COMP90015 Distributed System
Assignment 2 Report
Chenyang Dong
1074314

Problem Context

This aim of this assignment is to implement a distributed Tic-Tac-Toe gaming system, allowing multiple players to connect and play while facilitating real-time chatting. In such distributed system, it needs to allow multiple clients to interact with the server and perform operations concurrently. All communication between the server and clients must be reliable, utilizing either Java sockets or RMI, based on the chosen implementation approach. It is important to have proper error management on both server and client sides. This involves implementing exception handling to address errors such as input parameters, network communication, I/O to and from disk and so on.

The client application encompasses several essential functional requirements, including the Tic-Tac-Toe game and real-time chat. Additionally, the system incorporates two valuable features: client-side timeout, allowing a 20-second move time limit to with a visual countdown timer, and fault tolerance, which temporarily pauses games for 30 seconds upon a client crash, with the option to resume, or ending as a draw if the player does not return. These functionalities should be integrated into a user-friendly graphical user interface (GUI) within the client program. On the server side, the responsibilities include player matching, maintaining the current game state, and managing the player ranking system. The system should be designed to handle any potential errors that may arise, ensuring that users are informed promptly while performing the operation.

System Component

Server Component

The server component employs a multi-threaded architecture, a fundamental design aspect of the system to efficiently manage player interactions and game sessions. Each thread represents a server connection dedicated to communicating with a client. These threads effectively handle player connections, match pairing, and game session management. In particular, they incorporate specialized threads, such as heartbeat threads that initiate when a connection is established, continuously monitoring client status, and countdown threads that engages when clients disconnect. The multi-threaded design enables concurrent processing of player actions, ensures real-time communication between clients, and efficiently manages scenarios where clients disconnect or experience interruptions. Through this architecture, the server guarantees a responsive and seamless gaming experience, with each thread waiting for client requests and responding promptly.

Client Component

The client application establishes a connection with the server with a specified username, marking their presence in the system. After connecting to the server, the client will act as the bridge for sending and receiving data between the user interface and the server. Each user interaction, such as making a move or engaging in chat communication, prompts the client

to transmit these actions to the server. This transmission allows the server to process the actions and efficiently relay them to the other client involved in the match, ensuring that every in-game operation is coordinated and synchronized in real-time. Within this framework, specialized threads, similar to those on the server are integral to the system, handling server connections, monitoring server availability through heartbeat threads, and ensuring timely responses through countdown threads.

TCP/IP Communication via Sockets

The client-server communication relies on the TCP/IP protocol facilitated by sockets. When a client initiates communication, a socket is created to establish a connection with the server to ensure reliable and ordered data transmission.

JSON Message Exchange Protocol

To facilitate meaningful and structured data exchange, JSON is chosen to serve as the message exchange protocol between the server and client. When the client sends a message, the data is formatted as JSON objects, with operation types such as 'player' for joining the game, 'play' for making a move on the board, 'chat' for communicating with the other player, and 'quit' for leaving the game. The server processes these JSON requests, executing the required actions such as updating the player ranking, and then responds or directly communicates with the other player in the respective match by using JSON-formatted messages. The server's messages also include various types to inform the client of the necessary actions. For example, when one player receives a message from the server about the move made by their opponent, they will update the game board. By using this protocol, it simplifies the data handling and allows clear communication between the client and server.

Example:

```
// New match message from server
{   "type":"new match",
    "matchId":3,
    "playerUsername":"player1",
    "playerSymbol":"O",
    "playerRank":2,
    "opponentSymbol":"X",
    "opponentUsername":"player2",
    "opponentRank":7
}
```

```
// Recovery message from server
{   "type":"old match",
    "matchId":3,
    "playerUsername":"player2",
    "playerSymbol":"X",
    "playerRank":7,
    "opponentUsername":"player1",
    "opponentSymbol":"O",
    "opponentRank":2,
    "playsO":"127",
    "playsX":"56"
}
```

```
// Play message from client
{   "type":"play",
    "symbol":"O",
    "position":"7",
    "matchId":3,
    "username":"player1"
}

// Play by opponent message from server
{   "type":"play",
    "symbol":"O",
    "position":"7",
    "matchId":3
}
```

```
// Quit message from client
{   "type":"quit",
    "symbol":"O",
    "matchId":3,
    "username":"player1"
}

// Match over message from server
{   "type": "over",
    "winner": "player2",
    "symbol": "X",
    "new rank": 5
}
```

Class Design

Server Class Structure

Presented in Figure 1 is the UML Diagram depicting the structure of the *Server* program. The core components of the server include the ***TicTacToeServer***, ***TicTacToeServerConnection***, ***TicTacToeGame***, ***Player*** and ***MatchData*** classes. The ***TicTacToeServer*** serves as the central hub of the server program, overseeing the game's operations. It manages connections with clients via the ***TicTacToeServerConnection*** class, enabling efficient communication. The ***TicTacToeGame*** class is responsible for the game's logic and coordination. It connects with the ***Player*** and ***MatchData*** classes to organize players into matches and maintain their game state. The ***TicTacToeGame*** class maintains information about players, matches, and the queue for players waiting to join a match. The ***Player*** class represents individual players, and the ***MatchData*** class stores essential information about ongoing matches. These classes work in conjunction with the ***TicTacToeServer*** and ***TicTacToeGame*** to ensure seamless gameplay and player interactions.

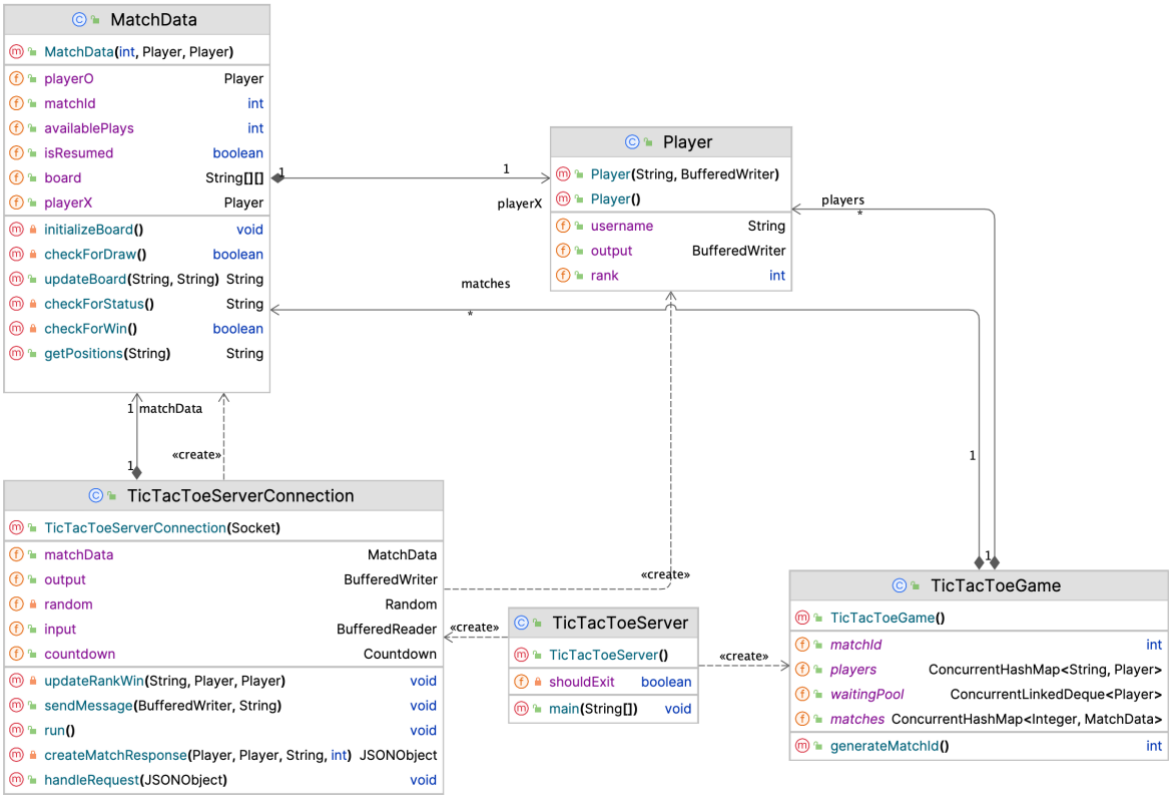


Figure 1: UML Class Diagram (Server)

Client Class Structure

In Figure 2, the UML Diagram is presented to illustrate the structure of the Client program. The key components include the ***TicTacToeClient***, ***TicTacToeClientConnection***, ***Player*** and ***GUI*** classes. ***TicTacToeClientConnection*** class effectively manages communication with the server by constructing and dispatching JSON-based requests for various operations. The ***Player*** class is closely connected to ***TicTacToeClientConnection*** since each client corresponds to a player. The ***GUI*** class provides users with an intuitive interface. As users input data and interact with the graphical elements, the ***TicTacToeClient*** class coordinates necessary actions for interacting with the server.

Figure 2: UML Class Diagram (Client)

Interaction Diagram

The interaction diagram shown in Figure 3 illustrates the seamless communication between the client application, server, and other clients. After the client application starts, it will start finding a match by entering the waiting pool of the server. Once a match is established, the client engages in gameplay, communicating various actions such as moves and chat messages to the server. The server processes these actions and efficiently relays them to the corresponding opponent, ensuring real-time updates for both clients. Additionally, the diagram depicts a scenario where, in the event of one client disconnecting due to a client crash or player closing the application, the server takes prompt action. It temporarily pauses the ongoing match, sending a 'wait' message to the other client to inform the situation. When the disconnected client successfully reconnects, the server transmits the match data, allowing for a seamless recovery of the game. Simultaneously, a 'resume' message is sent to another client, ensuring that both participants can continue the match without interruption. If the disconnected client does not reconnect in the 30-second time limit, a message indicating the match drawn will be sent to another client with the match data updated.

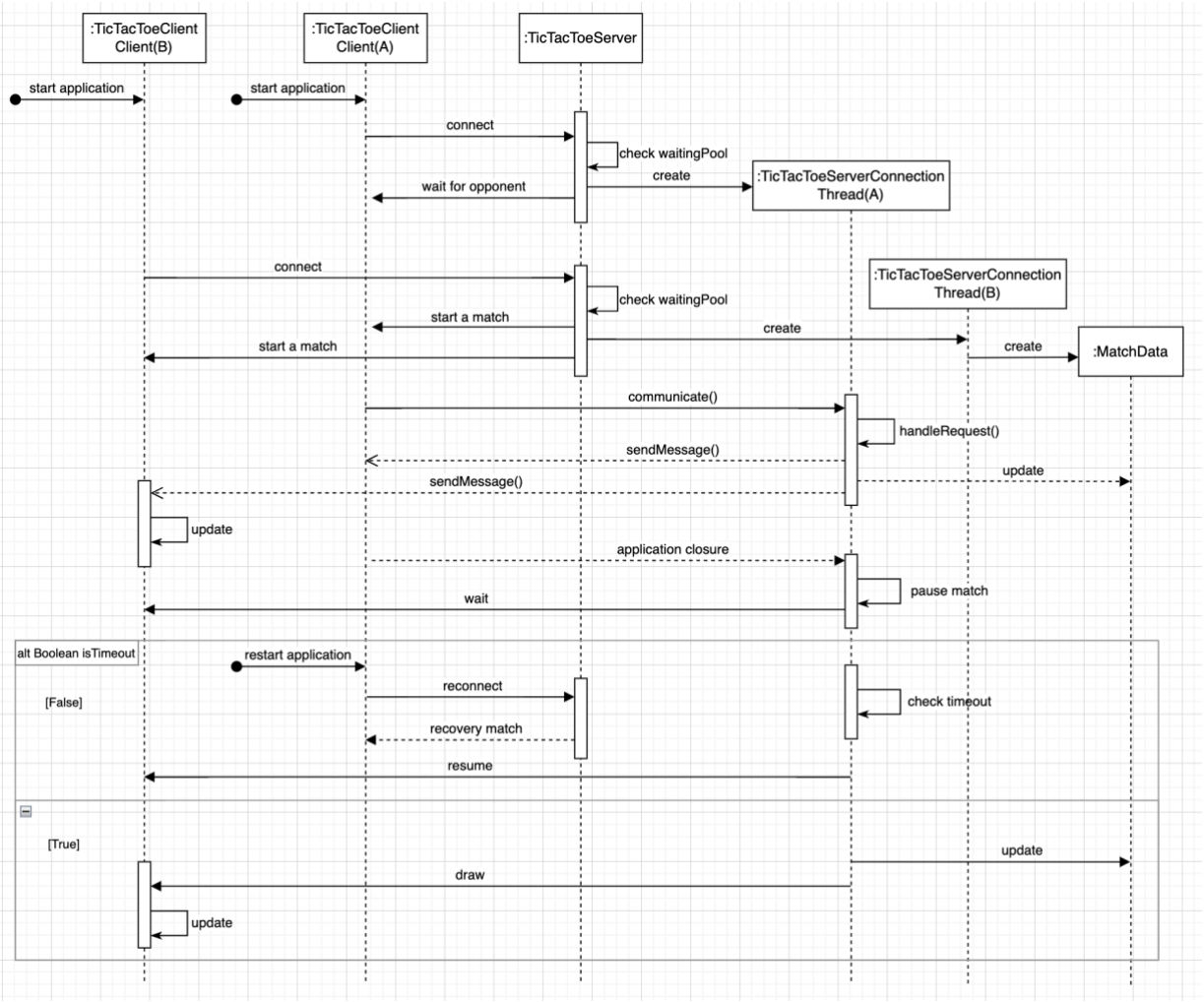


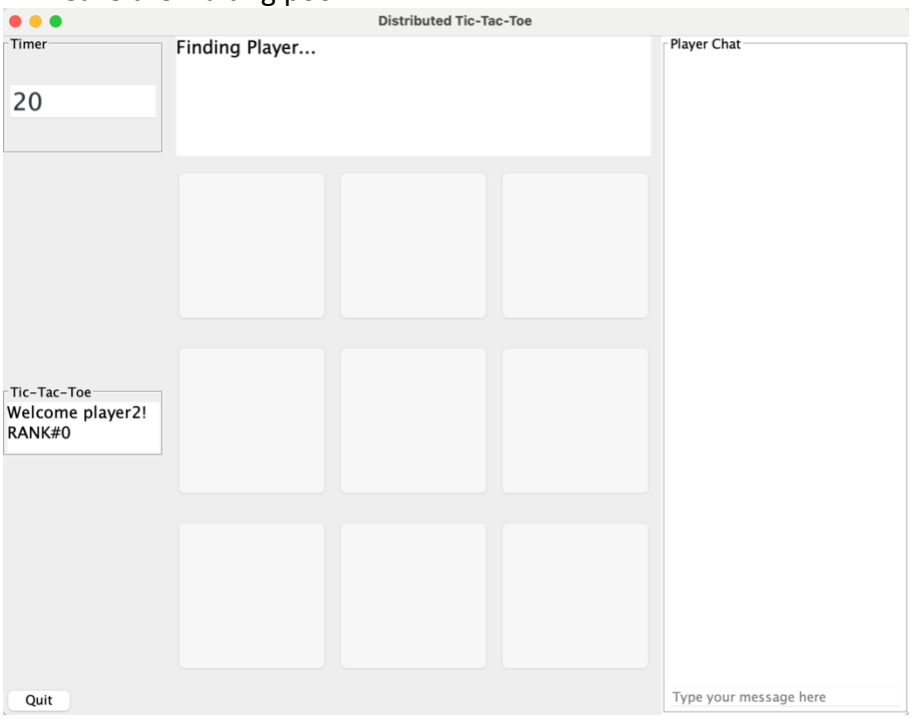
Figure 3: Interaction/Sequence Diagram

Critical Analysis

Functionality

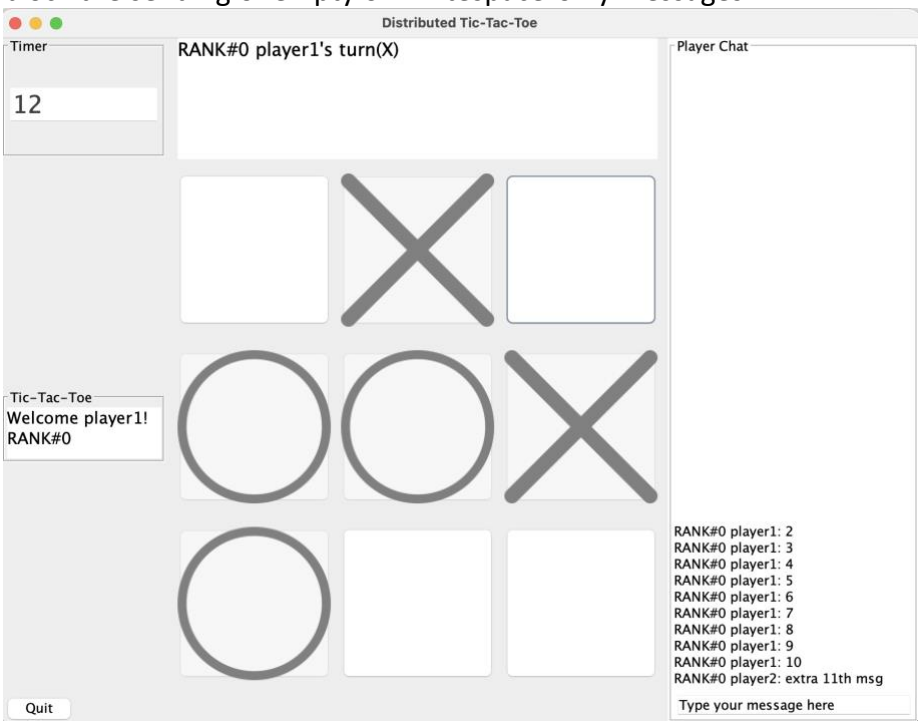
All functional requirements have been satisfactorily fulfilled as shown in graphs below. Through the systematic implementation of various features and components, the system effectively meets its intended goals.

1. Once the client launches the program, the player is presented with an empty board, with a text on top of the board showing "Finding Player" until the server successfully matches the player with an opponent. The matchmaking process involves placing the player into a waiting pool within the system, ensuring that an even number of active players is reached before proceeding to initiate the game.
2. The name and the rank of the player are always displayed at the left.
   A quit option is provided to exit the system anytime on the bottom-left of the window. If the player exits the system or the client crashes while finding a match, it will leave the waiting pool.

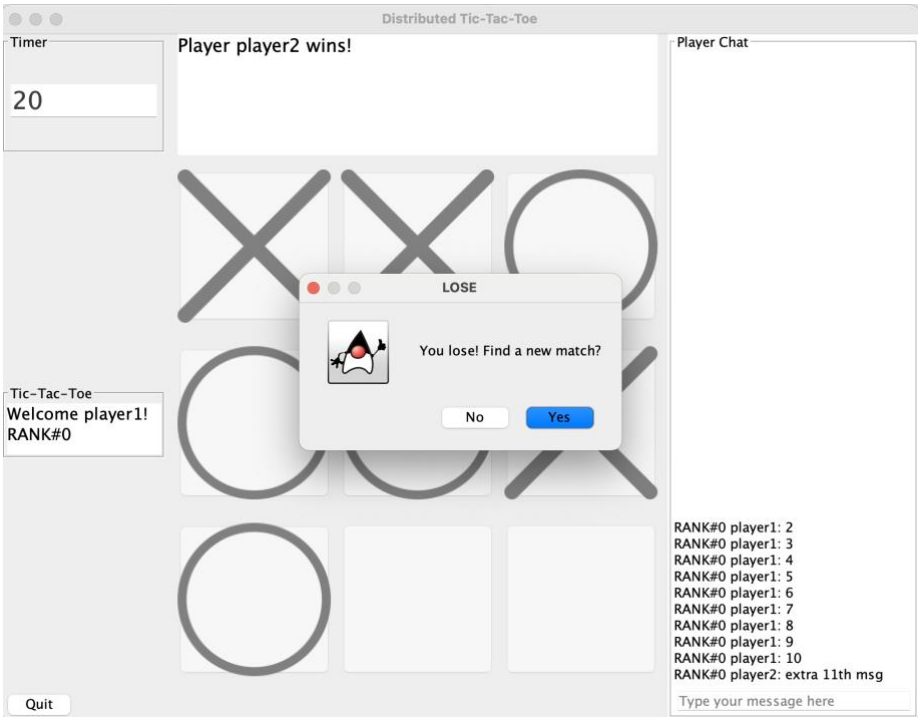


3. After a game starts, a text is shown on top of the board showing the current player's name, rank, and symbol. The symbol of the player is randomly assigned by the server.
4. During the player's turn, the player can click on an empty cell of the Tic-Tac-Toe board to place their assigned symbol, indicating their move. The client then promptly informs the server of the current move, and the turn switches to the other player. Both players engage in turn-based gameplay, with the state of the Tic-Tac-Toe board visually synchronized on both of their client GUIs in real-time.
5. Players are given a 20-second time limit to make their moves during their turn. If a player exceeds this limit, the client will automatically select an empty cell. A countdown timer on the top-left displays the remaining time, starting at 20 seconds and decrementing with each passing second.

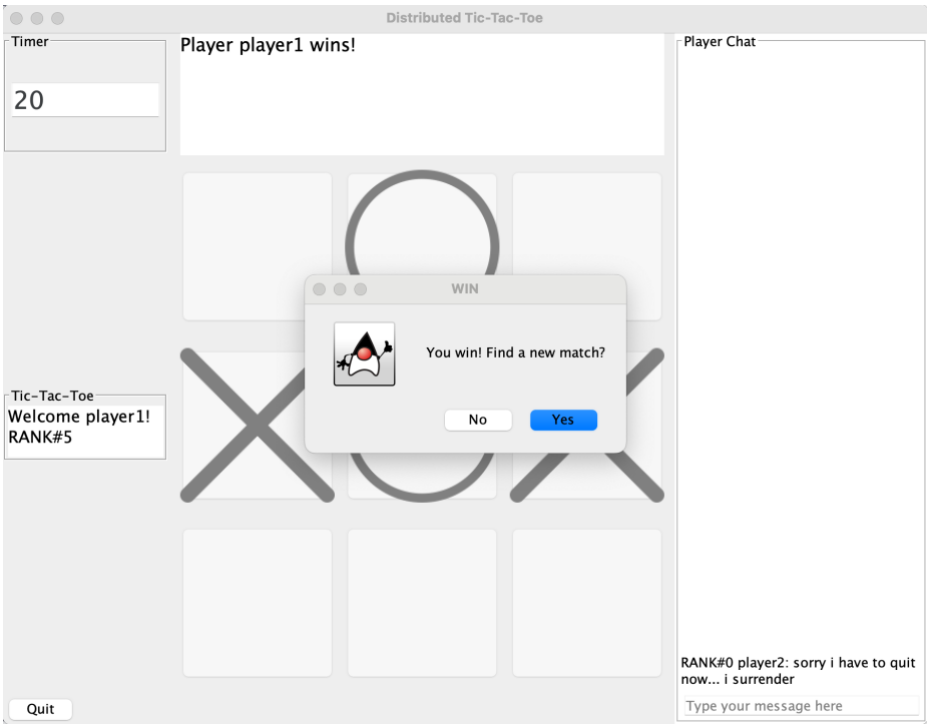
6. The system continuously checks for a winning condition after each move made by the players.
7. A chat window on the right side of the GUI allows players to engage in real-time communication with their opponents. The chat window displays the most recent 10 messages, with the newest message appearing at the bottom. It is also designed to block the sending of empty or whitespace-only messages.



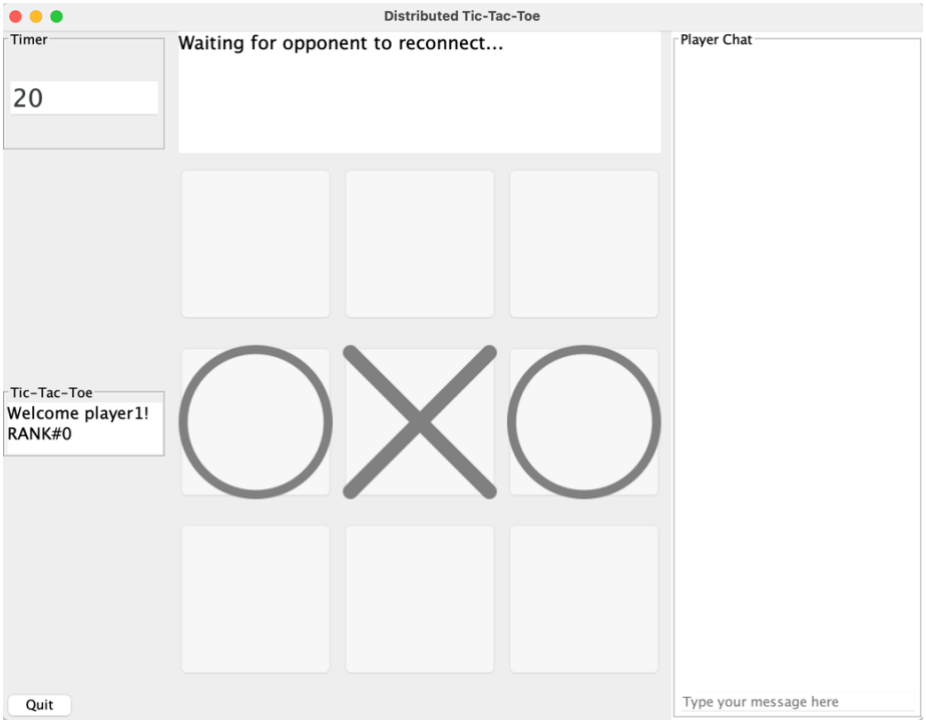
8. After a game is complete, the result is presented above the game board. If a player wins, their name is displayed. In the event of a draw, the message "Match Drawn" will be shown.
9. Also, players are presented with the choice to either find a new match or quit the system with a dialog box after a game is complete. If the player chooses ‘yes’, they will be placed back into the waiting pool; otherwise, the player exits the system.
10. Every player begins with a rank point of 0. Their rank is tracked by the server, adjusted as follows: they gain 5 points for a win, lose 5 points for a loss, and receive 2 points for a draw.



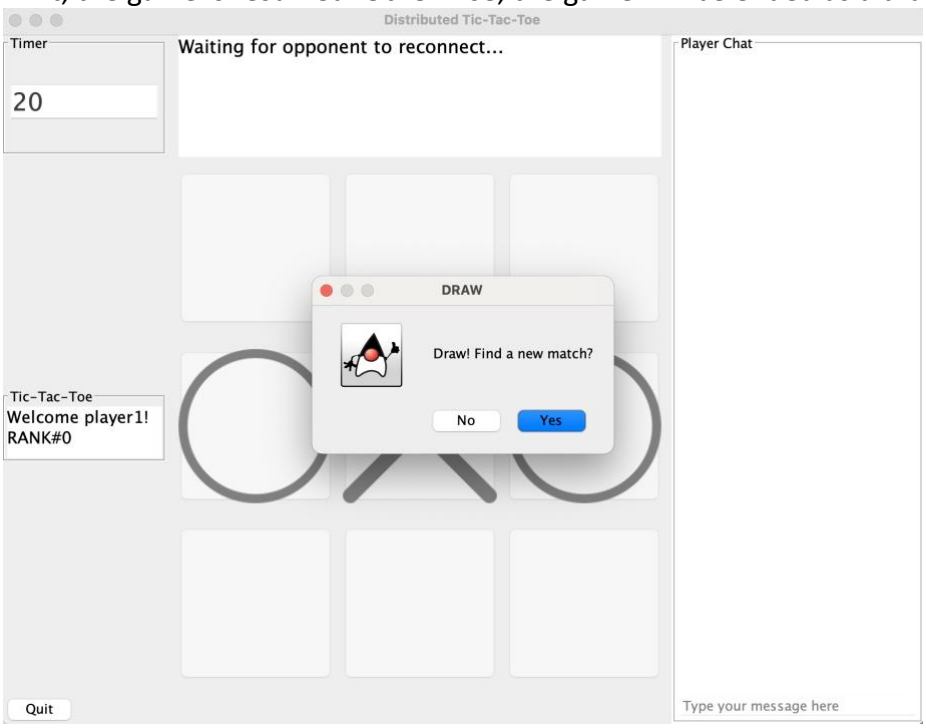
11. If a player decides to exit the system during an ongoing game using the quit button, the current match is automatically forfeited, and the opposing player is declared the winner.



12. In the event of a client crash or if a player closes the client application, the ongoing game is temporarily paused for a duration of 30 seconds. A message, "Waiting for opponent to reconnect," will be displayed above the game board of the opposing player. Additionally, the timer for the player who is expected to make the next move will also be reset.



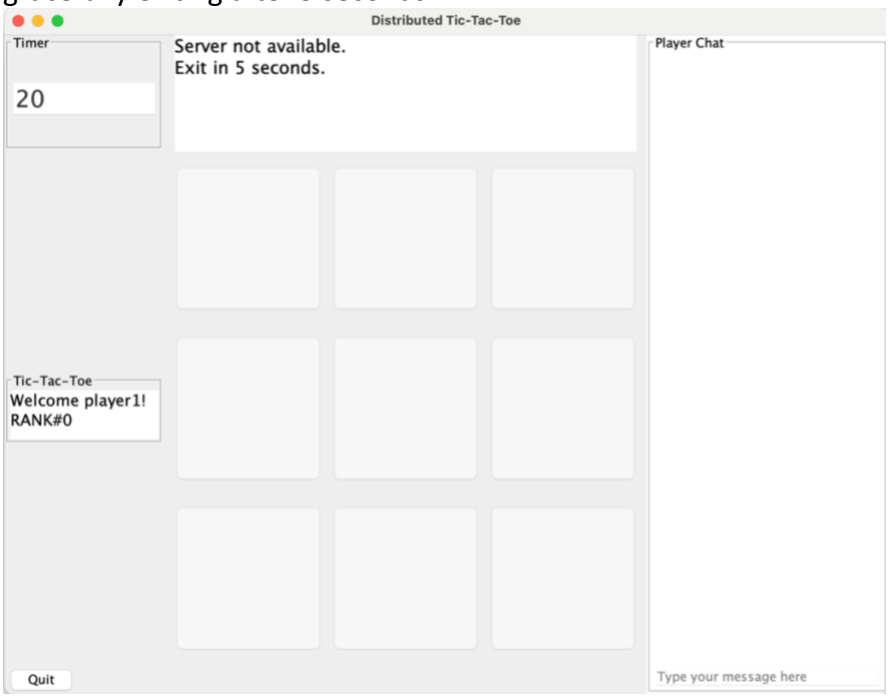
13. If the disconnected player successfully rejoins the system within the 30-second time limit, the game is resumed. Otherwise, the game will be ended as a draw.



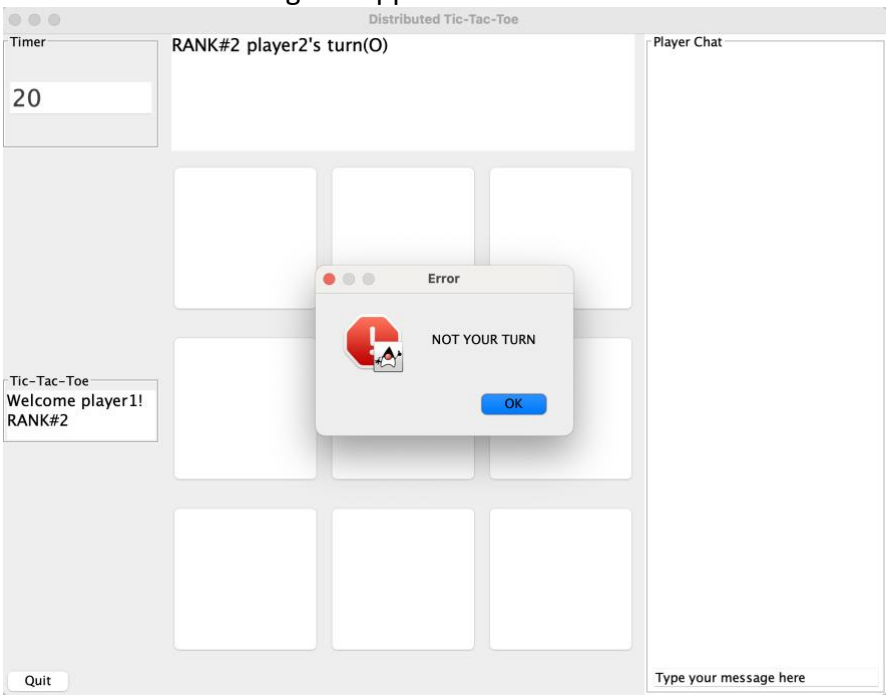
Error Handling

As the examples shown below, whenever a user initiates an incorrect request, the server promptly responds with an informative error message shown in a dialog or using the text box above the game board, guiding the user towards the correct usage. This enhances the user experience by minimizing confusion and facilitating correct interactions.

1. In the event of a server crash, clients receive a "Server unavailable" message, gracefully exiting after 5 seconds.



2. During the opponent's turn, if the player clicks on an empty cell of the Tic-Tac-Toe board, an error message, "NOT YOUR TURN", will be displayed in a dialog. However, to enhance the user experience and reduce confusion, it may be more effective to disable the cells during the opponent's turn.



Advantage of System

The Tic-Tac-Toe gaming system presents several key advantages. The utilization of TCP sockets allows for reliable and ordered data transmission between the client and the server. The system offers a real-time gaming experience, with integrated chat functionality. Also, it employs robust fault tolerance mechanisms to address both server and client disconnections effectively. The user-friendly GUI ensures a smooth and intuitive gaming experience. Error handling and reliability mechanisms prevent operational issues. The system demonstrates potential for scalability, allowing concurrent and seamless interactions among the clients.

Future Improvement

It's worth noting that an alternative communication method, RMI (Remote Method Invocation), could also have been employed. RMI provides its own set of advantages, such as seamless remote method invocation and object passing, which can simplify certain aspects of the distributed system. Exploring how RMI might improve the system's efficiency and scalability is a worthwhile consideration for future development. In addition, it’s important to recognize that the system currently assumes the server is always running. In future iterations, a mechanism to store match data in a database in case of server downtime could enhance the system's robustness. Additionally, addressing the uniqueness of usernames within the system and ensuring that multiple players don't accidentally use the same username is an area for improvement.

Conclusion

In conclusion, a distributed Tic-Tac-Toe gaming system is developed for multiple players to connect and play while facilitating real-time chatting. With a foundation built on reliable communication using Java sockets through TCP and a well-defined JSON message exchange protocol, the system ensures seamless interactions between the server and clients. Effective error management is implemented on both sides to handle issues like input errors and network problems, ensuring a smooth and orderly termination process. The functionalities of the Tic-Tac-Toe game including real-time gameplay, chat communication, client-side timeouts, and fault tolerance are integrated into a user-friendly GUI within the client program, allowing prompt user notification in case of errors or issues during operations.