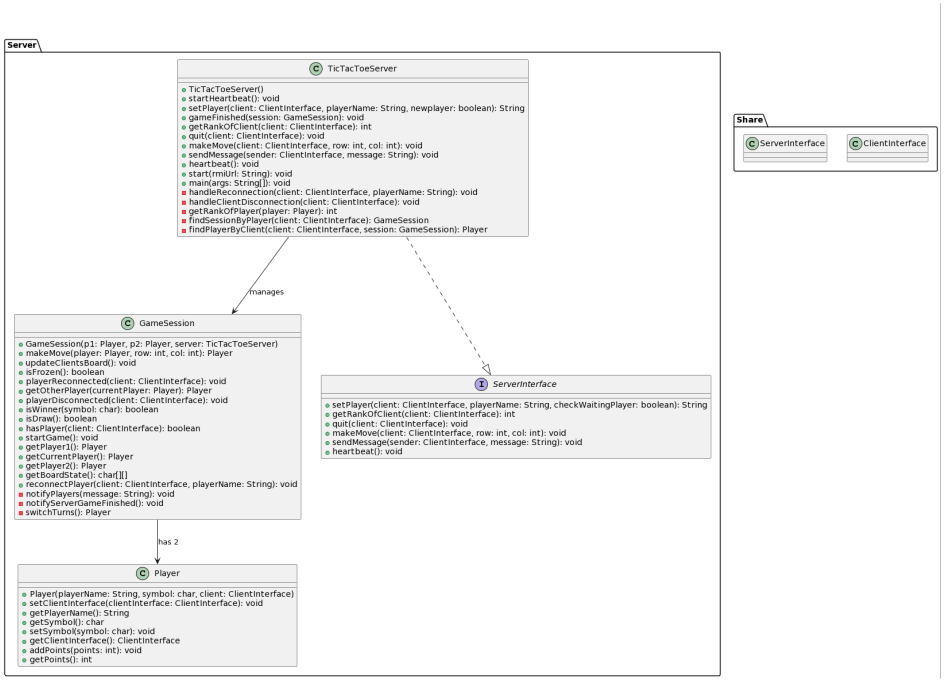

Figure 2: Server End Class Diagram

2.3 Share

This module facilitates shared functionalities and communication structures between the client and server by using the strategic RMI. It contain the ClientInterface and the ServerInterface, they include functions of the Client and TicTacToe classes, which need to use RMI to communicate and invocation.

3 Interaction Diagram

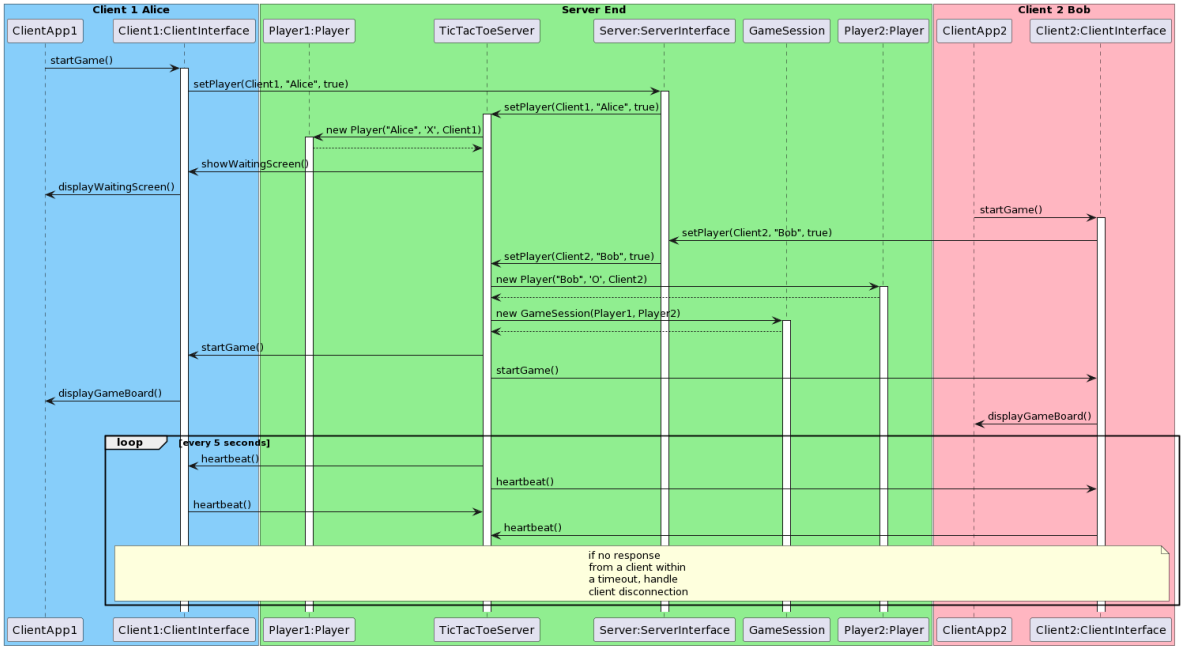

Figure 3: Interaction Diagram of system

In this interaction diagram, the process of establishing a TicTacToe game session between two clients (Alice and Bob) and the server is depicted. Initially, Alice starts the game and requests the server to set up a player for her via the ClientInterface. In response to this, the server creates a Player object representing Alice and instructs Alice's client to display a waiting screen, as there's no second player yet. Subsequently, Bob starts his game and also requests the server to set him up as a player. The server creates a Player object for Bob and initiates a new GameSession that encompasses both Alice and Bob. Both clients are then notified to start the game and display the game board. To ensure both clients maintain an active connection, the server and client send a heartbeat signal to each others every 5 seconds. If no response is received within a predefined timeout, the server will handle the disconnection of client or server.

4 Architecture & Protocols

4.1 System Architecture

4.1.1 Server

The server mainly contains those functions which are Central Repository, Session Management, Player Profiles, Heartbeat Mechanism and Reconnection Support. They cover all the essential part of what a server needs. The Central Manager acts as the server, orchestrating the entire game process, from managing connected clients and active game sessions to handling waiting players. Each game session, created and maintained by the server, represents a unique match between two players, overseeing

the game board state and monitoring whose turn it is. When a connection request is received, the server establishes a player profile; if an opponent is ready, a game session begins, but if not, the player is queued until a match is available. To maintain connection integrity, a heartbeat mechanism is employed, periodically verifying client presence and addressing any unresponsive clients. Moreover, if a client disconnects but reconnect later, the server can restore the game state, enabling players to pick up where they left off

4.1.2 Client

The client contains the function of game initation, movement making, message sending and game progress displayment, Heartbeat Mechanism. As the player use the client to initiates a game request, the server either places them in a waiting state or pairs them with another player to commence the game. Clients relay their moves to the server, which processes and updates the game state before sending the refreshed game board to both players. Additionally, during an active session, players can communicate via messages that are routed through the server, ensuring centralized communication. Throughout the game, clients remain updated on its progression, receiving notifications about their opponent's moves, game outcomes, and other pertinent in-game events. Also similar with the hearbeat mechanism in server end, it periodically verifies the status of the server whether it is connect or not. If the server disconnects, it will display the notice to remind the player that the server is down the client will close after 5 second.

4.1.3 Clientinterface and Serverinterface

Also in the system Architecture, these two key pieces called the ClientInterface and ServerInterface. They are the rule of how to communication between the server end and client end. Essentially, the ClientInterface defines operations that the server can invoke on the client, such as updating the game board or notifying of game results. Conversely, the ServerInterface prescribes methods the client can call on the server, like starting a game or making a move. These interfaces are the linchpins that ensure smooth, structured, and predictable communication between the distributed client and server entities.

4.2 Communication Protocol

The communication protocol of the TicTacToe game application leverages the capabilities of Java's Remote Method Invocation (RMI) to facilitate seamless interactions between distributed entities. RMI empowers Java objects residing within one virtual machine to effortlessly invoke methods on objects in another JVM, effectively bridging the client-server gap in our application.

For our game, the server hosts a TicTacToeServer object, which is the central hub of game logic and management. Clients communicate with this server object to perform essential game operations, such as initiating a game, making moves, or sending chat messages to opponents. Conversely, the server can also reach out to individual clients through the ClientInterface to update game states, notify of turn changes, or convey game results.

The advantage of using RMI is it can abstrac the detail between the commnucation. This abstraction enables the TicTacToe system to concentrate on delivering a fluid, interactive gaming experience. Furthermore, the RMI protocol ensures reliable and synchronized communication, which is a critical aspect to preserve the integrity and real-time nature of the game state across all participants.

5 Additional Features

5.1 Client-side Timeout

I implemented the client-side timeout feature to enhance the user experience and ensure the game progresses smoothly. By setting a timer on the client side, players are encouraged to make timely moves. If a move isn't made within 20 seconds, the client will automatically choose a valid move on behalf of the player. This client-based approach not only alleviates the server's burden of managing timers for every active game session but also eliminates potential time lags that might arise from network communication.

5.2 Fault Tolerance

Ensuring uninterrupted gameplay was paramount. I divided fault tolerance into two categories: server disconnection and client disconnection. The heartbeat mechanism is integral to this fault tolerance system. Every 5 seconds, the client and server exchange 'heartbeat' signals. If either party fails to receive this signal consecutively, it's assumed the other has disconnected.

5.3 Server Disconnection

In the unfortunate event of a server disconnection, all game data, including player rankings and game statuses, would be lost. The client recognizes this and triggers a five-second countdown, at the end of which the client application gracefully shuts down.

5.4 Client Disconnection

This scenario is treated with more leniency. If a client disconnects, they're given a 30-second window to reconnect. During this period, the disconnected player is added to a 'disconnected list'. If they attempt to reconnect while on this list, the server retrieves their previous game session, restoring their game status. Throughout the disconnection, the opponent is temporarily barred from making moves or sending messages, ensuring game fairness.

5.5 Player Ranking System

Player ranking is a collaborative effort between the Player and Server classes. While the Player class keeps track of individual scores based on game outcomes, the server compiles this data, sorting and ranking players accordingly. In case of tied scores, players are ranked alphabetically.

6 Critical Analysis

6.1 Challenges Faced

Building a multiplayer game presented unique challenges, especially when ensuring real-time synchronization between clients in different JVMs. Handling disconnections gracefully while maintaining game state and ensuring that players had a consistent experience required careful planning and implementation.

6.2 Design Choices

I employed Java's RMI for its seamless server-client communication. To address concurrency challenges, I implemented synchronization mechanisms, ensuring thread-safe game state updates. Additionally, client-side timeouts were integrated to maintain game flow and manage server resources. A player ranking system was also introduced, adding a competitive layer and motivating players to engage more.

6.3 Possible Improvements

Looking ahead, I see opportunities to enhance the user interface, introduce more interactive game elements, and perhaps even offer multi-board games where players can engage in multiple games simultaneously. Additionally, introducing a database could preserve game states and rankings even if the server disconnects.

7 Error Handle

To ensured a smooth, consistent, and error-free gaming experience

- Remote Exception Handling: Remote method calls were safeguarded with try-catch blocks to handle RemoteException, ensuring system stability.
- Player Name Collisions: Checks were in place to prevent duplicate player names, with appropriate user feedback.
- Game State Consistency: Synchronization techniques ensured no unexpected outcomes from concurrent moves.
- Malformed URLs: Errors while binding to the RMI registry were handled gracefully.
- Feedback to Clients: Any errors prompted immediate user feedback, ensuring clarity on the game's state.

8 Conclusion

Creating the TicTacToe game was both challenging and rewarding. While the journey presented its share of challenges, the final product is a testament to the power of RMI and the potential of well-thought-out design choices. As I reflect on this project, I'm inspired to explore more advanced features and game dynamics in the future.