**CERTIFICATE**



*This is to certify that the project entitled “Tic Tac Toe: A Peer to Peer Multiplayer Networked Game” is a bona fide record of the project work done by Varadraj Galgali, Yash Javalkar and Shubham Mirajkar under my supervision and guidance, in partial fulfillment of the requirements for the Outcome Based Education Paradigm in Computer science and Engineering  from Gogte Institute of Technology, Belagavi for the academic  year 2018-19.*

#### **Prof. Prasad Pujar**

**Dr. Veena Desai**

***Professor & Head***

***Dept. of Computer Science &Engineering***

Place: KLS Gogte Institute of Technology, Belagavi.

**ABSTRACT**

**Problem Definition**

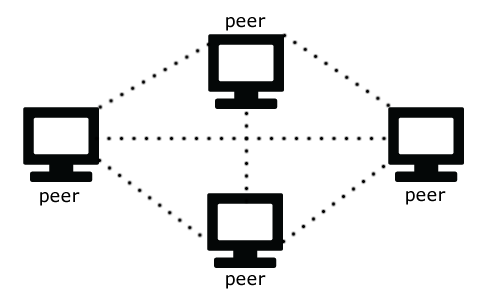
To implement a (P2P) Peer to Peer multiplayer networked game using the networking concepts offered by Java.

**Abstract**

Playing a multiplayer game is always fun. Instead of beating AI-controlled opponents, the player must face strategies created by another human being. This abstract presents the implementation of a multiplayer game played over the network using a non-authoritative peer-to-peer (P2P) approach.

**Peer-to-peer networking**

A simple way to connect players into the same virtual game world is through the peer-to-peer architecture. Although the name might suggest that only two peers (“nodes”) are involved, by definition a peer-to-peer network system is one in which two or more nodes are connected directly to each other without a centralized system orchestrating the connection or information exchange.



On a typical peer-to-peer setup, each peer serves the same function as every other one—that is, they all consume the same data and share whatever data they produce so that others can stay synchronized.

Some of the benefits of peer-to-peer networked games are as follows:

* **Fast data transmission:** Here, the data goes directly to its intended target. In other architectures, the data could go to some centralized node first, then the central node (or the “server”) contacts the other peer, sending the necessary updates.
* **Simpler setup:** You would only need to think about one instance of your game that, generally speaking, handles its own input, sends its input to other connected peers, and handles their output as input for its own system. This can be especially handy in turn-based games, for example, most board games such as *Tic-tac-toe*.
* **More reliability:** Here one peer that goes offline typically won’t affect any of the other peers. However, in the simple case of a two-player game, if one of the players is unable to continue, the game will likely cease to be playable. Imagine, though, that the game in question has dozens or hundreds of connected peers. If a handful of them suddenly lose their Internet connection, the others can continue to play. However, if there is a server that is connecting all the nodes and the server goes down, then none of the other players will know how to talk to each other, and nobody will know what is going on.

**Objective**

The objective of this assignment is to design a Tic Tac Toe Multiplayer Game which has the following features:

* Multiplayer on-line game without a central server using completely peer-to-peer networking
* Interactive multiplayer game involving a maximum of 2 players
* It is a continuous action game, the changes in the game board will be reflected on the players’ choices
* It will be developed for desktops

**Game Design Implementation**

The Tic Tac Toe game can have a maximum of 2 players playing at a time. The players will have limited number of turns. Each player has to prevent the other player from marking three adjacent squares with the same character. The player who succeeds to mark three adjacent squares with his character in a horizontal, vertical or diagonal row wins the game. The game ends as a draw when all the squares are marked and no player has been able to win. The implementation details are:

* The game will be programmed in Java
* The GUI will be developed using Java Swing library
* Socket programming using TCP will be used for communication

**Game Features**

* At the beginning of the game, the host will have to input the correct IP address and port to start a server for the game.
* Once the server is started, the host waits for a player to connect to it. The player can do so by entering the same IP address and port.
* The game can have a maximum of 2 players where each player can be network player or the user themselves.
* The host player is given an ‘X’ mark and the opponent is given the ’O’ mark.
* When either of the player marks three adjacent squares with his character in a horizontal, vertical or diagonal row wins the game.
* When either player fails to do so, the game ends in a tie.

**Networking**

We will use TCP for game networking. TCP stands for “transmission control protocol”. IP stands for “internet protocol”. Together they form the backbone for almost everything we do online, from web browsing to email, it’s all built on top of TCP/IP.

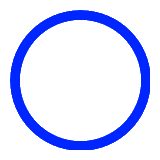
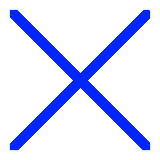
A TCP socket, is a reliable connection based protocol. This means you create a connection between two machines, then you exchange data much like you’re writing to a file on one side, and reading from a file on the other.

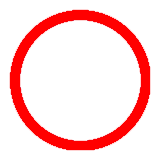
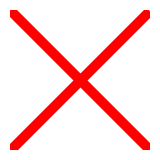
TCP connections are reliable and ordered. All data you send is guaranteed to arrive at the other side and in the order you wrote it. It’s also a stream protocol, so TCP automatically splits your data into packets and sends them over the network for you.

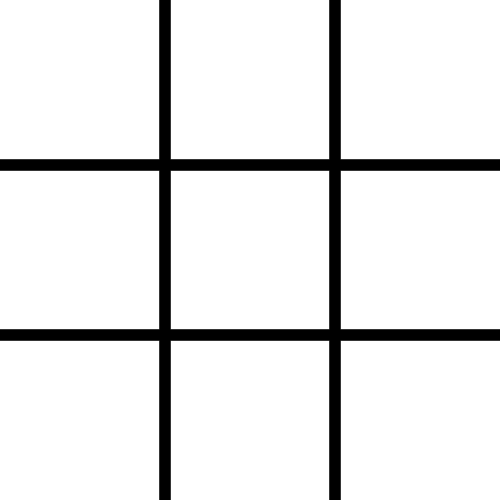
**Game Resources**

We have made use of Microsoft Paint to create a board of 500 X 500 px consisting of nine squares of 160 X 160 px each to implement the Tic Tac Toe game. A border of 10 px separates each row/column.

And the markers are implemented as images of 160 X 160 px each. The available markers are ‘X’ and ‘O’, both available in red and blue colours. The player is given the blue marker while the opponent is given the red marker.



**PROGRAM**

**package** project.tictactoe;

**import** java.awt.BasicStroke;

**import** java.awt.Color;

**import** java.awt.Dimension;

**import** java.awt.Font;

**import** java.awt.Graphics;

**import** java.awt.Graphics2D;

**import** java.awt.RenderingHints;

**import** java.awt.Toolkit;

**import** java.awt.event.MouseEvent;

**import** java.awt.event.MouseListener;

**import** java.awt.image.BufferedImage;

**import** java.io.DataInputStream;

**import** java.io.DataOutputStream;

**import** java.io.IOException;

**import** java.net.InetAddress;

**import** java.net.ServerSocket;

**import** java.net.Socket;

**import** java.util.Scanner;

**import** javax.imageio.ImageIO;

**import** javax.swing.JFrame;

**import** javax.swing.JPanel;

**public** **class** tictactoe **implements** Runnable {

**private** String ip = "localhost";

**private** **int** port = 22222;

**private** Scanner scanner = **new** Scanner(System.***in***);

**private** JFrame frame;

**private** **final** **int** WIDTH = 506;

**private** **final** **int** HEIGHT = 527;

**private** Thread thread;

**private** Painter painter;

**private** Socket socket;

**private** DataOutputStream dos;

**private** DataInputStream dis;

**private** ServerSocket serverSocket;

**private** BufferedImage board;

**private** BufferedImage redX;

**private** BufferedImage blueX;

**private** BufferedImage redCircle;

**private** BufferedImage blueCircle;

**private** String[] spaces = **new** String[9];

**private** **boolean** yourTurn = **false**;

**private** **boolean** circle = **true**;

**private** **boolean** accepted = **false**;

**private** **boolean** unableToCommunicateWithOpponent = **false**;

**private** **boolean** won = **false**;

**private** **boolean** enemyWon = **false**;

**private** **boolean** tie = **false**;

**private** **int** lengthOfSpace = 160;

**private** **int** errors = 0;

**private** **int** firstSpot = -1;

**private** **int** secondSpot = -1;

**private** Font font = **new** Font("Verdana", Font.***BOLD***, 32);

**private** Font smallerFont = **new** Font("Verdana", Font.***BOLD***, 20);

**private** Font largerFont = **new** Font("Verdana", Font.***BOLD***, 50);

**private** String waitingString = "Waiting for another player";

**private** String unableToCommunicateWithOpponentString = "Unable to communicate with opponent.";

**private** String wonString = "You won!";

**private** String enemyWonString = "Opponent won!";

**private** String tieString = "Game ended in a tie.";

**private** **int**[][] wins = **new** **int**[][] { { 0, 1, 2 }, { 3, 4, 5 }, { 6, 7, 8 }, { 0, 3, 6 }, { 1, 4, 7 }, { 2, 5, 8 }, { 0, 4, 8 }, { 2, 4, 6 } };

/\*\*

\* <pre>

\* 0, 1, 2

\* 3, 4, 5

\* 6, 7, 8

\* </pre>

\*/

**public** tictactoe() {

System.***out***.println("Please input the IP: ");

ip = scanner.nextLine();

System.***out***.println("Please input the port: ");

port = scanner.nextInt();

**while** (port < 1 || port > 65535) {

System.***out***.println("The port you entered was invalid, please input another port: ");

port = scanner.nextInt();

}

loadImages();

painter = **new** Painter();

painter.setPreferredSize(**new** Dimension(WIDTH, HEIGHT));

**if** (!connect()) initializeServer();

frame = **new** JFrame();

frame.setTitle("Tic-Tac-Toe");

frame.setContentPane(painter);

frame.setSize(WIDTH, HEIGHT);

frame.setLocationRelativeTo(**null**);

frame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

frame.setResizable(**false**);

frame.setVisible(**true**);

thread = **new** Thread(**this**, "TicTacToe");

thread.start();

}

**public** **void** run() {

**while** (**true**) {

tick();

painter.repaint();

**if** (!circle && !accepted) {

listenForServerRequest();

}

}

}

**private** **void** render(Graphics g) {

g.drawImage(board, 0, 0, **null**);

**if** (unableToCommunicateWithOpponent) {

g.setColor(Color.***RED***);

g.setFont(smallerFont);

Graphics2D g2 = (Graphics2D) g;

g2.setRenderingHint(RenderingHints.***KEY\_TEXT\_ANTIALIASING***, RenderingHints.***VALUE\_TEXT\_ANTIALIAS\_ON***);

**int** stringWidth = g2.getFontMetrics().stringWidth(unableToCommunicateWithOpponentString);

g.drawString(unableToCommunicateWithOpponentString, WIDTH / 2 - stringWidth / 2, HEIGHT / 2);

**return**;

}

**if** (accepted) {

**for** (**int** i = 0; i < spaces.length; i++) {

**if** (spaces[i] != **null**) {

**if** (spaces[i].equals("X")) {

**if** (circle) {

g.drawImage(redX, (i % 3) \* lengthOfSpace + 10 \* (i % 3), (**int**) (i / 3) \* lengthOfSpace + 10 \* (**int**) (i / 3), **null**);

} **else** {

g.drawImage(blueX, (i % 3) \* lengthOfSpace + 10 \* (i % 3), (**int**) (i / 3) \* lengthOfSpace + 10 \* (**int**) (i / 3), **null**);

}

} **else** **if** (spaces[i].equals("O")) {

**if** (circle) {

g.drawImage(blueCircle, (i % 3) \* lengthOfSpace + 10 \* (i % 3), (**int**) (i / 3) \* lengthOfSpace + 10 \* (**int**) (i / 3), **null**);

} **else** {

g.drawImage(redCircle, (i % 3) \* lengthOfSpace + 10 \* (i % 3), (**int**) (i / 3) \* lengthOfSpace + 10 \* (**int**) (i / 3), **null**);

}

}

}

}

**if** (won || enemyWon) {

Graphics2D g2 = (Graphics2D) g;

g2.setStroke(**new** BasicStroke(10));

g.setColor(Color.***BLACK***);

g.drawLine(firstSpot % 3 \* lengthOfSpace + 10 \* firstSpot % 3 + lengthOfSpace / 2, (**int**) (firstSpot / 3) \* lengthOfSpace + 10 \* (**int**) (firstSpot / 3) + lengthOfSpace / 2, secondSpot % 3 \* lengthOfSpace + 10 \* secondSpot % 3 + lengthOfSpace / 2, (**int**) (secondSpot / 3) \* lengthOfSpace + 10 \* (**int**) (secondSpot / 3) + lengthOfSpace / 2);

g.setColor(Color.***RED***);

g.setFont(largerFont);

**if** (won) {

**int** stringWidth = g2.getFontMetrics().stringWidth(wonString);

g.drawString(wonString, WIDTH / 2 - stringWidth / 2, HEIGHT / 2);

} **else** **if** (enemyWon) {

**int** stringWidth = g2.getFontMetrics().stringWidth(enemyWonString);

g.drawString(enemyWonString, WIDTH / 2 - stringWidth / 2, HEIGHT / 2);

}

}

**if** (tie) {

Graphics2D g2 = (Graphics2D) g;

g.setColor(Color.***BLACK***);

g.setFont(largerFont);

**int** stringWidth = g2.getFontMetrics().stringWidth(tieString);

g.drawString(tieString, WIDTH / 2 - stringWidth / 2, HEIGHT / 2);

}

} **else** {

g.setColor(Color.***RED***);

g.setFont(font);

Graphics2D g2 = (Graphics2D) g;

g2.setRenderingHint(RenderingHints.***KEY\_TEXT\_ANTIALIASING***, RenderingHints.***VALUE\_TEXT\_ANTIALIAS\_ON***);

**int** stringWidth = g2.getFontMetrics().stringWidth(waitingString);

g.drawString(waitingString, WIDTH / 2 - stringWidth / 2, HEIGHT / 2);

}

}

**private** **void** tick() {

**if** (errors >= 10) unableToCommunicateWithOpponent = **true**;

**if** (!yourTurn && !unableToCommunicateWithOpponent) {

**try** {

**int** space = dis.readInt();

**if** (circle) spaces[space] = "X";

**else** spaces[space] = "O";

checkForEnemyWin();

checkForTie();

yourTurn = **true**;

} **catch** (IOException e) {

e.printStackTrace();

errors++;

}

}

}

**private** **void** checkForWin() {

**for** (**int** i = 0; i < wins.length; i++) {

**if** (circle) {

**if** (spaces[wins[i][0]] == "O" && spaces[wins[i][1]] == "O" && spaces[wins[i][2]] == "O") {

firstSpot = wins[i][0];

secondSpot = wins[i][2];

won = **true**;

}

} **else** {

**if** (spaces[wins[i][0]] == "X" && spaces[wins[i][1]] == "X" && spaces[wins[i][2]] == "X") {

firstSpot = wins[i][0];

secondSpot = wins[i][2];

won = **true**;

}

}

}

}

**private** **void** checkForEnemyWin() {

**for** (**int** i = 0; i < wins.length; i++) {

**if** (circle) {

**if** (spaces[wins[i][0]] == "X" && spaces[wins[i][1]] == "X" && spaces[wins[i][2]] == "X") {

firstSpot = wins[i][0];

secondSpot = wins[i][2];

enemyWon = **true**;

}

} **else** {

**if** (spaces[wins[i][0]] == "O" && spaces[wins[i][1]] == "O" && spaces[wins[i][2]] == "O") {

firstSpot = wins[i][0];

secondSpot = wins[i][2];

enemyWon = **true**;

}

}

}

}

**private** **void** checkForTie() {

**for** (**int** i = 0; i < spaces.length; i++) {

**if** (spaces[i] == **null**) {

**return**;

}

}

tie = **true**;

}

**private** **void** listenForServerRequest() {

Socket socket = **null**;

**try** {

socket = serverSocket.accept();

dos = **new** DataOutputStream(socket.getOutputStream());

dis = **new** DataInputStream(socket.getInputStream());

accepted = **true**;

System.***out***.println("CLIENT HAS REQUESTED TO JOIN, AND WE HAVE ACCEPTED");

} **catch** (IOException e) {

e.printStackTrace();

}

}

**private** **boolean** connect() {

**try** {

socket = **new** Socket(ip, port);

dos = **new** DataOutputStream(socket.getOutputStream());

dis = **new** DataInputStream(socket.getInputStream());

accepted = **true**;

} **catch** (IOException e) {

System.***out***.println("Unable to connect to the address: " + ip + ":" + port + " | Starting a server");

**return** **false**;

}

System.***out***.println("Successfully connected to the server.");

**return** **true**;

}

**private** **void** initializeServer() {

**try** {

serverSocket = **new** ServerSocket(port, 8, InetAddress.*getByName*(ip));

} **catch** (Exception e) {

e.printStackTrace();

}

yourTurn = **true**;

circle = **false**;

}

**private** **void** loadImages() {

**try** {

board = ImageIO.*read*(getClass().getResourceAsStream("/board.png"));

redX = ImageIO.*read*(getClass().getResourceAsStream("/redX.png"));

redCircle = ImageIO.*read*(getClass().getResourceAsStream("/redCircle.png"));

blueX = ImageIO.*read*(getClass().getResourceAsStream("/blueX.png"));

blueCircle = ImageIO.*read*(getClass().getResourceAsStream("/blueCircle.png"));

} **catch** (IOException e) {

e.printStackTrace();

}

}

@SuppressWarnings("unused")

**public** **static** **void** main(String[] args) {

tictactoe ticTacToe = **new** tictactoe();

}

**private** **class** Painter **extends** JPanel **implements** MouseListener {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**public** Painter() {

setFocusable(**true**);

requestFocus();

setBackground(Color.***WHITE***);

addMouseListener(**this**);

}

@Override

**public** **void** paintComponent(Graphics g) {

**super**.paintComponent(g);

render(g);

}

@Override

**public** **void** mouseClicked(MouseEvent e) {

**if** (accepted) {

**if** (yourTurn && !unableToCommunicateWithOpponent && !won && !enemyWon) {

**int** x = e.getX() / lengthOfSpace;

**int** y = e.getY() / lengthOfSpace;

y \*= 3;

**int** position = x + y;

**if** (spaces[position] == **null**) {

**if** (!circle) spaces[position] = "X";

**else** spaces[position] = "O";

yourTurn = **false**;

repaint();

Toolkit.*getDefaultToolkit*().sync();

**try** {

dos.writeInt(position);

dos.flush();

} **catch** (IOException e1) {

errors++;

e1.printStackTrace();

}

System.***out***.println("DATA WAS SENT");

checkForWin();

checkForTie();

}

}

}

}

@Override

**public** **void** mousePressed(MouseEvent e) {

}

@Override

**public** **void** mouseReleased(MouseEvent e) {

}

@Override

**public** **void** mouseEntered(MouseEvent e) {

}

@Override

**public** **void** mouseExited(MouseEvent e) {

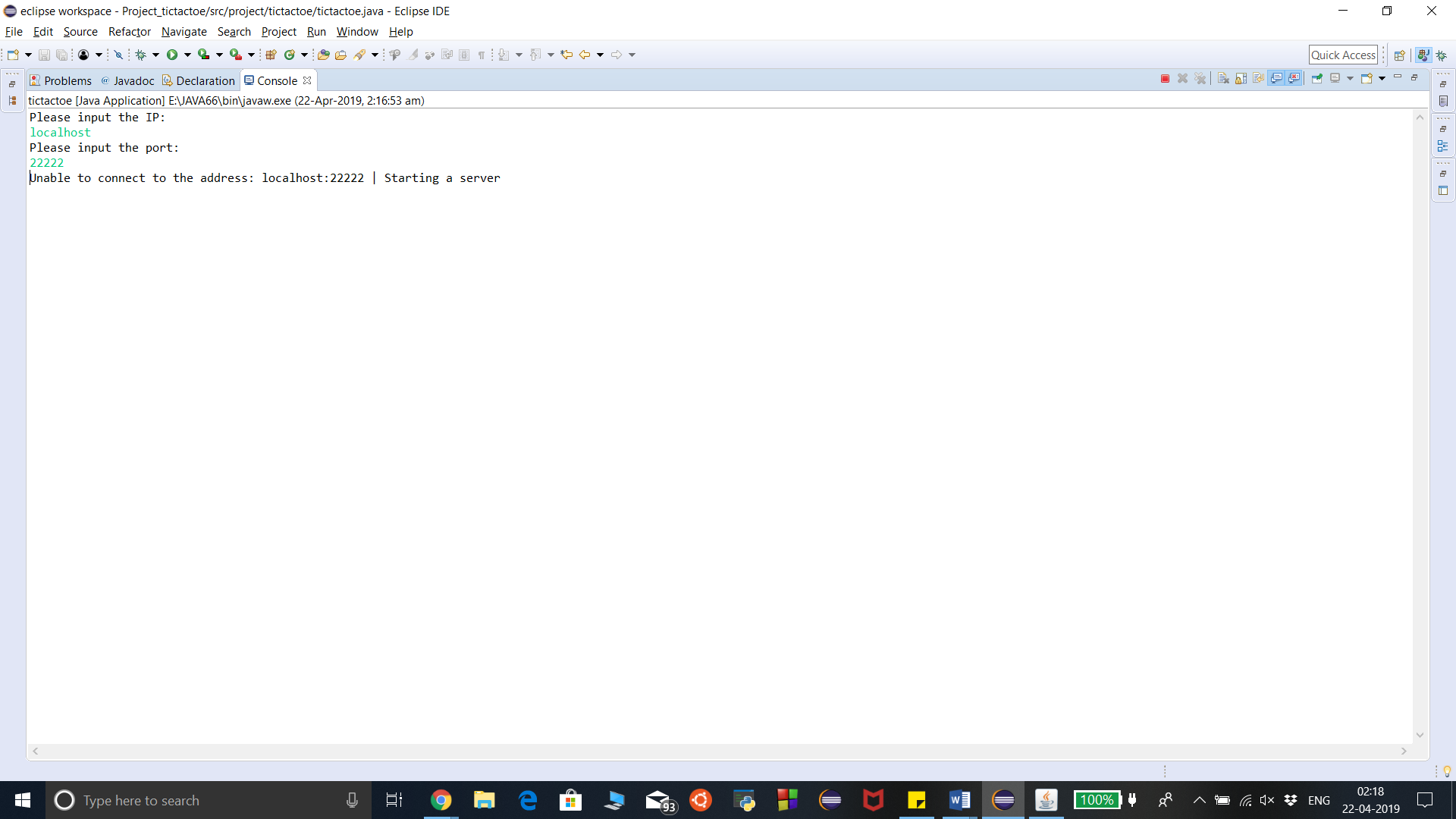
}

}

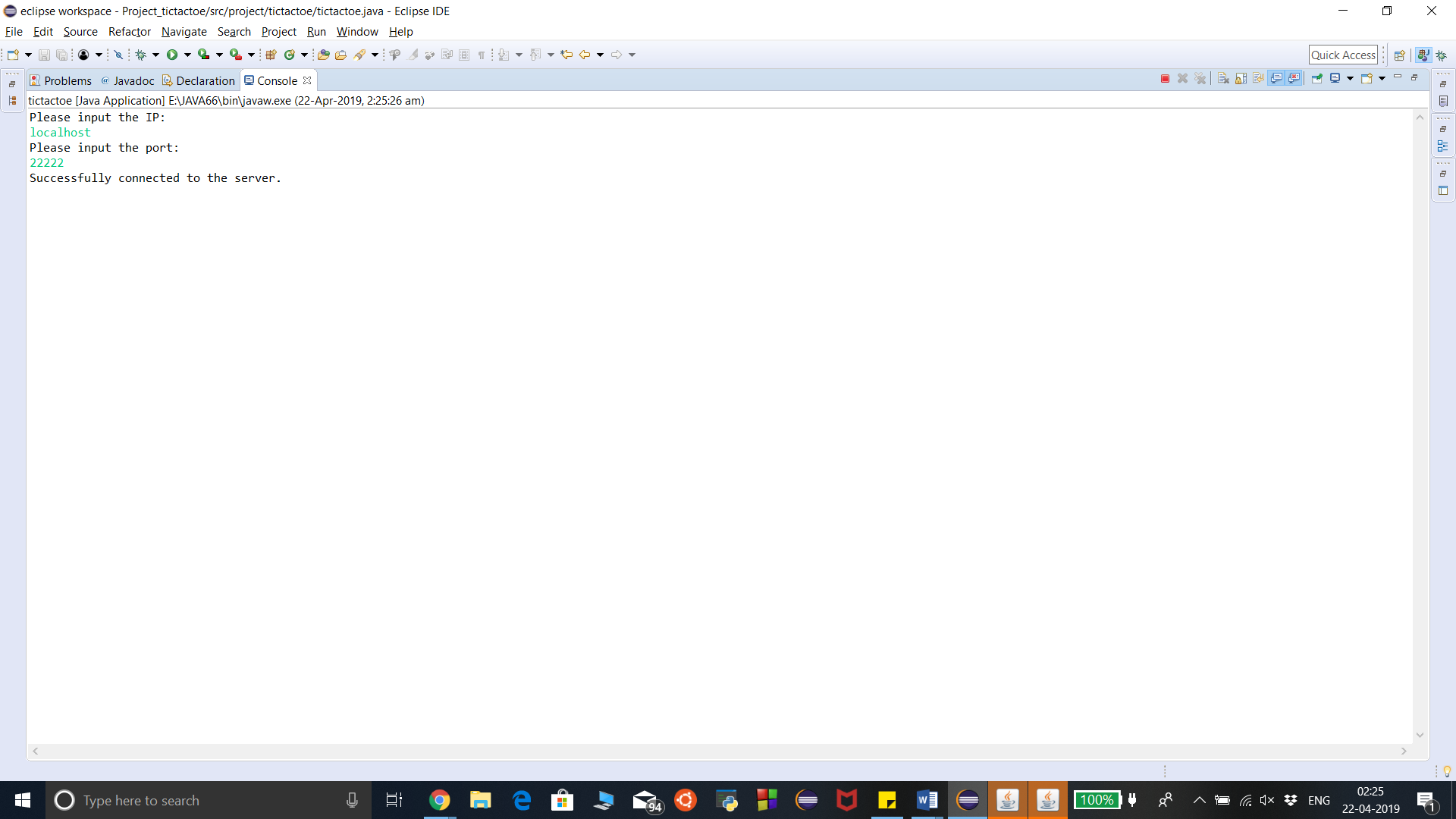
}

**OUTPUT**

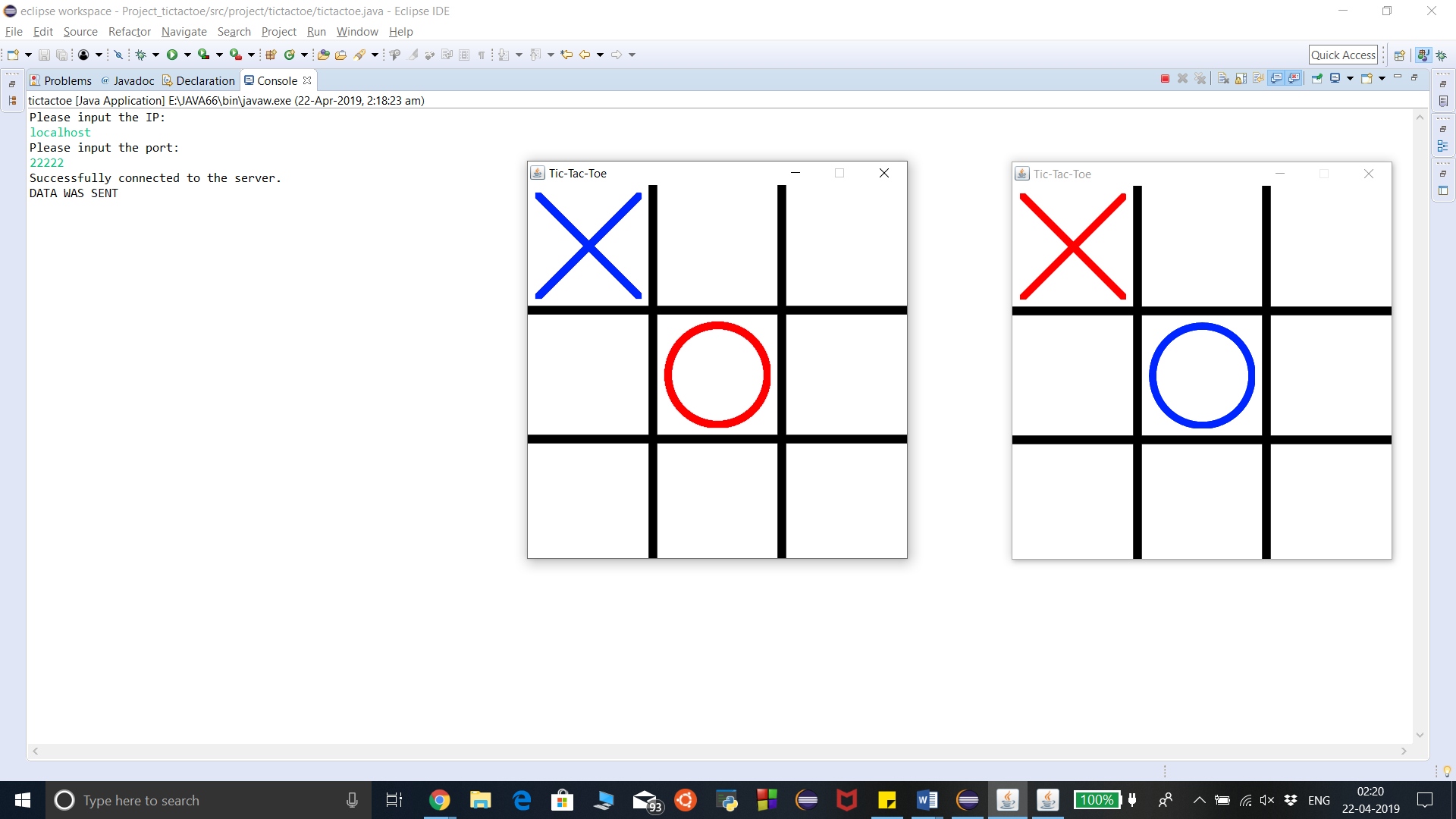
Starting a server



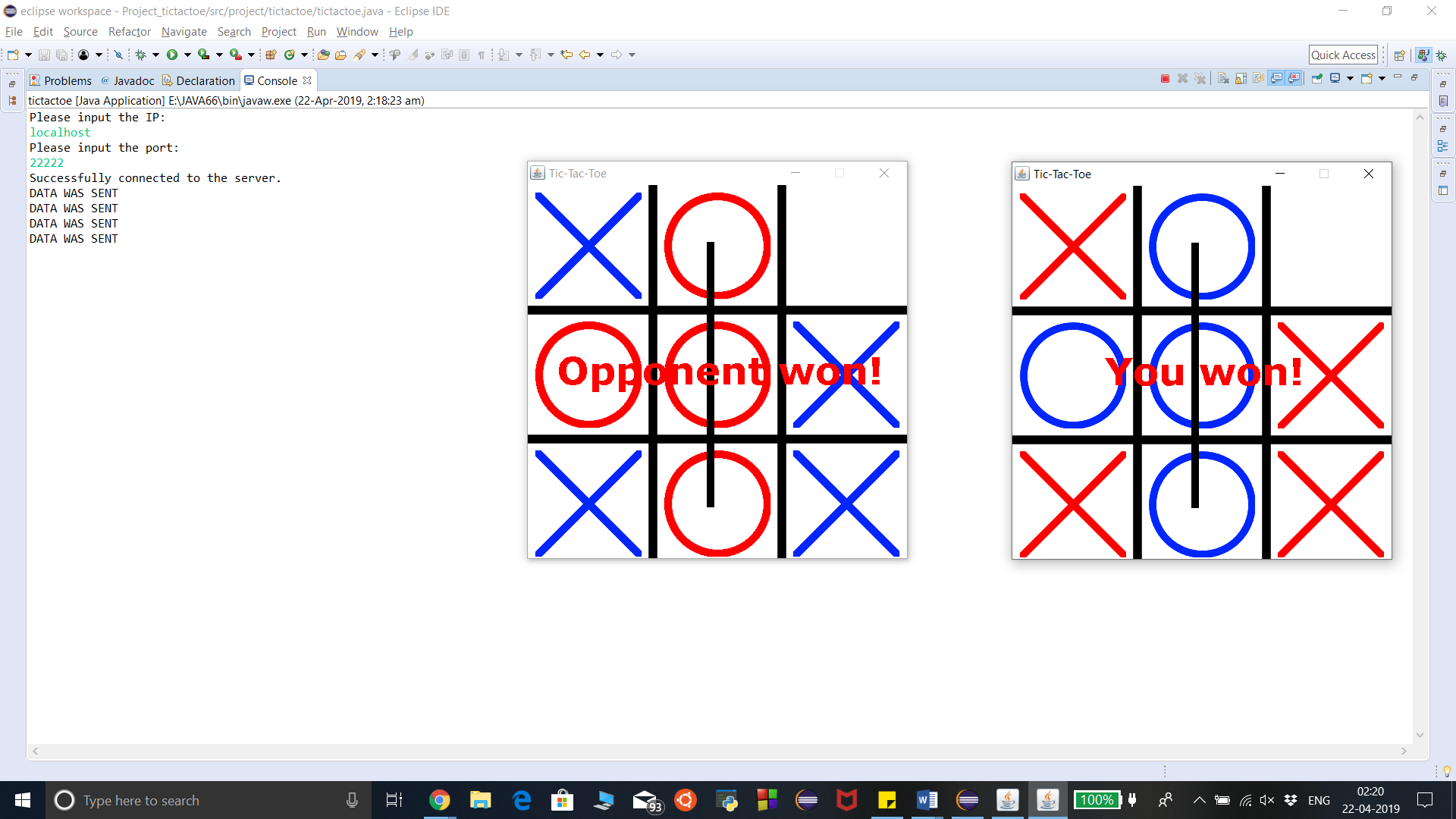
Connecting to a server



Data being transferred



Game ending as a win/loss



Game ending as a tie

