COMP90015: Semester 2, 2023
Assignment 2: Distributed Tic-Tac-Toe System

Release date: 14 September 2023
Due date: No later than 5pm Friday 13 October, 2023 AEDT

Weight: 25% of the final mark

1 Assignment Overview

In this assignment, you will build a distributed Tic-Tac-Toe gaming system using Java. This system will not only allow multiple players to connect and play but also facilitate real-time chatting between players. You will have the choice of implementing this assignment using either Java sockets or RMI (Remote Method Invocation). The game will feature a graphical user interface (GUI) for the client and a central server (no GUI) responsible for managing the games.

1.1 Game Rules

In the classic Tic-Tac-Toe game, two players take turns marking an empty square on a 3x3 grid with either 'X' or 'O'. The primary objective is to get three of your marks in a row, either horizontally, vertically, or diagonally, to claim victory. If all the squares on the grid are filled, and no player has achieved a winning pattern, the game is considered a draw.

2 Basic System (12 marks)

2.1 Client

Your client-side application will feature these essential components and functionalities:

/ • When the client application is launched for the first time, a player should be able to pass their username as a command-line argument. You may assume that the usernames are unique in the system, and multiple players won't accidentally pass the same username.

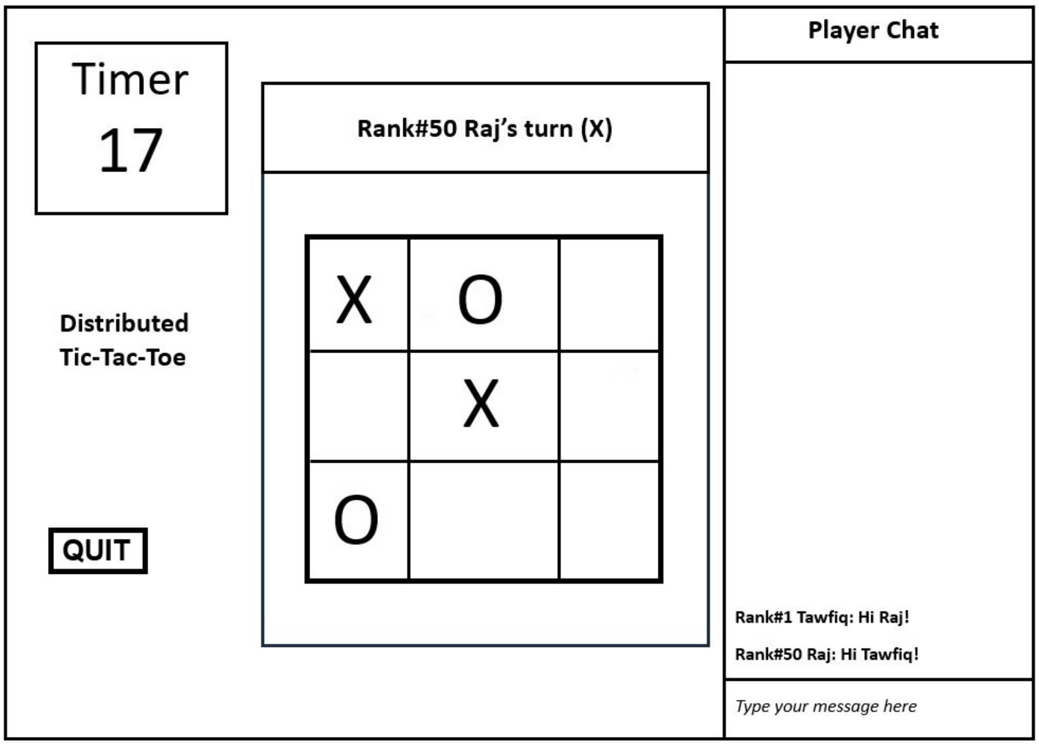
/ • A simple graphical interface (GUI) that visually represents the Tic-Tac-Toe game board in the middle, including the current state of the game (as shown in Figure 1).

✓ • As soon as the client launches the program, the player should see an empty board, with a text on top of the board showing "Finding Player" before the server finds a match. Hence, you may assume that a player is ready to play as soon as they launch the program.

✓ • After a game starts, a text should be shown on top of the board showing the current player's name (the player who has the turn to make a move), with an 'X' or 'O' written beside their name to indicate the symbol that has been assigned to them by the server.

✓ • When a player has a turn, they may click on an empty cell of the Tic-Tac-Toe board, and their assigned symbol should be drawn to that cell to indicate their move. The client should then inform the current move to the server, and then the turn will be switched to the other player.

✓ • When two players start a new game, they will take turns making moves on the Tic-Tac-Toe board, which should be visually reflected on both of their client GUIs.



Rank#1 Tawfiq: Hi Raj!

Rank#50 Raj: Hi Tawfiq!

Figure 1: Client-side Graphical User Interface (GUI). If you do not implement some features (e.g., chat, timeout, ranking), your GUI will appear slightly different, which is acceptable. You may keep the design of the GUI as simple as shown in the figure, as no extra marks will be awarded for a fancy GUI design.

- ✓ For the basic implementation of the system, the client/server should wait indefinitely until a move has been made by a player with the current turn. Only after a move is made by the current player (by clicking on an empty cell), the move information will be passed from the client to the server, so that the turn can be switched to the other player.
- ✓ There should be a chat window (on the right side of the GUI) that enables players to communicate with their opponents in real time. Only the last 10 messages should be displayed in the chat window, where the most recent message should appear at the bottom. You may assume that the length of each message can be at most 20 characters.
- ✓ A 'quit' option should be implemented (e.g., a button), which can be used to exit the system anytime.
- ✓ After a game is complete, the winner's name should be displayed above the board (e.g., "Player 'username' wins!") If the game ends as a draw, the message "Match Drawn" should be displayed instead.
- ✓ After a game is complete, the player should get an option to either 'find' a new match or 'quit' the system. This can be implemented with a simple dialog box.

2.2 Server

On the server side, you will develop the following core functionalities:

- ✓ The server should be able to handle multiple clients (players) that join the system and want to play the game.
- ✓ When a new client (player) joins the system, they will be added to a pool of active players in the system.
- ✓ From the pool of active players, the server should match any two players to start a new game. The server should continue to match players whenever there are active players in the pool. For an odd number of players, the last remaining player should wait until another player joins the system (the server may communicate this information to the corresponding client so that it keeps showing "Finding Player").

- When a game starts, the server may randomly select which player should take the first turn, and assign a symbol to each player (either ‘X’ or ‘O’).
- After a game is finished, if a player agrees to find a new game, they should be moved to the pool of active players again.
- The server should continuously update the game’s state as the players make moves, and communicate the game’s current state to the players after each move.
- The server should also check for a winning condition after each move. When the game is finished (a player wins, or the game ends as a draw), the server should communicate the result to both players.
- If a player quits the system in the middle of a game, their current match will be forfeited and the other player will be declared as the winner.

2.3 Design Considerations

- **Dealing with concurrency:** Regardless of the technology you use, you will have to ensure that access to shared resources is properly handled and that simultaneous actions lead to a reasonable state.
- **Dealing with networked communication:** You must decide when/what messages are sent across the network. You may have to design an exchange protocol that establishes which messages are sent in which situation and the replies that they should generate.
- **Sockets or RMI:** Clients should establish connections to the server using either Java sockets or RMI, based on your chosen implementation approach. If you use RMI, then you have to design your remote interface(s) and servants.
- **GUI implementation:** You can use any tool/API/library you want (e.g., Java2D drawing package[1]).

3 Additional Features (8 marks)

1. **Client-side Timeout (3 marks):** If this feature is implemented, each player should have at most 20 seconds to complete a move (instead of causing the client/server to wait indefinitely) during their turn. If the time runs out before they make a move, the client should randomly choose any empty cell of the board, draw the player’s symbol, and inform this move to the server, so the turn will be switched to the next player. The client-side GUI should also show a count-down timer (starting at 20, then 19, ..., 0) on the top-left part of the GUI.
2. **Fault Tolerance (3 marks):** During a client crash, the game will be paused for 30 seconds. If the client rejoins the system during this time, the game will be resumed. Otherwise, the game will be ended as a draw. During a server crash, all the players should see the message "Server unavailable", and then their clients will exit gracefully after 5 seconds. The server doesn’t need to persist any game data or player ranks in a file (or similar). Therefore, in the event of a server crash or shutdown, it is reasonable to expect that all game states and rankings will be lost along with it.
3. **Player Ranking System (2 marks):** Each player should be assigned a rank # based on the total rating points (non-negative) they have accumulated over their career (assume the server is always running). Each player starts with point 0 and gets +5 for each win, -5 for each loss, and +2 for each draw. If this feature is implemented, the player rank should always be visible in front of their username (e.g., Rank #50 ‘username’) during their turns (e.g., text above the board), and chat messages. Note that, player ranks will only be updated after each game. Hence, when a game is played, the client may show a static rank for each player that was acquired from the server before the game started.

[1]http://docs.oracle.com/javase/tutorial/2d/index.html

4 Report (5 Marks)

You should write a report describing your system and discussing your design choices. Your report should include:

- The problem context in which the assignment has been given
- A brief description of the components of the system
- An overall class design and an interaction diagram
- A critical analysis of the work done followed by the conclusions

Please note that the report is a written document, so do not put only graphs/figures. A report without any descriptive text addressing the problem, architecture, protocols, and analysis of the work done will not be considered valid. The length of the report is not fixed. A good report is auto-consistent and contains all the required information for understanding and evaluating the work done. Given the level of complexity of the assignment, a report in the range of 8 to 10 pages is reasonable. Please note that the length of the report is simply a guideline to help you avoid writing an extremely long or incomplete report.

It is important to put your details (name, surname, student id) in:

- On the first page of the report
- As a header in each of the files of the software project

This will help to avoid any mistakes in locating the assignment and its components on both sides.

5 Submission

You need to submit the following via LMS:

- Your report in PDF format only
- The executable jar files used to run your system's clients/server(s). The executable jar files should run the following way (and must handle the command-line arguments shown below).

  ```
  java -jar server.jar ip port
  java -jar client.jar username server_ip server_port
  ```

- Your source files in a .ZIP or .TAR archive only
- **Note:** Failure to comply with the submission criteria (file type, command-line args for running jars, etc.) will result in a **4 mark** deduction.
- **Late Submission Penalties:** -1 mark for each day

6 Testing Tips

- Conduct rigorous testing to validate the functionality and stability of your distributed Tic-Tac-Toe system across various scenarios. These should include situations where multiple games are running at the same time.
- Pay particular attention to synchronization and concurrency issues, ensuring that the game operates smoothly.
- Check if the players receive accurate and timely game updates and chat messages as expected.

7 Demonstration Schedule and Venue

You are required to provide a demonstration of the working application and will have the opportunity to discuss with the tutors the design and implementation choices made during the demo. You are free to develop your system where you are more comfortable (at home, on one PC, on your laptop, or in the labs...) but keep in mind that the assignment is meant to be a distributed system that works on at least two different machines in order to separate the clients from the server. We will announce the demo date, time, and venue closer to the due date. Each tutor will hold 2-3 demo sessions and you will be required to showcase your system in one of the sessions held by the tutor of the workshop in which you are enrolled. If you need any clarification on the assignment, kindly ask questions during the tutorials or in the Ed forum, so that all students benefit from it. NOTE: The demo is going to be campus-based, and we will announce the demo schedule. **Note that a mark deduction will be applied if you miss a registered demo or ask to reschedule without a valid reason.**

8 Collaboration and Plagiarism

You may discuss this assignment abstractly with your classmates but what gets typed into your program must be individual work, not copied from anyone else. Do **not** share your code and do **not** ask others to give you their programs. Do **not** post your code on the subject's discussion board Ed. The best way to help your friends in this regard is to say a very firm "**no**" if they ask to see your program, pointing out that your "**no**", and their acceptance of that decision are the only way to preserve your friendship. See `https://academicintegrity.unimelb.edu.au` for more information.

Note also that solicitation of solutions via posts to online forums, whether or not there is payment involved, is also an Academic Misconduct. You should not post your code to any public location (e.g., `GitHub`) while the assignment is active or prior to the release of the assignment marks.

If you use any code not written by you, you must attribute that code to the source you got it from (e.g., a book or Stack Exchange).

Plagiarism policy: You are reminded that all submitted project work in this subject should be your own work. **Automated similarity-checking software will be used to compare submissions.** It is the University's policy that cheating by students in any form is not permitted and that work submitted for assessment purposes must be the independent work of the student concerned.

Using git may help you with the verification of authorship later.

Good luck!

"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable!" - Leslie Lamport[2]

[2] `https://amturing.acm.org/award_winners/lamport_1205376.cfm`