Semester II Final Project Report

Networked Tic-TacToe Java Game

Rino Santoso – 2201798736

Programming Languages

BiNus University International

**TABLE OF CONTENTS**

1. Project Specifications: …………………………………………………3
2. Solution Design: ……………………………………………………….4
3. Functions and Classes: …………………………………………………5
4. UML Chart: ……………………………………………………………10
5. Evidence: ……………………………………………………………....11
6. Lessons Learned: ……………………………………………………....14
7. Problem Overcame: ……………………………………………………16
8. References: …………………………………………………………….17

“Networked Tic-TacToe Java Game”

Name: Rino Santoso

ID: 2201798736

**I. Project Specification**

This program is a multiplayer game based on the classic game of Tic-Tac-Toe. The game involves two players playing on a 3 by 3 board. Each player is assigned as either “X” or “O”. The goal of the game is to make a straight or diagonal line of three squares of either Xs or Os.

The main feature of this game is the fact that it is networked based. The players of this game does not need to be playing on the same computer or even network as the players could configure which server and port they choose to connect to.

The purpose of this game is to test the capabilities of a networked connection in a Java program. As a Computer Science student, this project serves as a basis for understanding the way servers, clients, and sockets work in java, as well as how to utilize them when creating an application.

The rules of this game can be summarized as the following:

1. The server starts as Player 1 with a red X for a sign, and the client starts as Player 2 with a blue O for a sign.
2. Player 1 will be given the first turn one out of the nine fields on the board, then that board will be marked by an X.
3. The turn will be passed on to Player 2 and they will have to choose between the 8 remaining fields, after which said field will be marked by an O.
4. The turn is passed back to Player 1 and this will continue until the game ends.
5. The game ends when one player is able to make a straight or diagonal line using their marks or when all the fields on the board is filled.
6. The game ends in a draw when no player has won when all the fields on the board is filled.

**II. Solution Design**

The initial design of this program is based on an early Java assignment I received during the course of this semester. The assignment deals with the utilization of 2D arrays and making a text-based local tic-tac-toe game. Upon receiving info on the final project and its requirements, I made up my mind on making a networked multiplayer game as my Java final project. Initially, I planned on making a multiplayer minesweeper game. However, upon attempting to do the project, I realized that developing a multiplayer minesweeper is quite complex and I was not confident that I could finish it in time given my limited knowledge on Java networking. Finally I decided to make a simpler networked game, the Tic-Tac-Toe.

The core design of this project is based on the utilization of two separate instances of the main game class, the server and the client. The server is designed to run the main logic of the game, such as deciding the state of the board and declaring when the game ends, as well as receiving packets from the client that indicates the client’s move choices as well as sending back packets to the client that indicates the new state of the board and if the game has ended.

There are three different packets implemented in this program, the UpdatePacket, ClientPlayPacket, and GameEndPacket. The UpdatePacket is a packet sent from the server to the client that contains information of the current state of the board as well as the new current player. The ClientPlayPacket is a packet sent from the client to the server that contains information on what move is taken by the client player. This packet is then used to update the current state of the board. Finally, the GameEndPacket is a packet sent from the server to the user that indicates that the game has ended. This packet contains information on which player, if any, wins the game.

**III. Functions and Classes**

**1. Main.java**

This is the main class of the game and is the file in which the game is run from. This class is part of the ‘game’ package and it can create an instance of either a ServerGame() or ClientGame().

*main()*

* Displays a JOptionPane window that prompts the user to input either ‘1’ to create a server or ‘2’ to create a client.
  + If the player inputs ‘1’, it will create a new instance of the ServerGame() class
  + If the player inputs ‘2’, it will create a new instance of the ClientGame() class

**2. Game.java**

Base class for both instances of the game, Server and Client. This class contains the following attributes:

* PORT: the port required to make an instance of the server, set to 22222
* WIDTH and HEIGHT: size of the window frame and GameWindow panel
* FIELD\_WIDTH and FIELD\_HEIGHT: attributes that translates the pixels of the height and width to an int to be used by the frame array
* NOBODY, PLAYER\_ONE, PLAYER\_TWO: integers that indicate the players as well as the sign contained in each index of the frame array (the board)

*Game(int thisPlayer)*

* Constructor method of the Game class.
* Requires one argument, the player number
* Initializes which player is playing on the instance of the game
* Initializes the GameWindow panel and adding the panel to the window frame
* Initializes the fields, a 2D array that simulates a 3x3 board.

*showWinner(int winner)*

* Method to display the result of the game
* Calls the showMessageDialog of the JOptionPane to display either a tie or the winner of the game

*isMyTurn()*

* Returns true if the player running the game is the same as the current player taking the turn, else, return false

*InputReceived(int x, int y), packetRecieved(Object object), close ()*

* Three abstract classes that will be configured later on each instance of the Game class, the ServerGame and ClientGame.

*getFields()*

* Method to get the current state of the fields.

**3. ServerGame.java**

Server class that extends from the Game class. This is the instance of the game that is made when the user chooses to make a server.

*ServerGame()*

* Constructor method of the ServerGame class.
* Calls the argument of the Game class and passes the player assigned to the server instance, Player 1
* Initializes the server socket and socket that handles input and output, as well as the Connection class object that handles the sending and receiving of packets

*InputRecieved(int x, int y)*

* Upon receiving a mouse input, if it is the player’s turn, update the current state of the field.

*packetRecieved(Object object)*

* Upon receiving a packet form the client, updates the field based on the received packets.

*updateField(int x, int y)*

* Updates the field based on the index of the board generated upon converting the x and y coordinates of the mouse pointer when it was clicked.
* Afterwards, sends an update packet to the client based on the current state of the board
* Repaints the board to reflect the current state of the board
* Checks for wins
* If nobody has won the game, check if the game is a draw
* If the game has ended, calls the endGame method

*endGame(int winner)*

* Shows the end result of the game
* Sends a GameEndPacket to the client

*checkWin()*

* Checks if there is any winner based on the current contents of the fields array (the board)

**4. ClientGame.java**

Client class that extends from the Game class. This is the instance of the game that is made when the user chooses to make a client.

*ClientGame()*

* Constructor method of the ServerGame class.
* Calls the argument of the Game class and passes the player assigned to the server instance, Player 2
* Initializes the socket that handles input and output, as well as the Connection class object that handles the sending and receiving of packets

*InputRecieved(int x, int y)*

* Upon receiving a mouse input, if it is the player’s turn, sends a ClientPlayPacket to the server.

*packetRecieved(Object object)*

* Upon receiving a packet form the server, checks what kind of packet it is
* If it is an UpdatePacket, updates the current content of the fields array (board) as well as the current player on the client side
* If it is a GameEndPacket, shows the result of the game based on the information from the packet.
* Repaints the gameWindow

**5. Connection.java**

Implements the Runnable interface, as it is required for the run method which constantly waits for an inputStream of packets.

*Connection(Game game, Socket socket)*

* Initializes the input and output streams as well as starting a new thread based on the instance of the Game object running the connection and the port.

*run()*

* Constantly loops to check for packets being received through the inputStream

*sendPacket(Object object)*

* Sends a packet through the outputStream.
* Requires one argument, which is the packet going to be sent.
* Before sending the object, it resets the previous object. This is done because if the object type being passed through the argument is the same as the one previously processed, outputStream will not recognize it as a different instance, and simply writes the previous object. Calling the reset method effectively erases the history of objects being written, allowing outputStream to recognize each object as a different instance every time this method is called.

**6. Window.java**

A class that extends from JFrame, it is used to create a new window frame to contain what is going to be displayed. In essence, this is a container for the GameWindow panel.

*Window(Game game, String title, int width, int height)*

* Constructor for the window frame
* Initializes the title, game GUI, and dimensions of the frame.

*class Listener extends WindowAdapter*

* A class inside the window frame that detects if the window is being closed.

**7. GameWindow.java**

A class that extends from JPanel, it is inserted into the window. The panel is the intractable part of the window, where painting and rendering of all the items seen in the GUI, such as lines of the board as well as X and O signs, happens.

*GameWindow(Game game)*

* Constructor, initializes the class displayed by this panel as well as a listener for mouse inputs.

*paint(Graphics g)*

* A method used to update what is displayed on the window
* Draws the lines that serves as the grid for the game board
* Will draw the Xs and Os at their appropriate spots

*class Input extends MouseAdapter*

* A class that extends MouseAdapter
* Used to check for mouse events, in this case a mouse press.

**8. UpdatePacket.java**

A class that implements Serializable, a packet is essentially an object containing a bunch of information being sent and received by both the server and client. The UpdatePacket is meant to be sent by the server to the client, containing information on the current state of the board as well as the current active player.

**9. ClientPlayPacket.java**

A class that implements Serializable, it is meant to be sent by the client to the server. It contains information on the x and y position of the mouse when clicked by the client, indicating which field is selected.

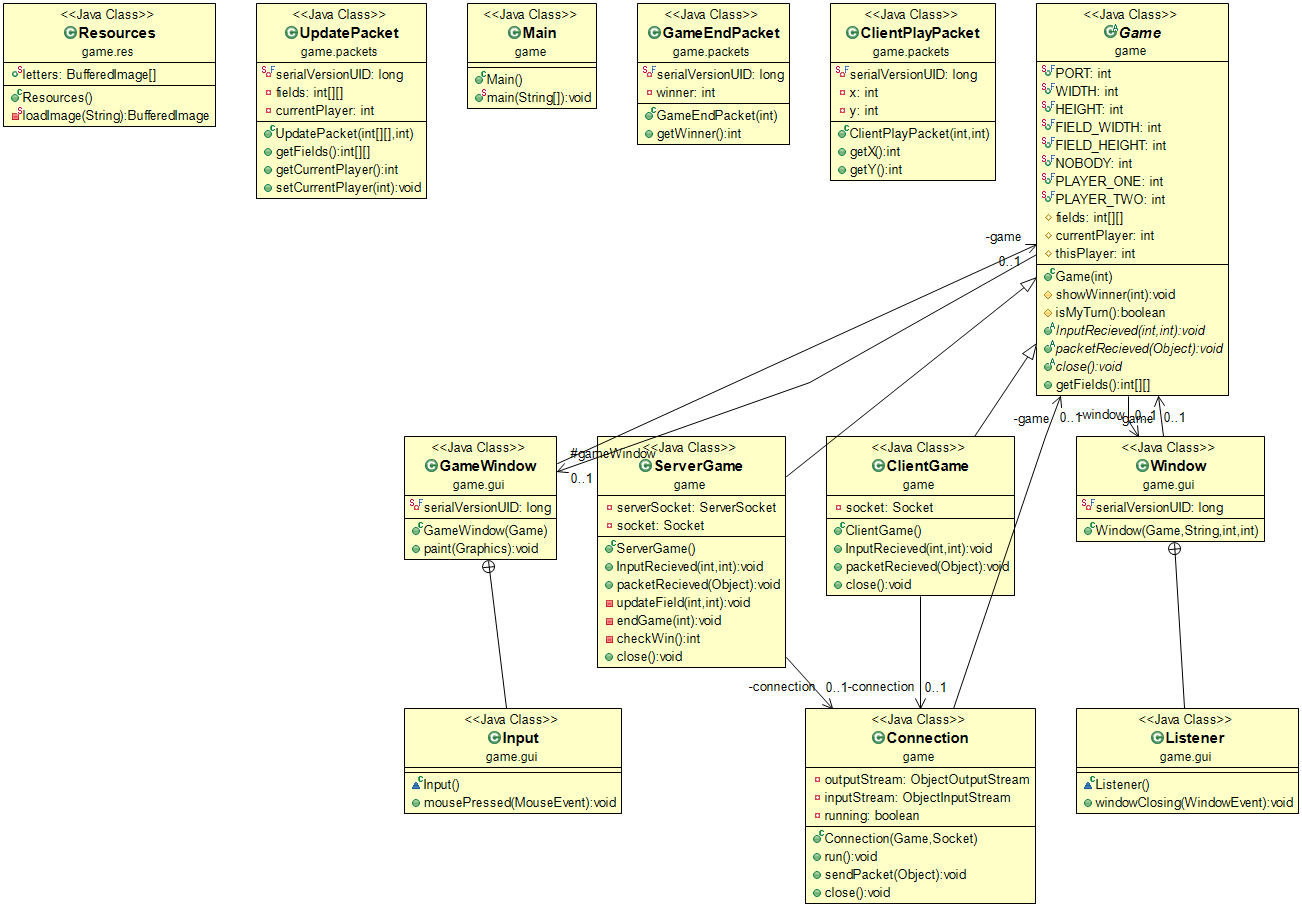
**10. GameEndPacket.java**

A class that implements Serializable, it is meant to be sent by the server to the client. Contains information that indicates that the game has ended, as well as the winner of the game, if any, or if the game ends in a draw.

**11. Resources.java**

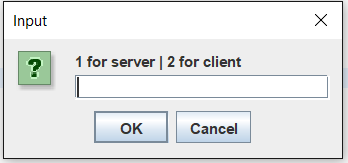
A class to contain buffered images. It is used to access the resources used within the game, such as the images for the X and O mark. It loads images using the ImageIO.read method imported from javax.imageio.ImageIO.

**IV. UML Chart**

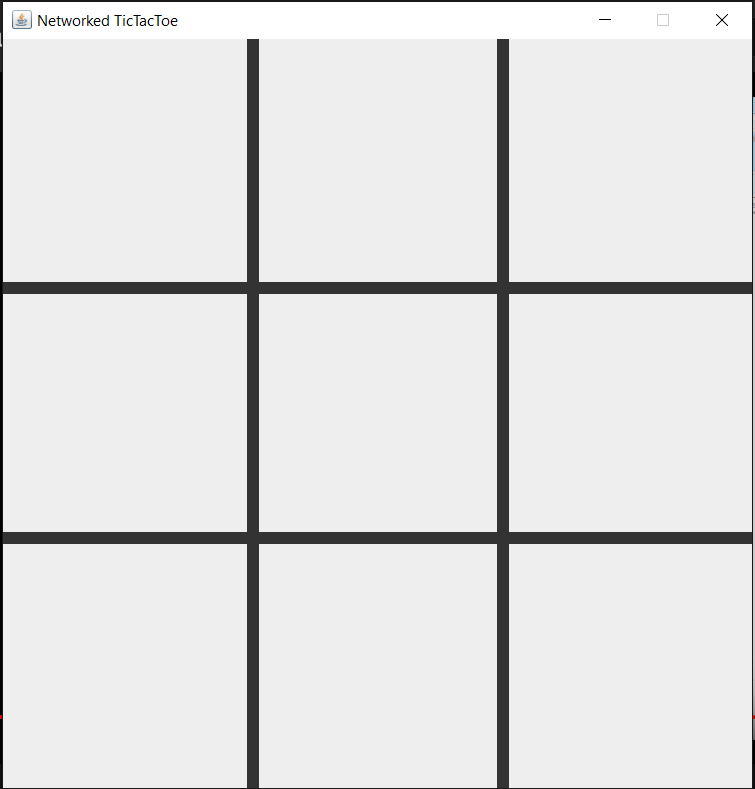
****

**V. Evidence**

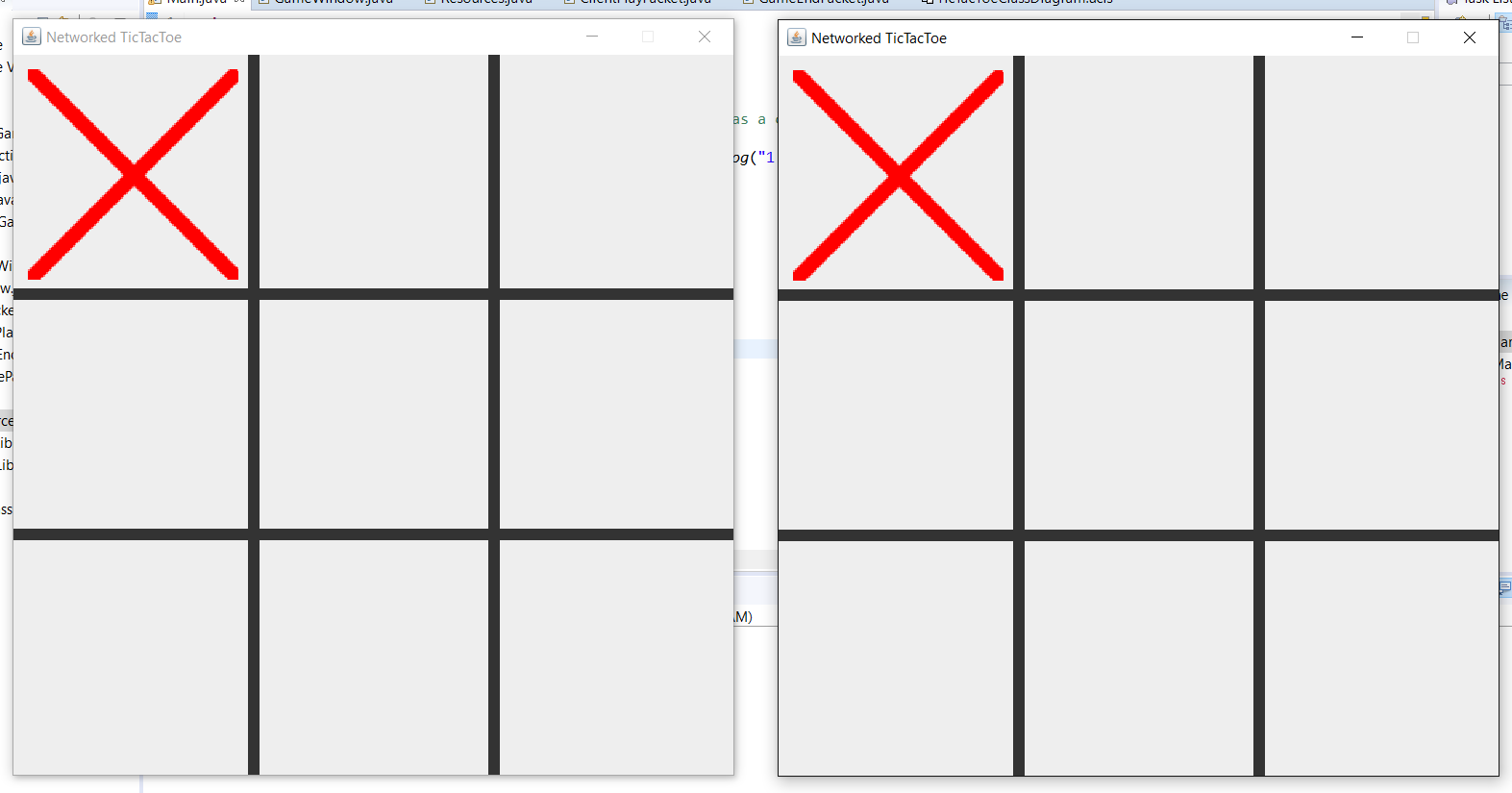
These are the screenshots from the program being run



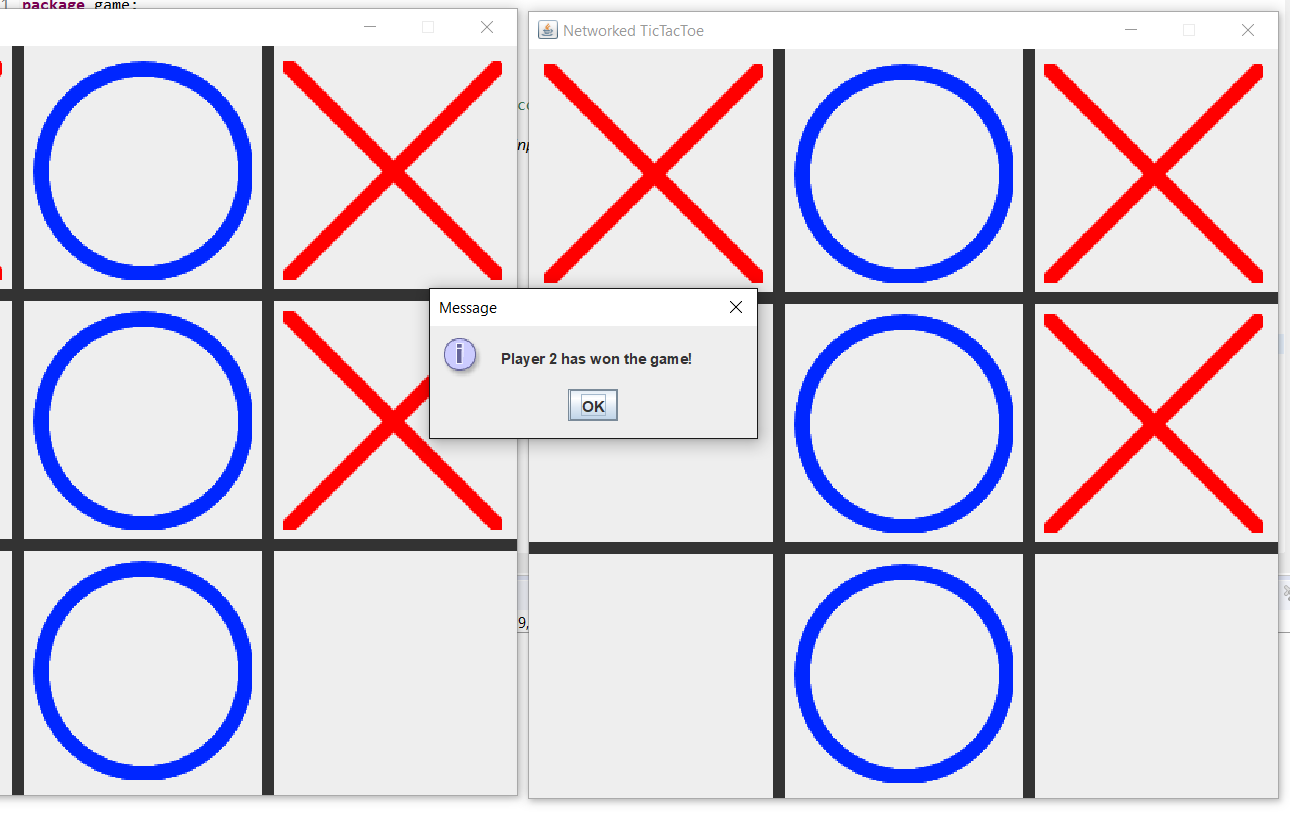
Initial window, prompting the user to choose between making a server and a client



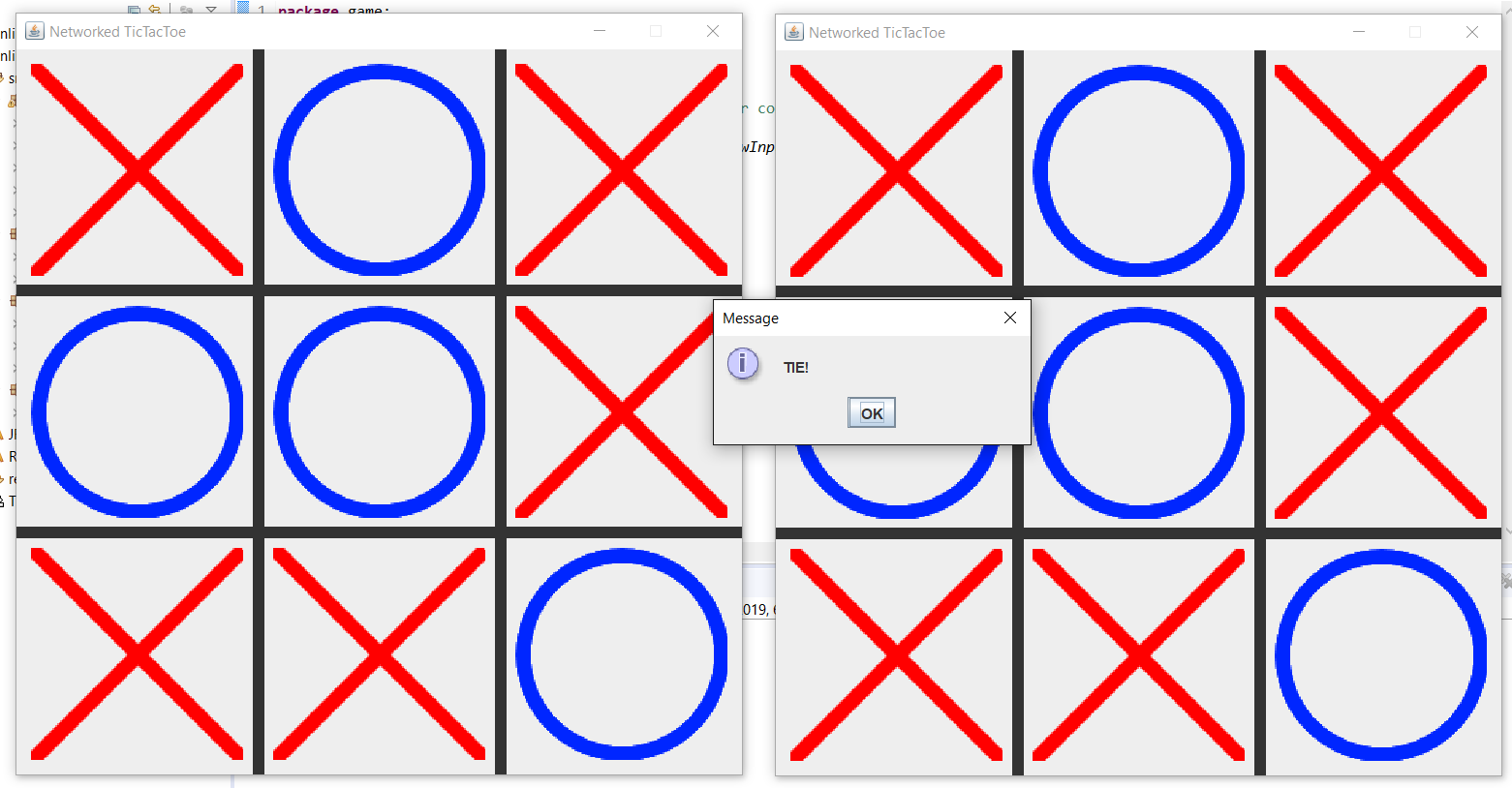
Initial board for both the server and the client



Paints the position of the red X on both instances upon clicking on the server side



Game ending with a win

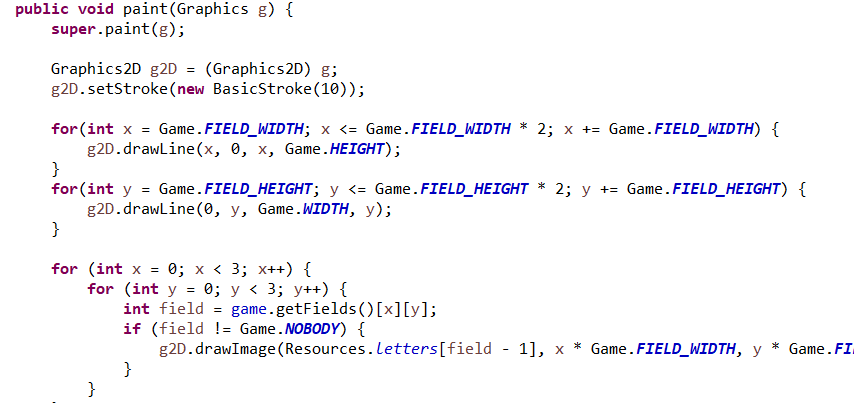


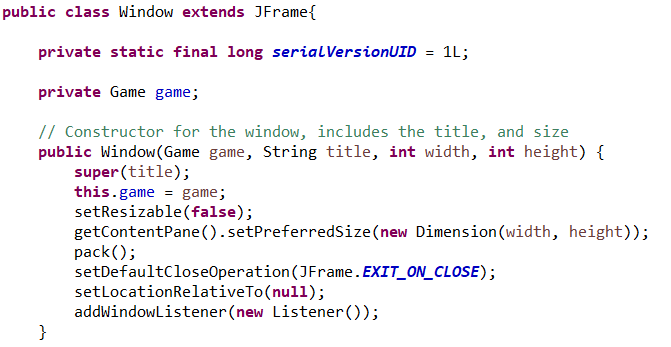
Game ending in a draw

**VI. Lessons Learned**

**1. JFrames and JPanels**

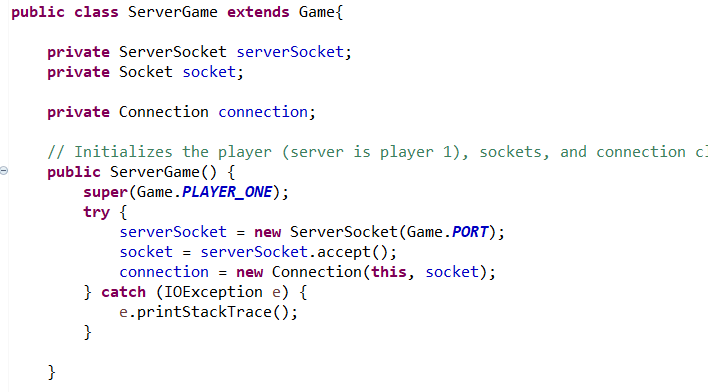
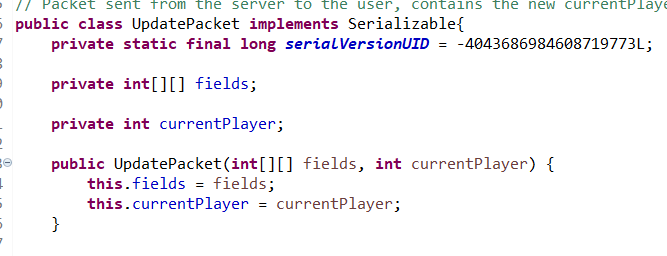
From this project I have learned the utilization of JPanel and JFrames for creating a Graphics User Interface. By using both of these, I am able to create windows to display graphics as well as for the user to interact with. It also allows for a lot of customization as it can paint custom images.





.

**2. Sockets and Packets**

In this project I have also learned how to utilize sockets in order to create servers and clients that can communicate with each other. Furthermore, I also learned a practical way to send and receive data using packet classes to contain the information as objects.  

**VII. Problem Overcame**

One of the greatest problem that I faced during the development of this program is the deadline. This project was meant to be done during the holidays along with other projects from other courses. However, a majority of my holiday was spent on vacation. As a result, I did not have time to maximize my program for this project as well as other projects.

On the technical side, I faced a major problem upon trying to read the image being implemented in this game. The images to mark the Xs and Os were somehow unable to be read using the getResource method of the ImageIO.read, which allows reading of a file using a relative path. As a result, I changed the whole read method to specifically read the file using the full path of the images.

**VIII. References**

The majority of the code for this project was taken from a tutorial series on making a networked TicTacToe game on YouTube by user PROJECT: Joza100**.** The images used to display the X and O marks were taken from another YouTube video by MrWayFarOut. Other additional codes are taken from StackOverflow and GeeksForGeeks

1. Java Multiplayer TicTacToe Tutorial Series: <https://www.youtube.com/watch?v=GPoBPSnmj7Y&list=PL3-YJ4YD-sfkqoVA1zgIUDkcwB1TUFQ3N>
2. Make A Networked Tic-Tac-Toe in Java: <https://www.youtube.com/watch?v=aIaFFPatJjY&list=LLmqtq91b9gyGKTtUN2s4OAQ&index=2&t=3200s>
3. Networked Tic-Tac-Toe resources: <https://www.dropbox.com/s/lc6ktznzusk92y1/TicTacToe.zip?dl=0>
4. <https://stackoverflow.com/questions/19447104/load-image-from-a-filepath-via-bufferedimage>
5. <https://www.geeksforgeeks.org/socket-programming-in-java/>