

**PROJECT REPORT ON**  
**TIC-TAC-TOE USING TCP/IP SOCKET PROGRAMMING IN JAVA**

Submitted in partial fulfilment of the requirements for the award of the degree of

**BACHELOR OF TECHNOLOGY**

**Submitted by**

123003207- **RAMAPAVAN KOPPARAPU - CSE**



**Under the Guidance of**

**Prof. Dindayal Mahto**

**School of Computing**

**SASTRA DEEMED TO BE UNIVERSITY**

(A University established under section 3 of the UGC Act, 1956)

Tirumalaisamudram

Thanjavur - 613401

January(2021)

**SHANMUGHA**  
**ARTS, SCIENCE, TECHNOLOGY & RESEARCH ACADEMY**  
**(SASTRA DEEMED TO BE UNIVERSITY)**  
**(A University Established under section 3 of the UGC Act, 1956)**  
**TIRUMALAISAMUDRAM, THANJAVUR – 613401**



**BONAFIDE CERTIFICATE**

Certified that this project work entitled “**Tic-Tac-Toe Using TCP/IP Socket Programming in Java**” submitted to the Shanmugha Arts, Science, Technology & Research Academy (SASTRA Deemed to be University), Tirumalaisamudram - 613401 by RAMAPAVAN KOPPARAPU (123003207), CSE in partial fulfillment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY** in their respective programme. This work is an original and independent work carried out under my guidance, during the period August 2021 - January 2021.

**Prof. Dindayal Mahto**

**ASSOCIATE DEAN  
SCHOOL OF COMPUTING**

Submitted for Project Viva Voce held on – 11<sup>th</sup> February 2022

**Examiner – I**

**Examiner – II**

## TABLE OF CONTENTS

S.NO	TOPIC	Page No.
1	ABBREVIATIONS	5
2	ACKNOWLEDGEMENTS	6
3	ABSTRACT	7
4	CHAPTER 1 INTRODUCTION	8
5	CHAPTER 2 RELATED WORKS	10
6	CHAPTER 3 PROPOSED FRAME WORK	11
7	CHAPTER 4 SOURCE CODE	13
8	CHAPTER 5 RESULTS	27
9	CHAPTER 7 CONCLUSION AND FUTURE WORKS	39
10	CHAPTER 8 REFERENCES	40

## LIST OF FIGURES

<b>SNO</b>	<b>TOPIC</b>	<b>PAGE NO</b>
1	OSI Model vs TCP/IP model	8
2	Client-server network architecture	9
3	Server Side Programming	12
4	Server started	27
5	Client 1 joined	27
6	Client 2 joined	28
7	Server stops accepting clients after 2 join	28
8	Player 1 wins (“X”)	29
9	Player 2 wins (“O”)	31
10	Match Draw	34

## **ABBREVIATIONS**

TCP	Transmission Control Protocol
IP	Internet Protocol
UDP	User Datagram Protocol
GUI	Graphical User Interface
LAN	Local Area Network
IPConfig	Internet Protocol Configuration

## ACKNOWLEDGEMENTS

First of all, I would like thank God Almighty for his endless blessings.

I would like to express my sincere gratitude to **Dr S. Vaidyasubramaniam, Vice-Chancellor** for his encouragement during the span of my academic life at SASTRA Deemed University.

I would forever remain grateful and I would like to thank **Dr A.Umamakeswri, Dean, School of Computing and R. Chandramouli, Registrar** for their overwhelming support provided during my course span in SASTRA Deemed University.

I am extremely grateful to **Dr. Shankar Sriram, Associate Dean, School of Computing** for his constant support, motivation and academic help extended for the past three years of my life in School of Computing.

I would specially thank and express my gratitude to **N.Dindayal Mahto , Associate Professor, School of Computing** for providing me an opportunity to do this project and for her guidance and support to successfully complete the project.

I also thank all the Teaching and Non-teaching faculty, and all other people who have directly or indirectly help me through their support, encouragement and all other assistance extended for completion of my project and for successful completion of all courses during my academic life at SASTRA Deemed University.

Finally, I thank my parents and all others who help me acquire this interest in project and aided me in completing it within the deadline without much struggle.

## ABSTRACT

**Socket programming** connects 2 nodes (socket being the plug for connecting two devices in a network. Ex: **client and server**) on a network to communicate with each other. One socket listens to the Ip address and the port number, while the other extends to form a connection (Client and server have established a connection- inter process connection). Most programming languages like java have bindings to the socket as API. They are connection between low level Application to Transport layer API's.

Socket is first created, associated with a type of **transport layer** (TCP, UDP, etc), associated with an IP address and port, and then used to either listen to connections, or send or receive data, files.

**TCP/IP** protocol is the communication protocol used to connect devices on internet. It specifies how data is communicated over internet by providing end-to-end communication which identifies how it should be broken as packets, segments, addressed, routed and received at destination.

Online multiplayer games have become a fab among younger generations, being a good entertainment and a way to socialize with people all over the world (using internet- computer network). Online gaming has seen a boost especially in these tough covid times. People have started playing multiplayer games to increase concentration level, relax/reduce stress and to socialize.

Initially online multiplayer were LAN connected games, the first game being "**PONG**". The moves played by one player were sent to the local server and they are written to the other player, cycle repeats. This interconnection was done by **SOCKET PROGRAMMING** which used TCP/IP protocol in the transport protocol.

Tic-Tac-Toe is the **strategical turn-based game**, where logic is applied to win. It is a 2 player 3x3 grid with players filling the blocks with X's or O's and the first player to place three of their marks consecutively in a row, column or diagonal wins. We are using socket programming to communicate between the server and the 2 clients (players) who are distant/in remote areas.

**Key words:** Socket programming, Client and server, transport layer, TCP/IP, strategical turn-based game

# CHAPTER 1

## INTRODUCTION

In this **project** we are illustrating a multiplayer game using client-server architecture and by using socket programming, by using TCP/IP protocol to establish a connection between client and server.

### 1.1 COMPUTER NETWORKING

Computer network is defined as set of interconnected computers that exchange resources amongst themselves via a network such as the **Internet**. There are several types of networks-based type of computers and scale of the networks. Multiplayer games, chat facility have rose to fame in the past decade. Computer networking provides multiple computers to communicate effortlessly with each other without having to be close by (connected via cables/wires), making it convenient, ease of accessibility and is flexible.

### 1.2 OSI MODEL (Open System Interconnection)

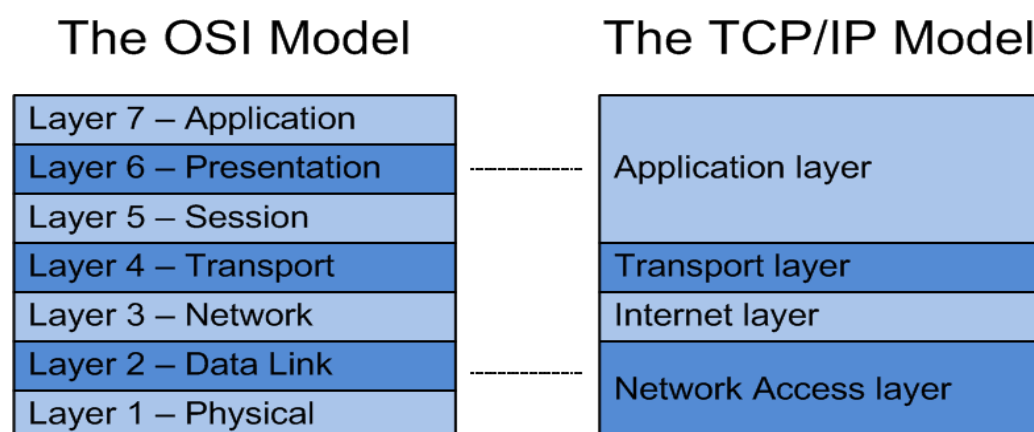
The OSI model is a conceptual model that describes how information from a software application in one computer moves through physical medium to another computer.

This model is widely used as it is less complex and provides a reliable data transfer service.

The OSI model splits into seven layers: Physical, Data Link, Network, Transport, Session, Presentation and Application layer.

Each lower layer adds its services to the higher layer to provide services to control communications and to run the application. OSI model gives guidelines on how to communication needs to be done. Each layer is independent and has a particular task.

The 7 layer OSI model is converted into 4 layers TCP/IP protocol with layers 1,2 layer joining into Network Access layer, layer 3 known as Internet layer, 4 remaining as it is, layer 5,6 and 7 converging into one layer called Application layer.



### 1.3 TRANSMISSION CONTROL PROTOCOL (TCP)



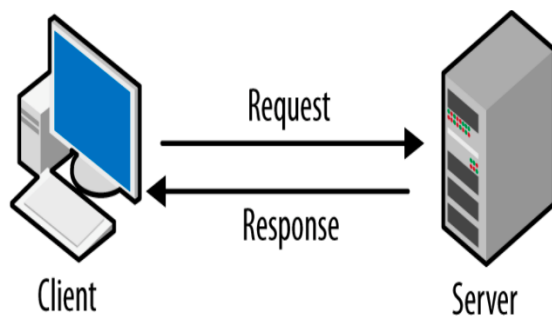
It is a transport layer protocol that facilitates the transmission of packets from source to destination. It establishes connection prior to communication that occurs between the devices connected via internet. It's main function is to take the data from application layer, divides the packets and provides numbering to them then transmits these packets to destination.

The connection is established using three-way handshake.

## 1.4 TCP CLIENT AND SERVER

The client server architecture is defined as logical design of software, hardware, and a medium of transmission of data. In this type of architecture the client computer sends a request for data to the server through the internet/local IP address and the server accepts and processes the request for data packets that were requested, delivers back to the client. Multiple clients are able to communicate using the server

In this project we are hosting the server locally on computer and both the clients/players are able to play the game, or players from any region can join and connect using IP address.



## 1.5 SOCKET PROGRAMMING

**Socket programming** connects 2 nodes (socket being the plug for connecting two devices in a network. Ex: **client and server**) on a network to communicate with each other. One socket listens to the Ip address and the port number, while the other extends to form a connection (Client and server have established a connection- inter process connection). Most programming languages like java have bindings to the socket as API. They are connection between low level Application to Transport layer API's.

Socket is first created, associated with a type of **transport layer** (TCP, UDP, etc), and OSI model is used associated with an IP address and port, and then used to either listen to connections, or send or receive data, files.

To make a connection request, the client tries to engage with the server. If no errors, the server accepts the connection.

```
private Socket socket;  
socket = new Socket(ip,26771);
```

## **CHAPTER 2**

### **RELATED FRAMEWORKS**

Kalita, L. et al (Kalita, L ,2014). Socket programming. *International Journal of Computer Science and Information Technologies* has made us realize what is socket programming and how to create basic sockets, connect and transfer data in java.

Li, Y. et, al(Li. Y, 2011 May). Research based on OSI model has described what are the types of OSI layers with brief explanation of each sub layers and basic understanding of the application layer which provides common web application services, and transport layer that facilitates transport of data from source to dest.

Leung, K. et. al(Leung, k2007). An overview of packet reordering in transmission control protocol (TCP) and basic understanding of how packets are created and distributed/transported by numbering them and also understanding what TCP protocol is and why it is better than OSI model.

Xue, M., & Zhu, C. at. el(Xue, M & Zhu, C,2009, May). The socket programming and software design for communication based on client/server. How client server architecture works and how to use socket programming to achieve this using java.

Karamchandani, S. et. al(Karamchandani, S, 2015, January). A simple algorithm for designing an artificial intelligence based Tic Tac Toe game and a generalized understanding as to how tic-tac-toe game works

Joe Todd. et. Al(Joe Todd, April 2020) Social video games to play during the coronavirus quarantine and how to pro, cons of online socialization to stay healthy and be socially active.

Rishabh Mahrsee. et. Al(Rishabh Mahrsee, 2020) Introducing threads in Socket programming in java and estavblsh a connection using socket

## CHAPTER 3

### PROPOSED FRAMEWORK

The Tic-Tac-Toe multiplayer game that uses socket programming (an API used in java programming), uses TCP/IP protocol to establish a connection between client and server. This strategic game is a 2 player game i.e.; 2 clients interact with each other via a server that records/acts as a communication medium between the 2 clients that are either playing in the same desktop/computer(localhost) or different computers which are connected via IP address.

First a connection is established between the 2 clients via the server using TCP/IP protocol and sockets are used to connect them. When the first player makes a move, the action listener records the move/mark and sends it to the server which is then sent/returned back to the second/other player who has already established connection. The received mark/move is written as opponent move. Each player plays a move and the move is recorded and reflected back, this continues until one of them wins or is a draw.

A player wins if the marker is the same in all blocks of the certain full row/column/ cross diagonals or if all the blocks are filled and no player wins then is declared as draw. Score of each player is updated until both players decide to exit the game. A function is used to block a player to record more than one move/mark consecutively.

#### Client

The Client connects to the server using TCP protocol. The client sends a request to connect to the server and if the server accepts the request the connection is implemented and secure. The first player to connect to the server is named as #player1 and the second player as #player2. As soon as both the players connect with the server using socket no more clients are accepted. Player1 gets 'X' mark whereas #player2 gets 'O' mark. The color of the player blocks is blue and that of the opponent is red, and only one move can be made by a player consecutively.

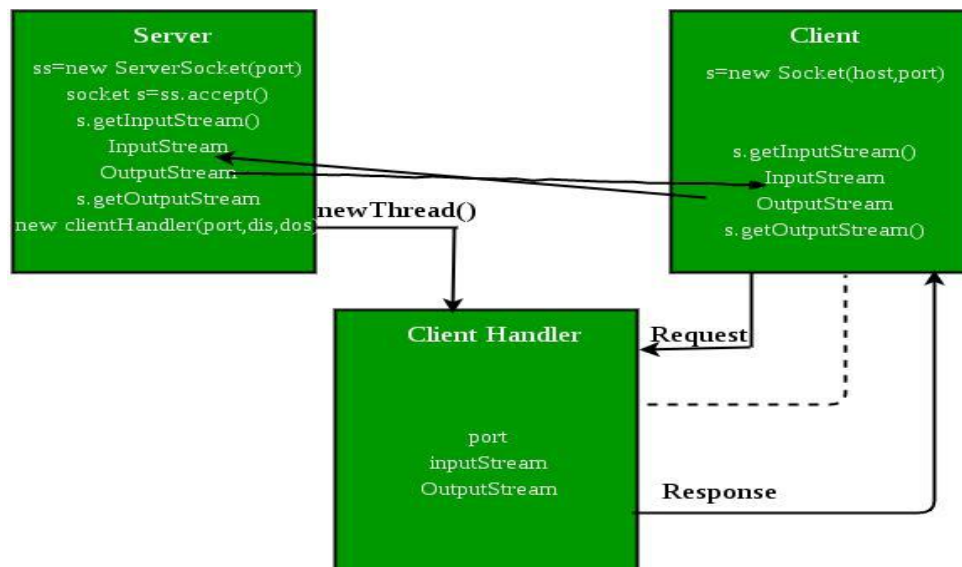
The game continues until any one wins or it's a draw. A restart window pops up asking to restart the game or exit, if any of the clients does not restart the game exits/terminates and server also shuts down.

#### Server

The server is written in java and uses TCP protocol. TCP is scalable, and uses client-server architecture and most used protocol. Port number is fixed for both server and client and cannot be changed. Once both the clients establish a connection data is sent through a loop. The Server does not do much work except sending in player1 move to opponent player, hence much space is not required. Server has only one job that is to send the integer value across the clients. The loop breaks when any client disconnects from server or because of IOException. Using threading the server communicates.

This is a multi client game but since the game is a 2 player game, code is restricted for only 2 clients to connect to the server at once and play the game. Players can play using

localhost or using IP address of other player. Using the command IPCONFIG we can get the IPv4, IPv6 addresses and use them to connect to the server and play, meaning players can play from anywhere with a internet connection (it is vital).



Socket module is imported in python programming language.

1. socket.socket() – Creation of sockets, use this in client and server side to create sockets.

The first step in the program

2. socket.accept() – accepts an incoming connection request from a TCP client. In client side IP address, port number of server, the connection request is received with accept() call serverside.

3. socket.bind() – A server used this to bind itself to a specific IP and port so that it can listen to incoming request on that IP and port.

4. socket.connect() – used in client side to establish a connection to a server

5. socket.send() – used to send data from one socket to another

6. socket.close() – clean way to close the application/socket.

### Tic-Tac-Toe rules

1. The game is played on a grid that's 3 squares by 3 squares. You are “X”, your friend (or the computer) is “O”. Both the players take turns marking the empty squares.

2. The first player to get 3 of their marks in a row (up, down, across, or diagonally) is the winner. When all 9 squares are full, the game is over. If no player has 3 marks in a row, the game ends in a tie.

## CHAPTER 4

### SOURCE CODE

#### 4.1 server source code:

```
import java.util.*;
import java.net.*;
import java.io.*;

public class Server {

    private int playerCount;
    private int player1Val;
    private int player2Val;
    private ServerConnection player1;
    private ServerConnection player2;
    private ServerSocket ss;

    Scanner scan = new Scanner(System.in);
    //Creates a Server socket

    public Server() {

        playerCount = 0;

        try{

            ss = new ServerSocket(26771); } catch(IOException ex) {
            System.out.println("IOException in Server Constructor");

        }
    }
    //Connects with 2 clients

    private void initiate() {

        try {

            System.out.println("Waiting for clients to connect with the server.....");
            while (playerCount < 2) {
                Socket s = ss.accept();

                playerCount++;

                // reflected in command prompt output
```

```

System.out.println("player #" + playerCount + " is connected." );

ServerConnection SC = new ServerConnection(s,playerCount);

if(playerCount == 1) {

    player1 = SC;
    // if first client to join to server then player 1
}
else {

    player2 = SC;
    // else player 2
}

Thread t = new Thread(SC);

t.start();
// start the thread/server
}
System.out.println("No more connections accepted.(both the players have
joined)");

// no more than 2 players accepted
}catch (IOException ex){

System.out.println("IOException while initiating connection, please try
again.");

}
}

private class ServerConnection implements Runnable {
private Socket socket;
private DataInputStream Din;
private DataOutputStream Dout;
private int playerID;

ServerConnection (Socket s, int id) {

    socket = s;
    // socket is created
    playerID = id;
    try {

        Din = new DataInputStream (socket.getInputStream()); Dout = new
        DataOutputStream(socket.getOutputStream());

```

```

    }
    catch(IOException ex) {
        System.out.println("IOException in ServerConnection");
    }
}

public void run() {

    try {

        Dout.writeInt(playerID);
        Dout.flush();
        while(true) {

            if(playerID == 1) {

                player1Val = Din.readInt();

                System.out.println("player1 clicked "+ player1Val);
                // output shown in command prompt as to which grid player 1 has selected
                player2.sendButtonVal(player1Val);
                // reflect which button did player 1 select and display that in player 2 GUI

            }

            else if(player1.socket.isClosed() || player2.socket.isClosed()) {
                System.out.println("Connection lost/broken. Please connect again.");
                // connection lost
                break;
            }

            else {

                player2Val = Din.readInt(); System.out.println("player2 clicked "+
                player2Val); player1.sendButtonVal(player2Val);

            }

        }}catch(Exception ex) {

            player1.closeConnection();
            player2.closeConnection();

        }
    }
}

```

```

public void sendButtonVal(int n) {
    try {

        Dout.writeInt(n);

        Dout.flush();

        System.out.println("Block mark value succesfully written back to client");
        // to verify if the value was successfully written in the GUI for both the
        clients
    } catch(IOException ex){
        System.out.println("IOException in sending block move value to opponent");

    }
}

public void closeConnection() {
    try {

        socket.close();

        System.out.println("Connection successfully closed.");

    }
    catch(IOException ex) {

        System.out.println("IOException while closing the connection");
    }
}

public static void main(String args[]) {

    Server server = new Server();
    server.initiate();

}
}

```



## **4.2 Client side code:**

```
// Game Client Side

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;
import java.net.*;
import java.util.*;

public class Client extends JPanel {

    private char playerMark;
    private int playerID;
    private int otherPlayer;
    private int x;
    private int currentplayerscore;
    private int OppplayerScore;
    private boolean setButtons;
    private JButton[] buttons = new JButton[9];
    private clientConnection CC;

    Scanner scan = new Scanner(System.in);

    public Client() {
        // client side values refreshing/reseting to 0
        setButtons = false;

        x = 0;

        playerID = 0;

        otherPlayer = 0;

        currentplayerscore = 0;

        OppplayerScore = 0;

    }

    //Build Gui for the Client

    public void buildGui() {

        JFrame window = new JFrame("player #" + playerID);

        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```

window.getContentPane().add(this); window.setBounds(400,400,400,400);

window.setLocationRelativeTo(null);

setLayout(new GridLayout(3,3));
//setting the values for playerId for both the players and synchronising them
if(playerID == 1) {

    otherPlayer = 2;
    setButtons = true;
    toggleButtons();

}

else {

    otherPlayer = 1 ;
    setButtons = false;
    toggleButtons();

    Thread t2 = new Thread(new Runnable() {

        public void run() {

            updateTurn();
        }
    });

    t2.start();
}

window.setVisible(true); // show the window

}

public void initializeButtons()

{

    for(int i = 0; i <= 8; i++)
    {

        buttons[i] = new JButton();
        buttons[i].setText(" ");
        buttons[i].setBackground(Color.WHITE);
        add(buttons[i]);

    }

}

```

```

}
//Gets the button clicked and send to server

public void keyPress() {

    ActionListener al = new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
            JButton buttonClicked = (JButton) ae.getSource();
            buttonClicked.setText(String.valueOf(playerMark));
            buttonClicked.setBackground(Color.RED);

            for(x = 0 ; x<9 ; x++) {
                //noting down which button was clicked
                if(buttonClicked == buttons[x]) {

                    break;

                }
            }
            //sending the buttonvalue to server
            CC.sendButtonVal(x);
            System.out.println("value sent to server");
            setButtons = false;
            toggleButtons();
            displayVector();
            System.out.println("You Clicked button no "+ x+" Wait for Player #" +
            otherPlayer);

            Thread t = new Thread(new Runnable() {

                public void run() {

                    updateTurn();
                }
            });

            t.start();

        }
    };

    for(int i=0;i<9;i++) {
        buttons[i].addActionListener(al);
    }
}

//Receives Value from server and write in Client
public void updateTurn() {

```

```

int n = CC.recButtonVal();

oppMove(n);
//writing opposition move and reflecting it in gui and commandprompt
System.out.println("Opponent feedback has been written");
setButtons = true;
toggleButtons();
}

public void toggleButtons() {

if(setButtons == false) {

for (int i=0;i<9;i++) {

buttons[i].setEnabled(setButtons);
}
}
else {

for(int i =0;i<9;i++) {

if(buttons[i].getText().charAt(0) == ' ') {

buttons[i].setEnabled(setButtons);
}
}
}
//To write an opponent feedback

public void oppMove(int oppval) {

char oppPlayerMark;

if (playerMark == 'X') {

oppPlayerMark = 'O';

}

else {

oppPlayerMark = 'X';

}
}

```

```

buttons[oppval].setText(String.valueOf(oppPlayerMark));
buttons[oppval].setBackground(Color.GREEN);
buttons[oppval].setEnabled(false);

displayVector();

}

public void playerMarkDecider(){

if (playerID == 1) {

playerMark = 'X';
}

else {

playerMark = 'O';
}
}

public void UpdatePoints(char x) {

if(x == playerMark) {

currentplayerscore++;
}

else {

OppplayerScore++;
}
}

//display the victorious player
public void displayVector() {

if(checkForWinner('X') == true) {

JOptionPane pane = new JOptionPane();

int dialogResult = JOptionPane.showConfirmDialog(pane, "X wins. \n
you(#1):"+currentplayerscore+" \t opponent(#2):"+OppplayerScore+" \n Would you
like to play again?", "Game over.",JOptionPane.YES_NO_OPTION);

if(dialogResult == JOptionPane.YES_OPTION)

{resetTheButtons();}

```

```

else {

CC.closeConnection();
System.exit(0);
}
}

else if(checkForWinner('O') == true) {

JOptionPane pane = new JOptionPane();

int dialogResult = JOptionPane.showConfirmDialog(pane, "O wins. \n
you(#1):"+currentplayerscore+" \t opponent(#2):"+OppplayerScore+" \n Would you
like to play again?", "Game over.", JOptionPane.YES_NO_OPTION);

if(dialogResult == JOptionPane.YES_OPTION) resetTheButtons();

else {

CC.closeConnection();
System.exit(0);
}
}

else if(checkDraw()) {

JOptionPane pane = new JOptionPane();
int dialogResult = JOptionPane.showConfirmDialog(pane, "Draw \n
you(#1):"+currentplayerscore+" \t opponent(#2):"+OppplayerScore+" \n Would you
like to play again?", "Game over.", JOptionPane.YES_NO_OPTION);
if(dialogResult == JOptionPane.YES_OPTION)
{resetTheButtons();}

else {

CC.closeConnection();
System.exit(0);
}
}
}

private void resetTheButtons() {

if(playerMark == 'O') {

playerMark = 'X';
setButtons = true;

```

```

}
else {

playerMark = 'X' ;
setButtons = false;

}
toggleButtons();

for(int i =0;i<9;i++) {

buttons[i].setText(" ");
buttons[i].setBackground(Color.WHITE);
}
}

// checks for draw
public boolean checkDraw() {

boolean full = true;
for(int i = 0 ; i<9;i++) {

if(buttons[i].getText().charAt(0) == ' ') {

full = false;
}
}
return full;
}

// checks for a winner

public boolean checkForWinner(char x) {

if(checkRows(x) == true || checkColumns(x) == true || checkDiagonals(x)
==true) {
return true;
}
else return false;
}

// checks rows for a win
public boolean checkRows(char x) {
int i = 0;
for(int j = 0;j<3;j++) {

if( buttons[i].getText().equals(buttons[i+1].getText()) &&

```

```

buttons[i].getText().equals(buttons[i+2].getText()) &&
buttons[i].getText().charAt(0) != ' ' &&
buttons[i].getText().charAt(0) == x) {

    UpdatePoints(x);
    return true;
}
i = i+3;
}
return false;
}

// checks columns for a win

public boolean checkColumns(char x) {
    int i = 0;
    for(int j = 0; j<3; j++) {
        if( buttons[i].getText().equals(buttons[i+3].getText()) &&
            buttons[i].getText().equals(buttons[i+6].getText())&&
            buttons[i].getText().charAt(0) != ' ' &&
            buttons[i].getText().charAt(0) == x)
        {

            UpdatePoints(x);
            return true;
        }
        i++;
    }
    return false;
}

// checks diagonals for a win

public boolean checkDiagonals(char x) {
    if(buttons[0].getText().equals(buttons[4].getText()) &&
        buttons[0].getText().equals(buttons[8].getText())&&
        buttons[0].getText().charAt(0) != ' ' && buttons[0].getText().charAt(0) == x)
    {

        UpdatePoints(x);
        return true;

    }
    else if(buttons[2].getText().equals(buttons[4].getText()) &&
        buttons[2].getText().equals(buttons[6].getText())&&
        buttons[2].getText().charAt(0) != ' ' && buttons[2].getText().charAt(0) == x)
    {

        UpdatePoints(x);

```



```

return true;
}
else return false;
}

public void connectToServer() {

CC = new clientConnection();
}

private class clientConnection {
private Socket socket;
private DataInputStream Din;
private DataOutputStream Dout;
private String ip;

public clientConnection() {

try {

System.out.println("Client side.");
System.out.println("Enter IP address:");
ip = scan.nextLine();
socket = new Socket(ip,26771);
Din = new DataInputStream(socket.getInputStream()); Dout = new
DataOutputStream(socket.getOutputStream()); playerId = Din.readInt();
System.out.println("player # " + playerId + "is connected to server"); }catch
(IOException ex) {

// System.out.println("IOException in client constructor"); System.exit(0);
}
}

public void sendButtonVal(int a) {

try {

Dout.writeInt(a);
Dout.flush();
System.out.println("value sent to server !");

}catch(IOException ex) {

System.out.println("IOException in sendButton()");
System.exit(0);

}
}
}

```

```

}

public int recButtonVal() {

int n=-1;

try {

n = Din.readInt();

System.out.println("Value received from server"); }catch (IOException ex) {

System.out.println("error in recButton()");

}

return n;

}

public void closeConnection() {

try {

socket.close();

System.out.println("Socket closed successfully");

}catch(IOException ex) {

System.out.println("IOException while closing");

}

}

}

public static void main(String[] args) {

Client cl = new Client();

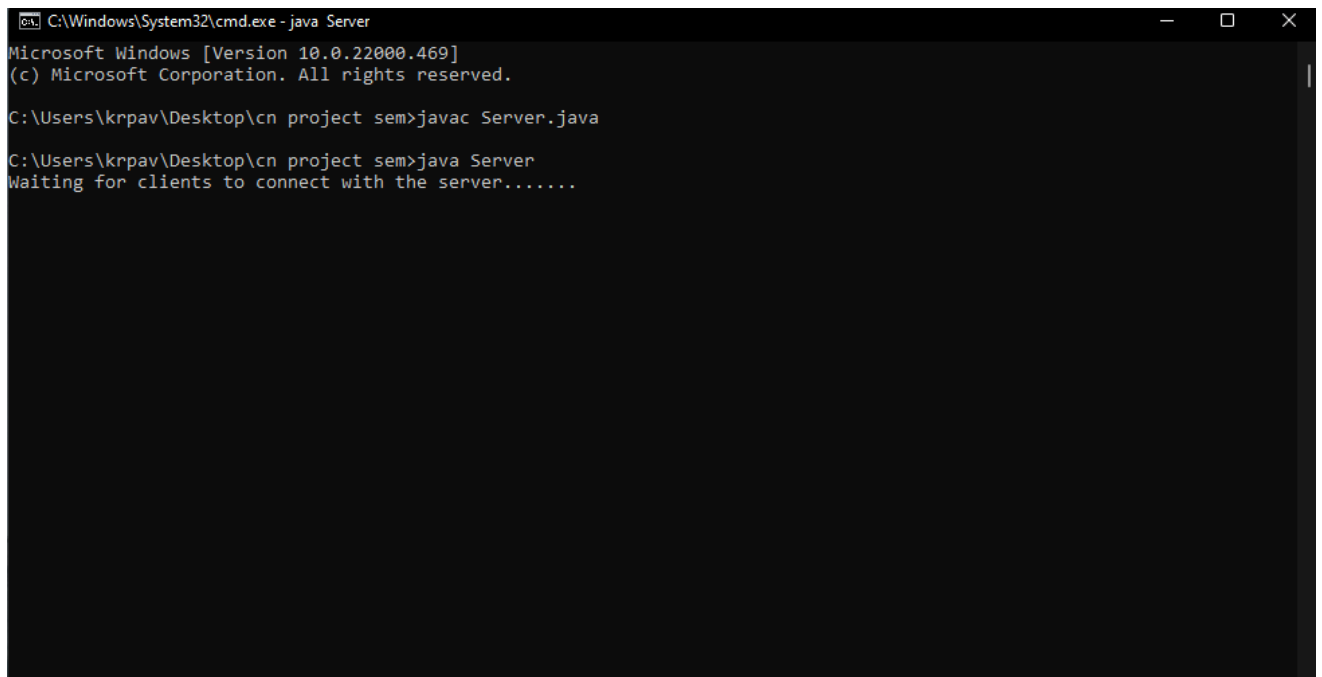
cl.connectToServer();
cl.playerMarkDecider();
cl.initializeButtons();
cl.buildGui();
cl.keyPress();
}
}

```

## CHAPTER 5

### RESULTS

**Fig 4.1 Server started**

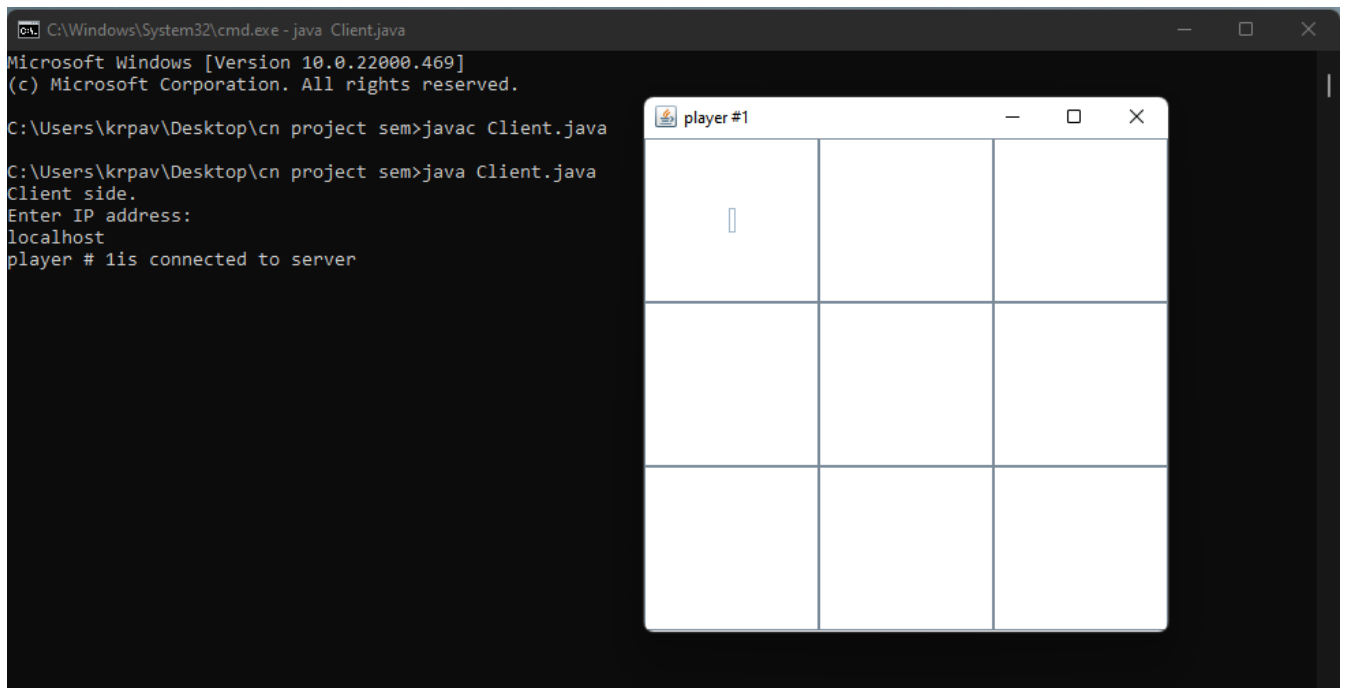


```
C:\Windows\System32\cmd.exe - java Server
Microsoft Windows [Version 10.0.22000.469]
(c) Microsoft Corporation. All rights reserved.

C:\Users\krpav\Desktop\cn project sem>javac Server.java

C:\Users\krpav\Desktop\cn project sem>java Server
Waiting for clients to connect with the server.....
```

**Fig 4.2 Client 1 joined using localhost**



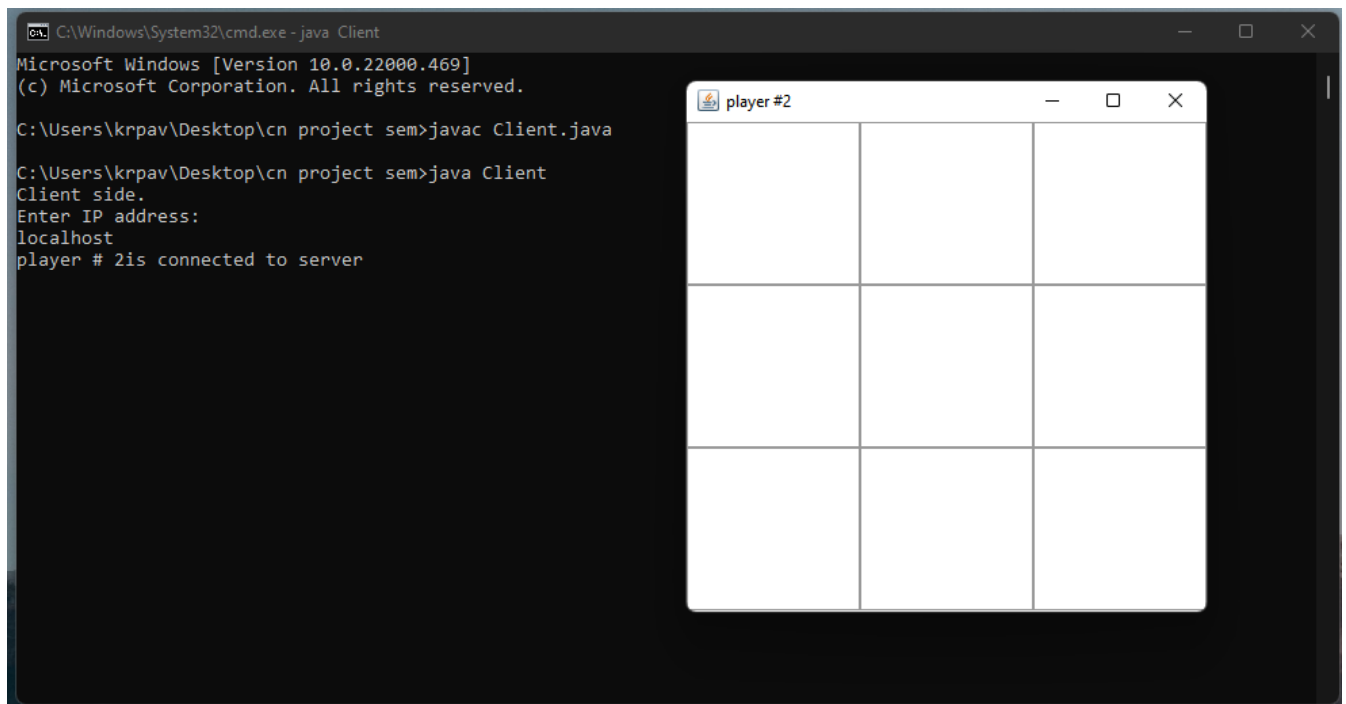
```
C:\Windows\System32\cmd.exe - java Client.java
Microsoft Windows [Version 10.0.22000.469]
(c) Microsoft Corporation. All rights reserved.

C:\Users\krpav\Desktop\cn project sem>javac Client.java

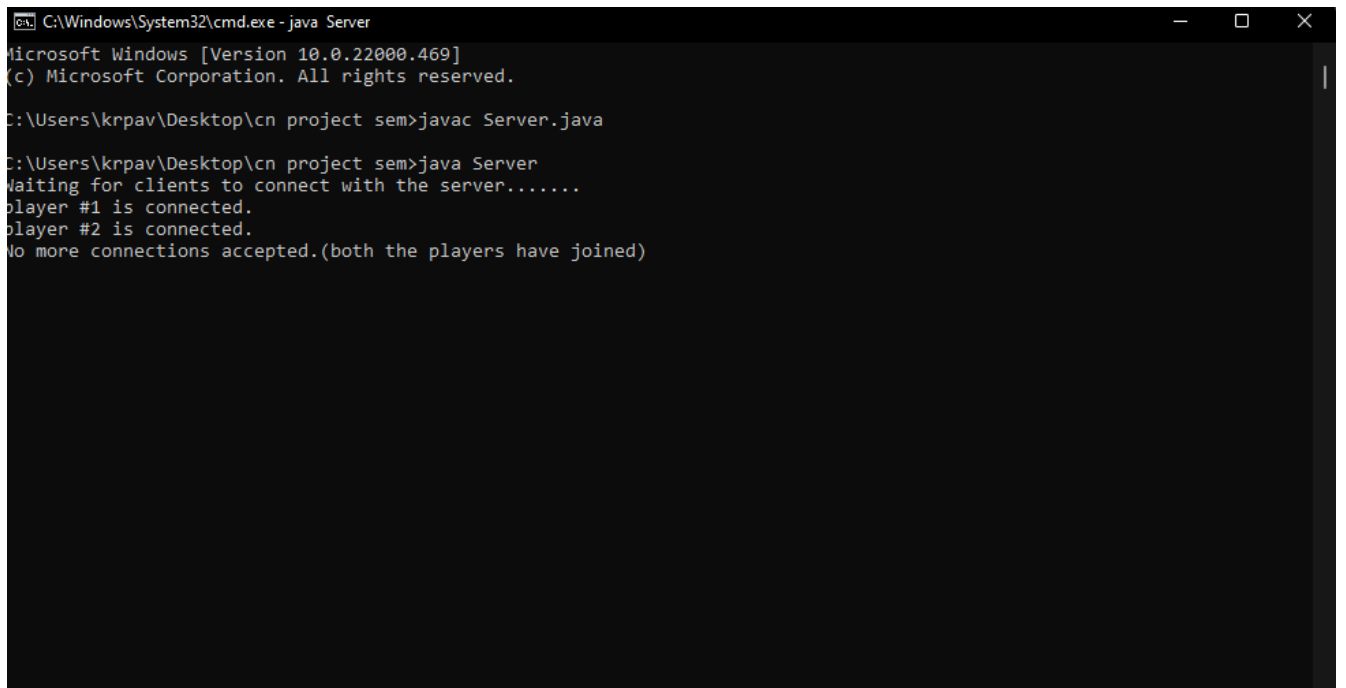
C:\Users\krpav\Desktop\cn project sem>java Client.java
Client side.
Enter IP address:
localhost
player # 1 is connected to server
```

player #1		

**Fig 4.3 Client 2 joined using localhost**

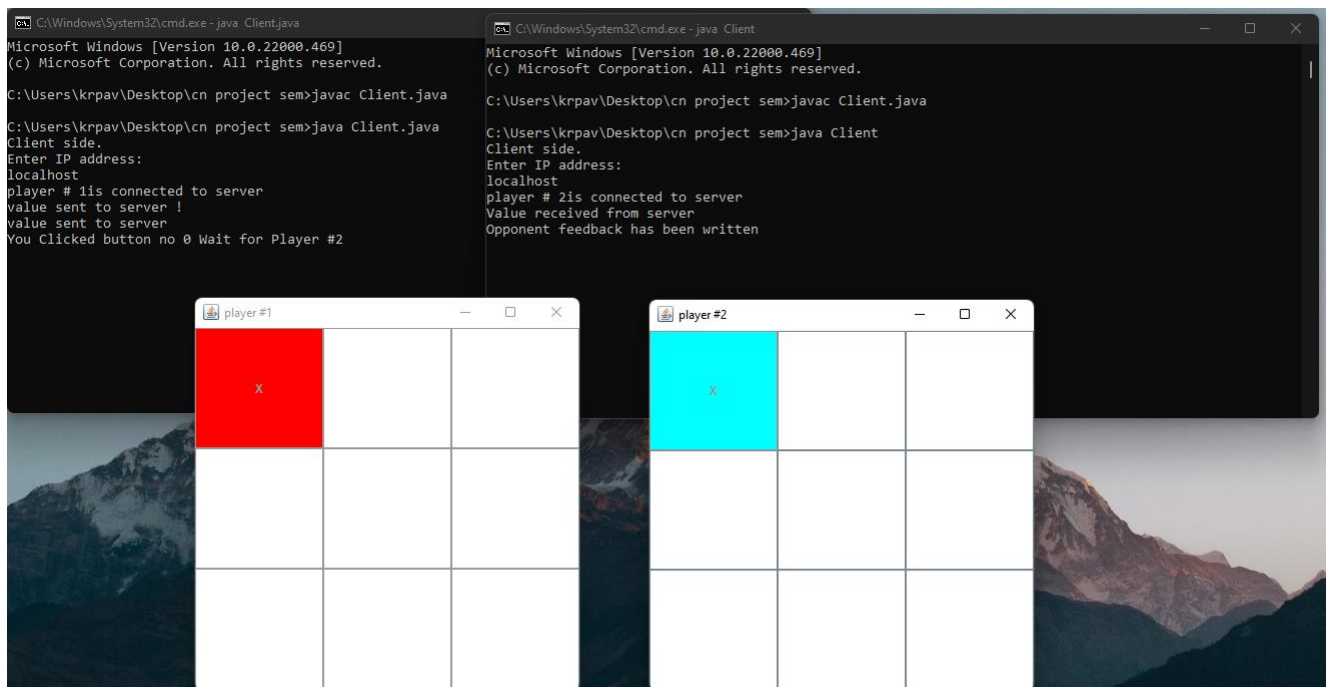


**Fig 4.4 Server stops accepting anyomore clients (2 joined)**

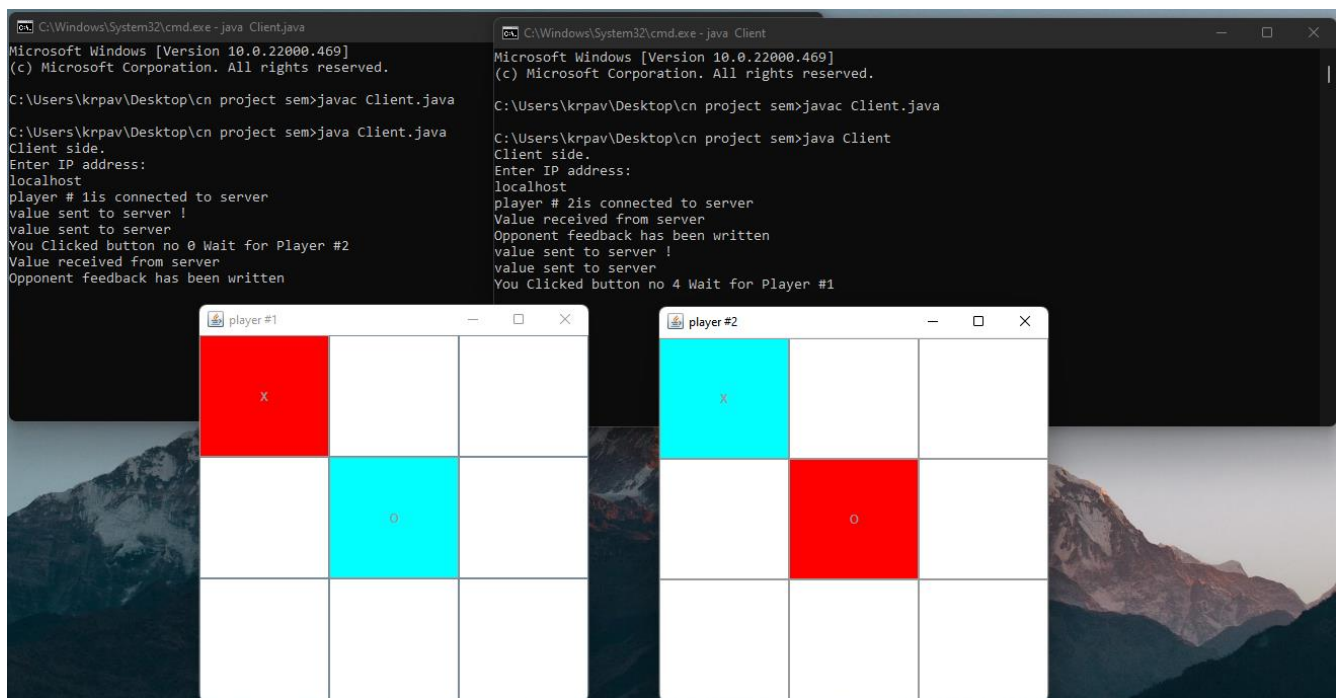


## Player 1 wins(“X” wins):

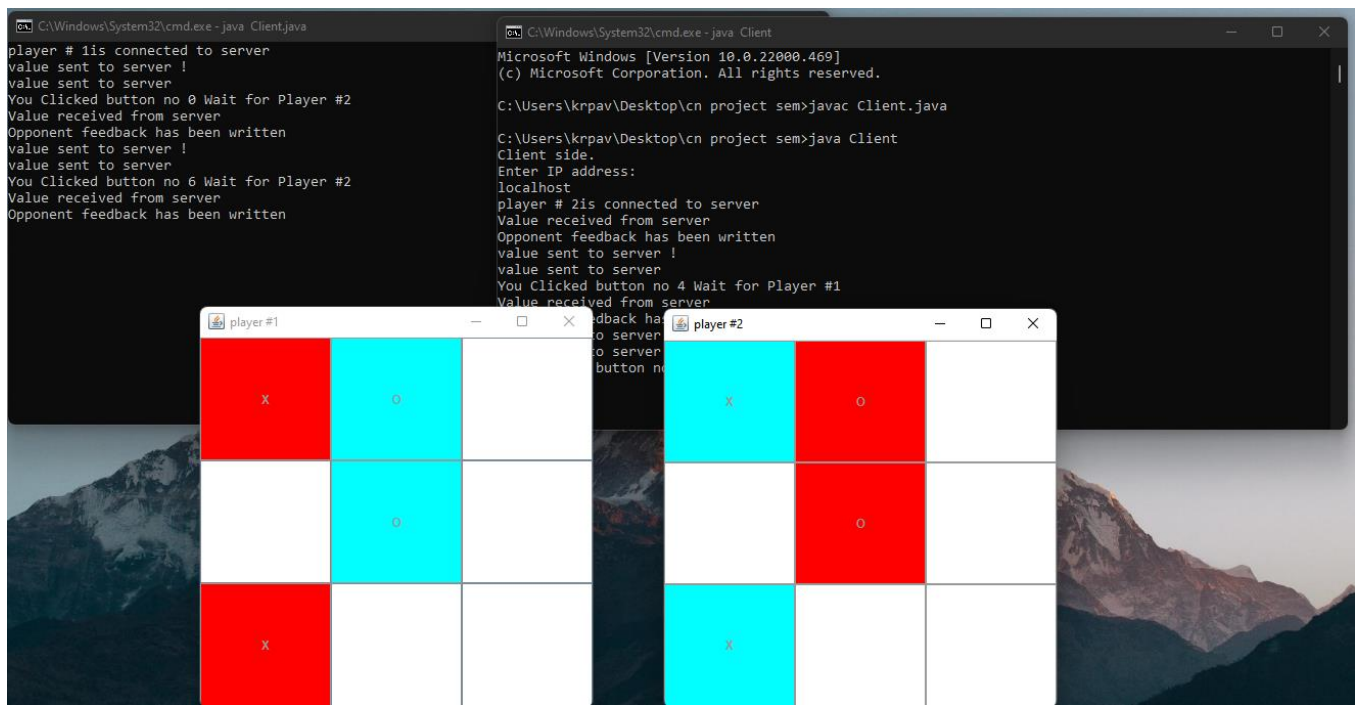
**Fig 4.5: Input 1 from #player1**



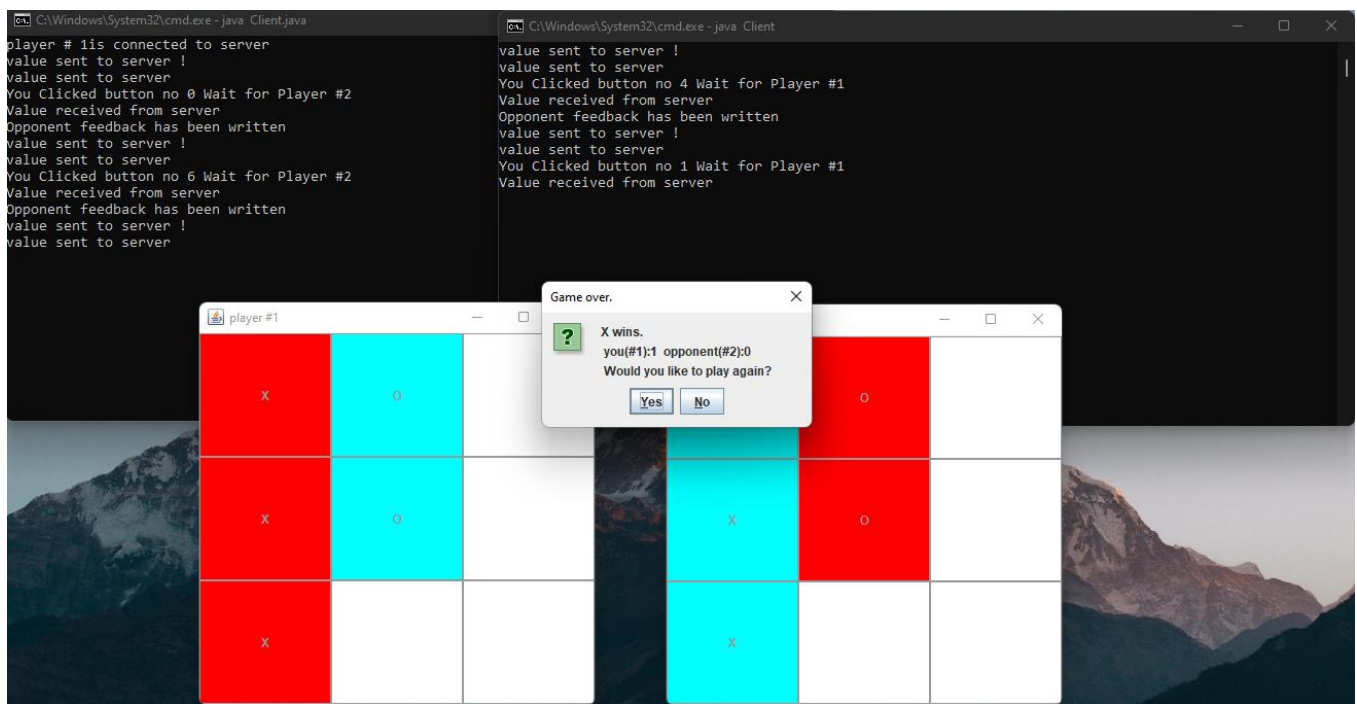
**Fig 4.6 1<sup>st</sup> input from #player 2:**



**Fig 4.7 2<sup>nd</sup> input from #player 1:**

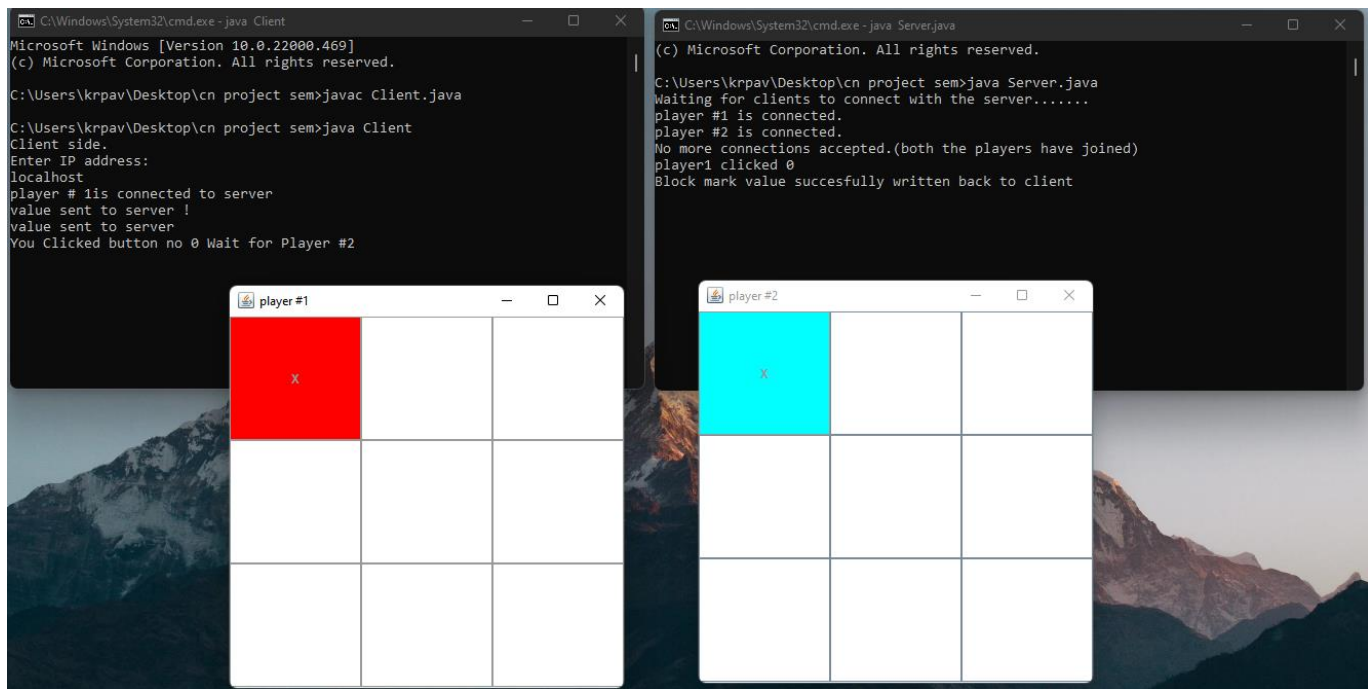


**Fig 4.8: 2<sup>nd</sup> input from #player 2**

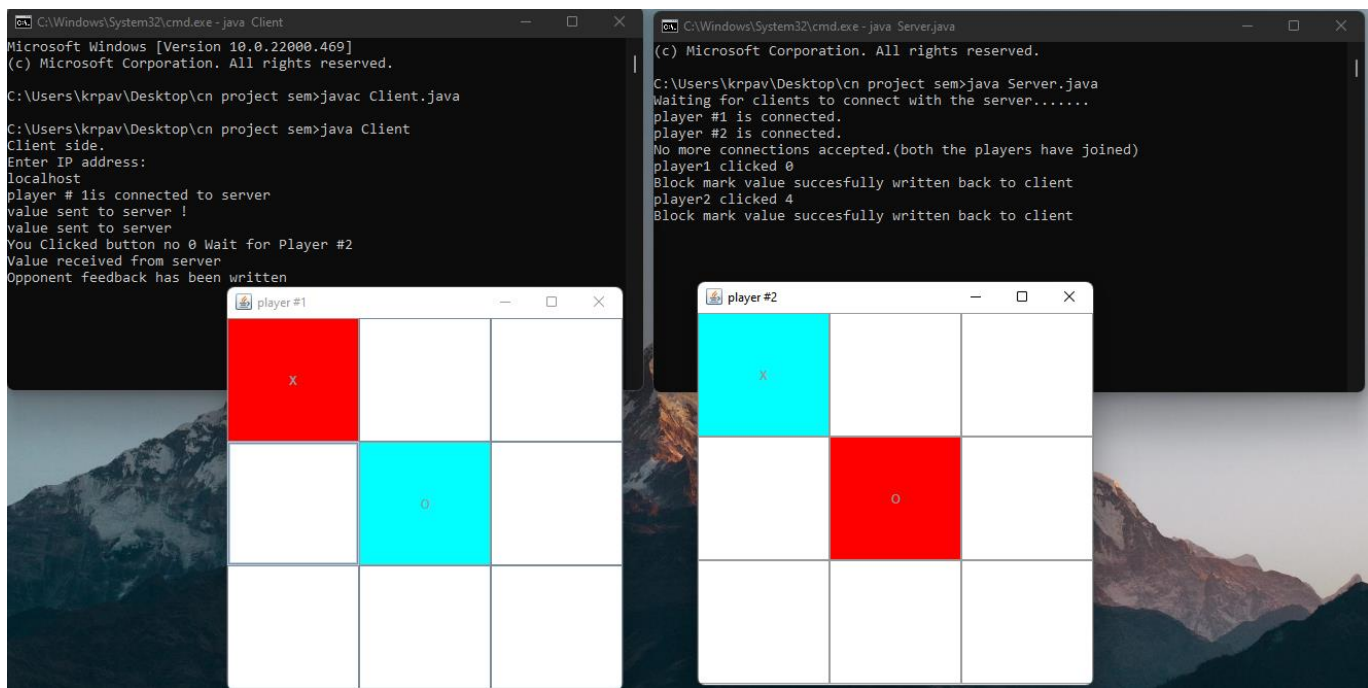


## Player 2 wins(“O” wins):

**Fig 4.9 1<sup>st</sup> input from #player1**

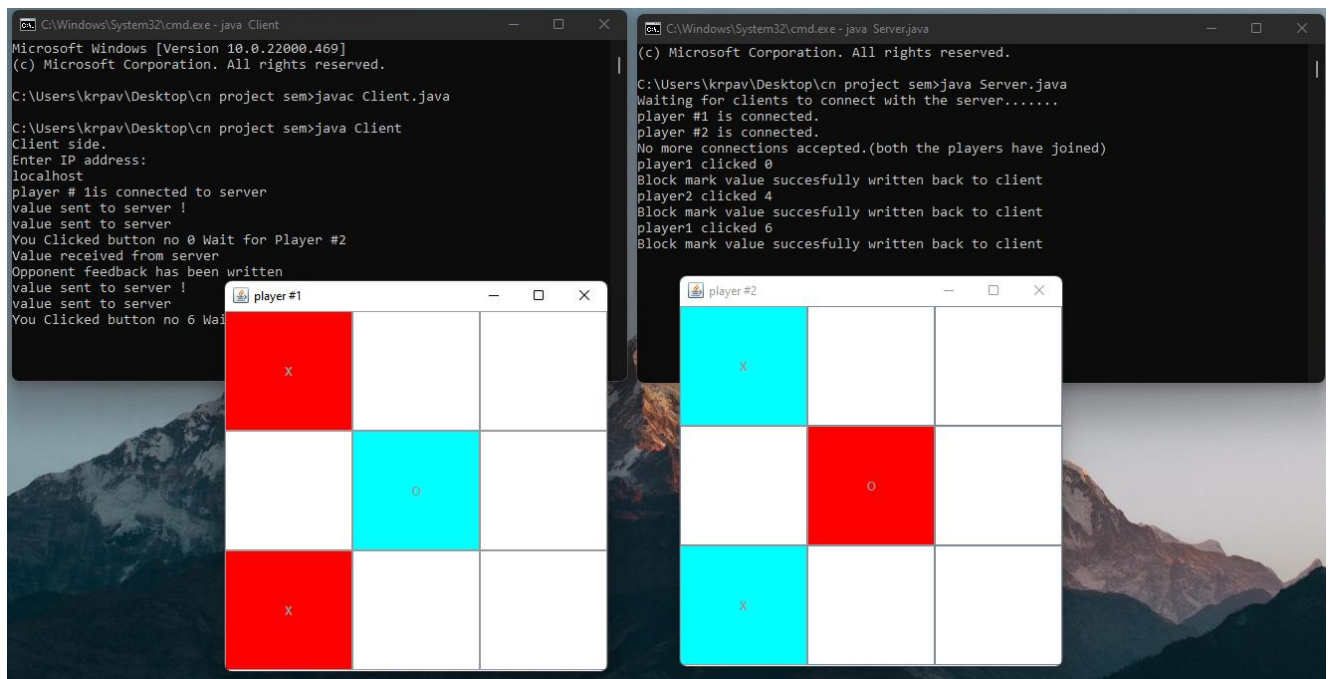


**Fig 4.10:1<sup>st</sup> input from #player2**

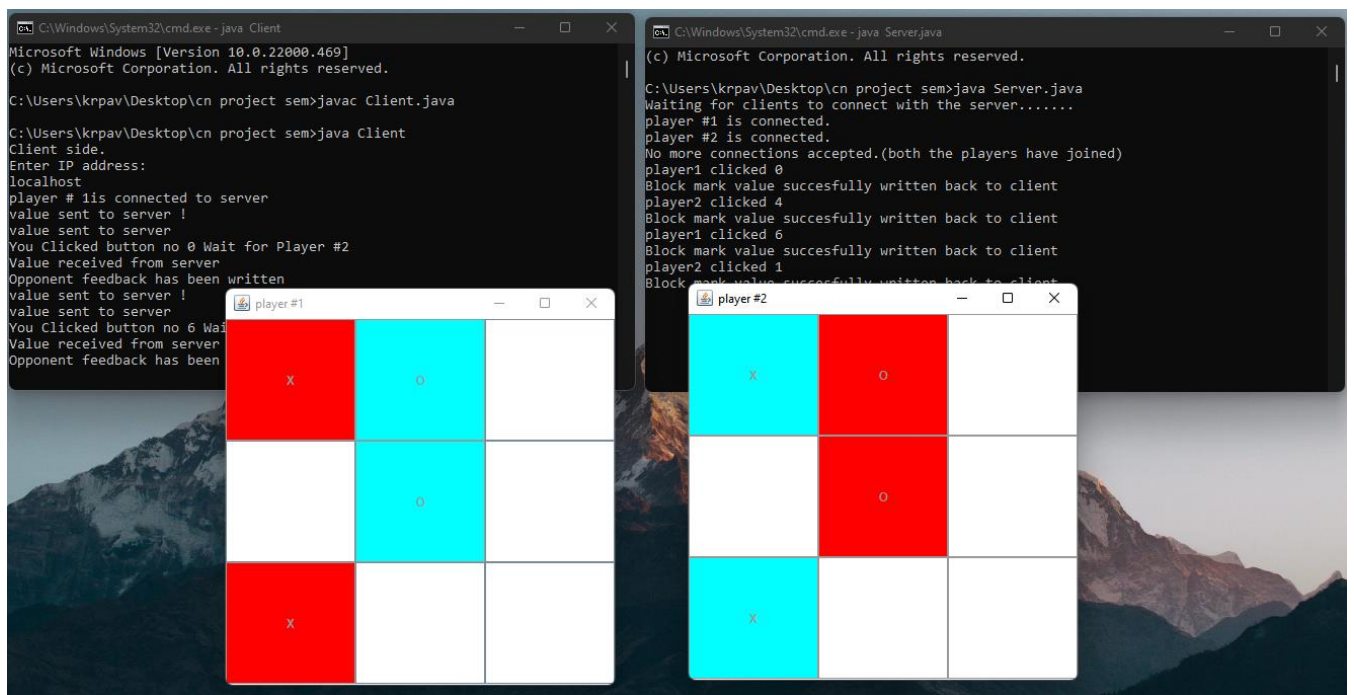




**Fig 4.11: 2<sup>nd</sup> input from #player2**

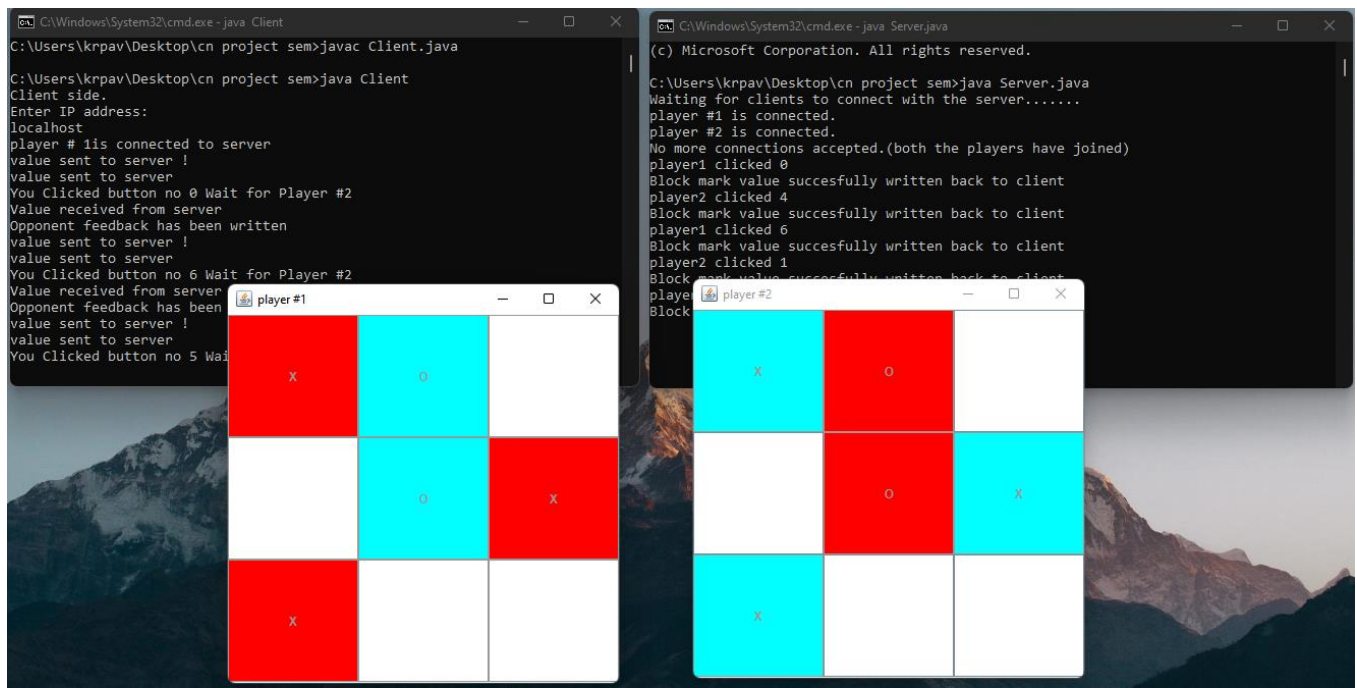


**Fig 4.12: 2<sup>nd</sup> input from #player2**

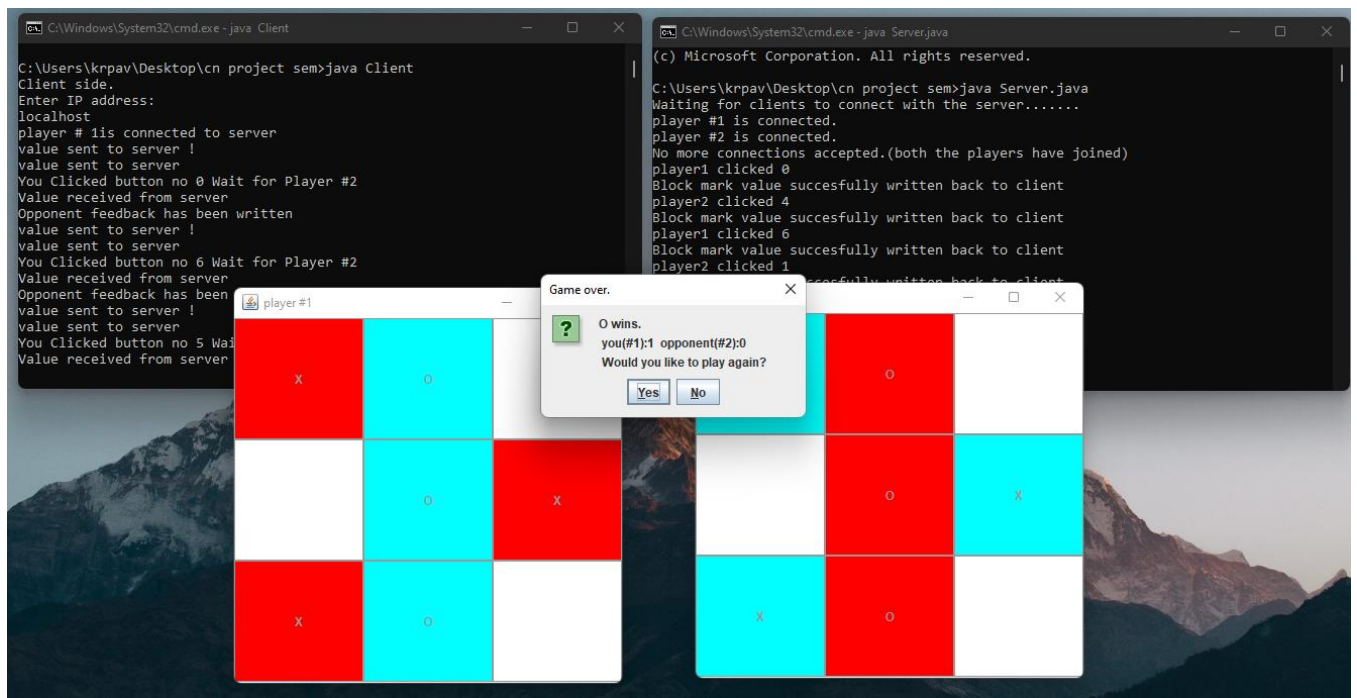




**Fig 4.13: 3<sup>rd</sup> input from #player1:**

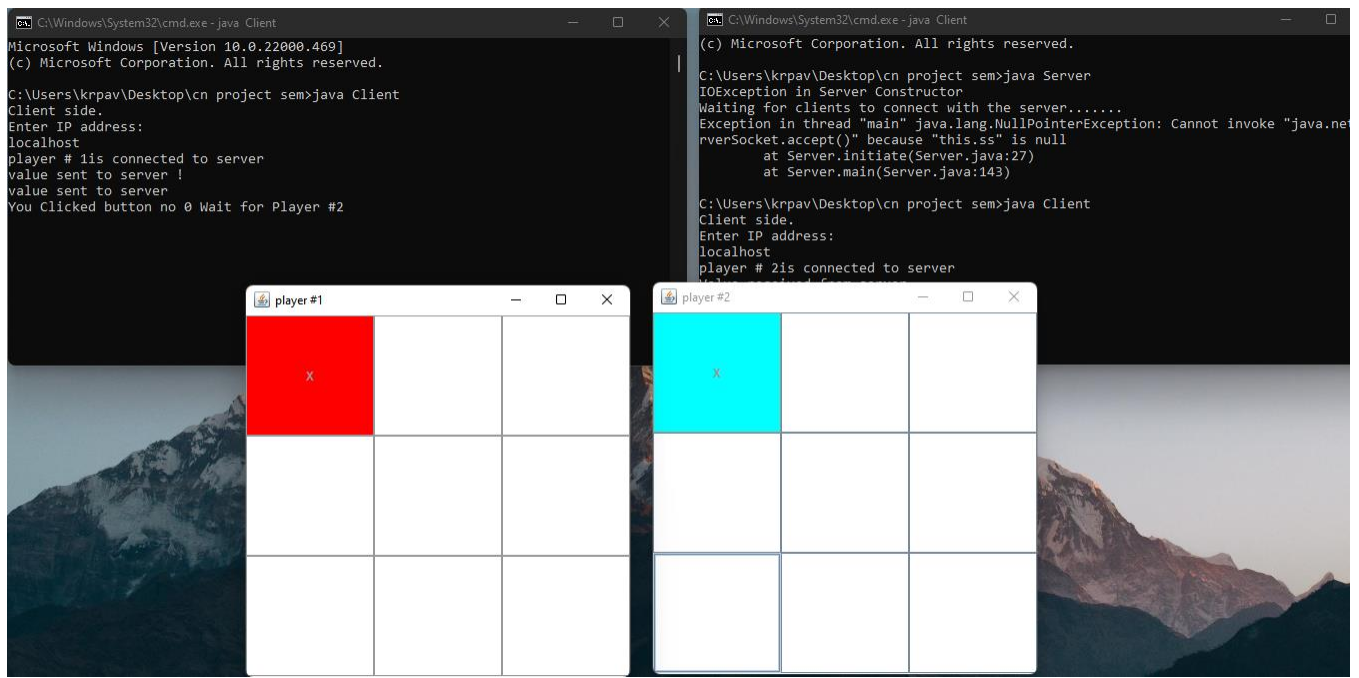


**Fig 4.14: player 2 wins**

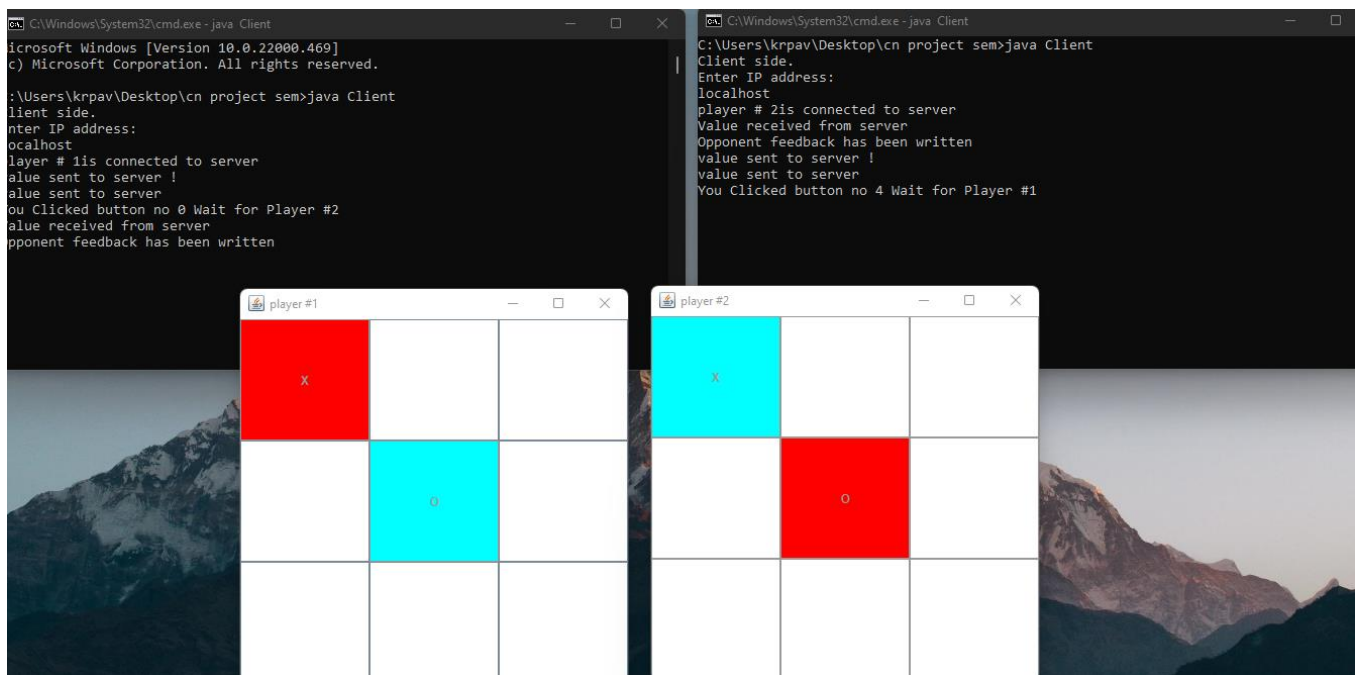


## Draw match:

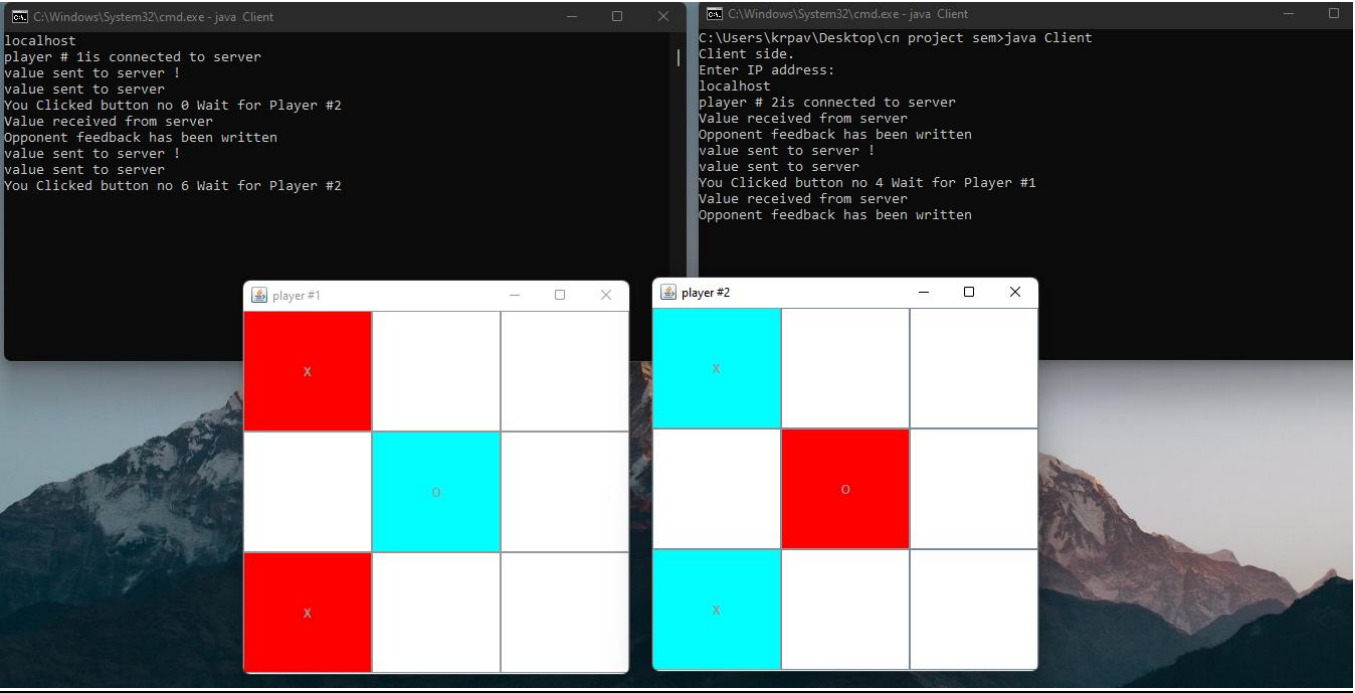
**Fig 4.15: 1<sup>st</sup> input from #player1**



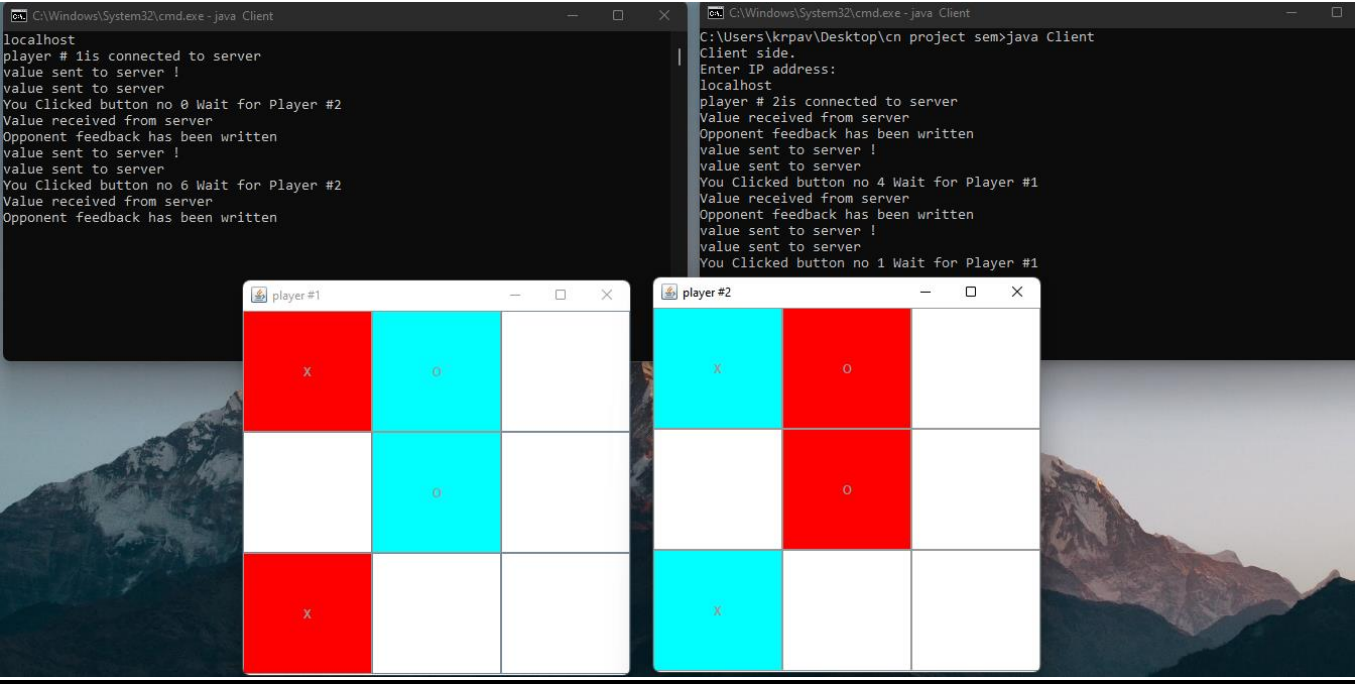
**Fig 4.16: 1<sup>st</sup> input from #player2**



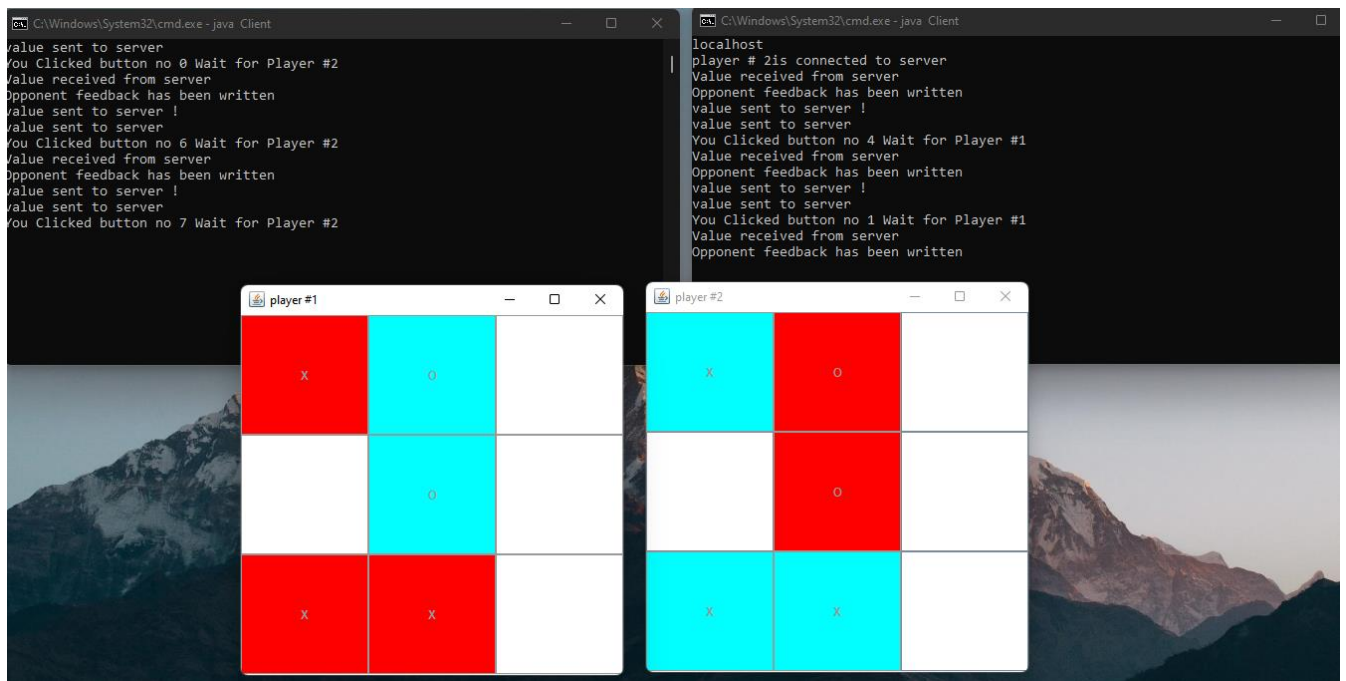
**Fig 4.17: 2<sup>nd</sup> input from #player1**



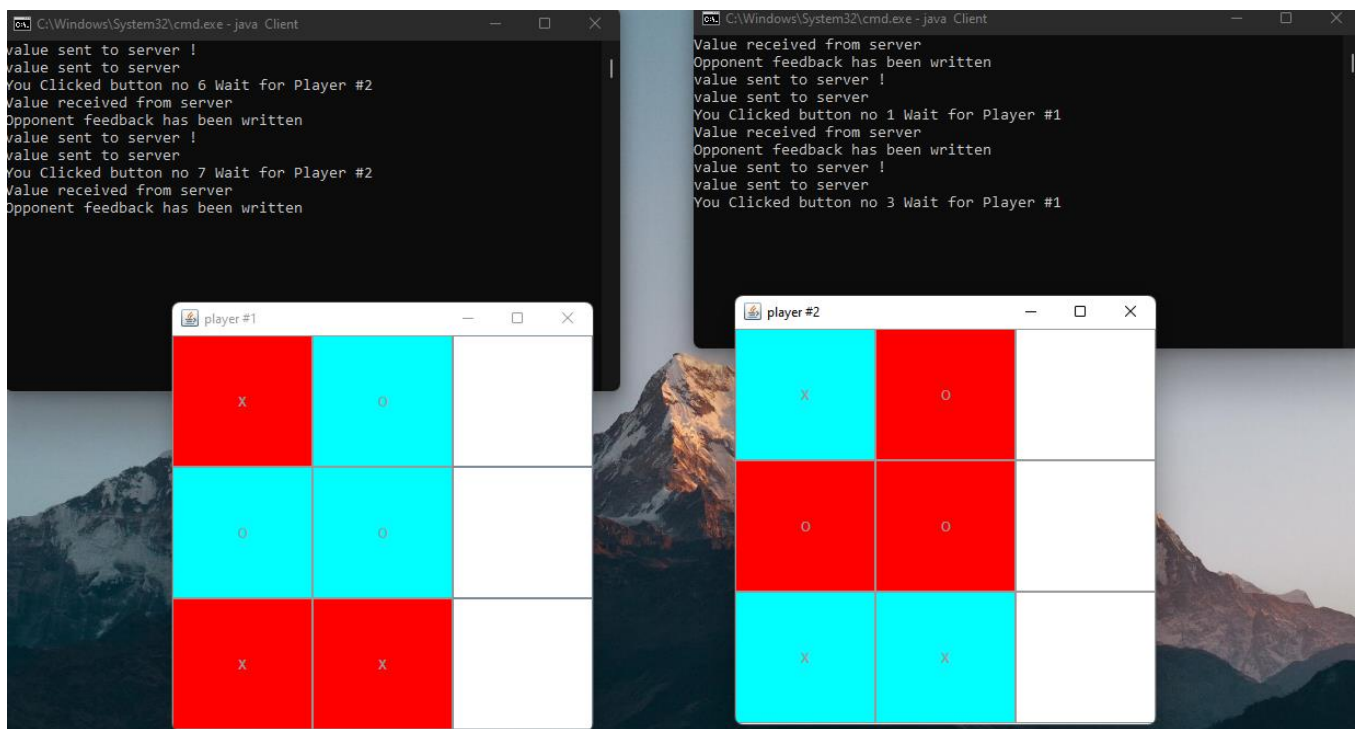
**Fig 4.18: 2<sup>nd</sup> input from #player2**



**Fig 4.19: 3<sup>RD</sup> input from #player1**

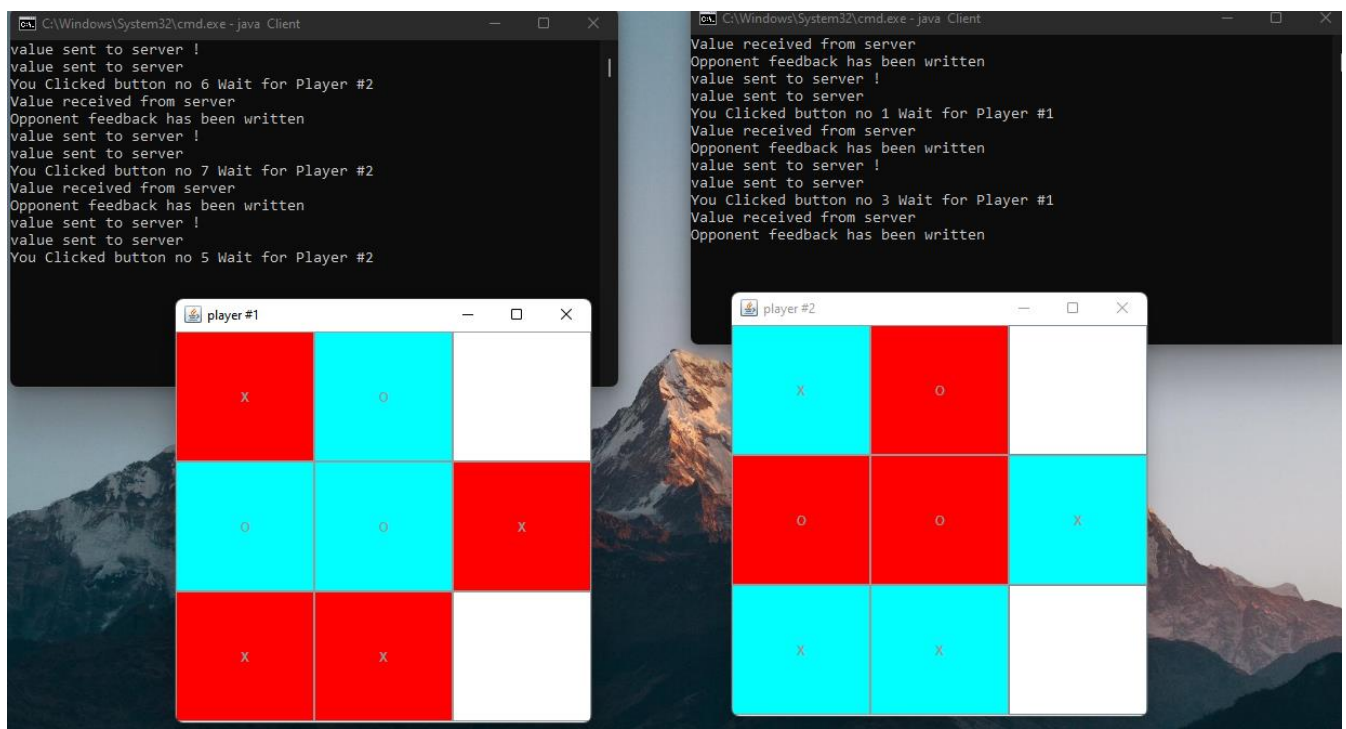


**Fig 4.20: 3<sup>rd</sup> input from #player2**

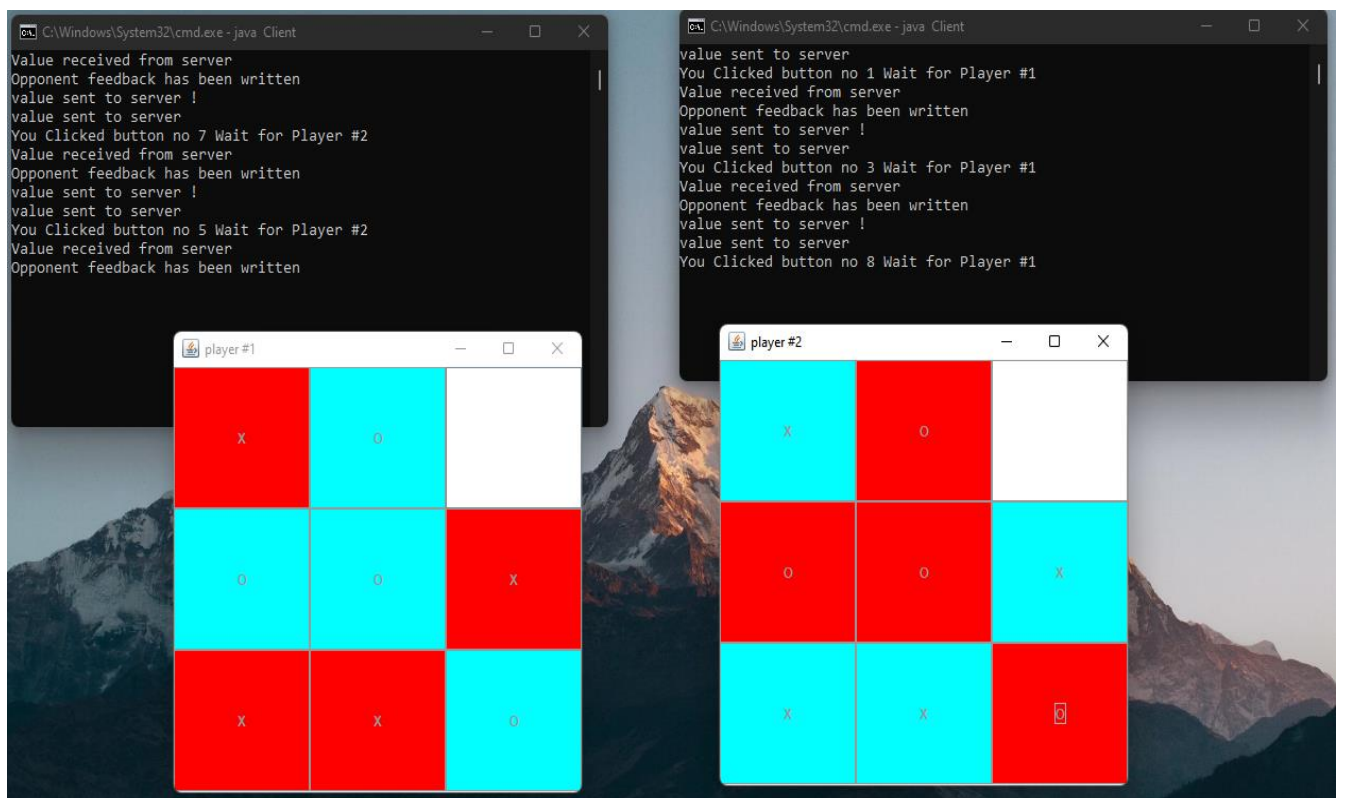




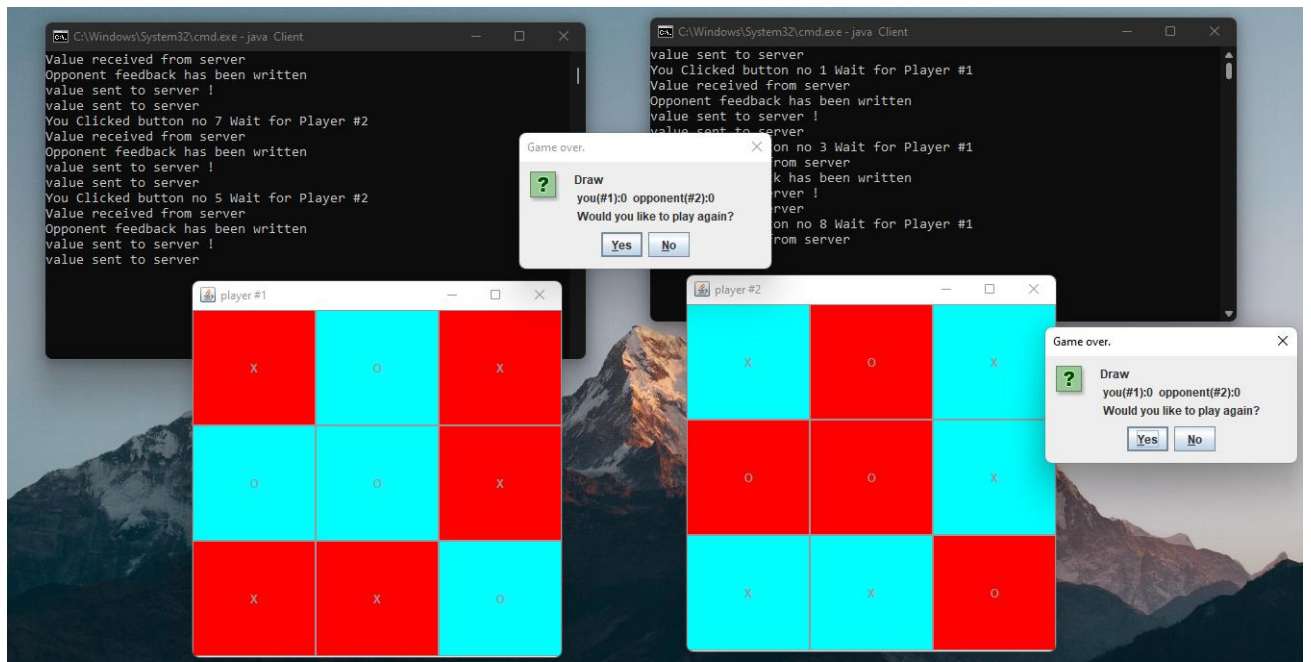
**Fig 4.21: 4<sup>th</sup> input from #player1**



**Fig 4.22: 4<sup>th</sup> input from #player2**



**Fig 4.23: Draw match so no one gets a point**



## **CHAPTER 7**

### **CONCLUSION**

We have created a tic-tac-toe game where 2 players across any computer are able to play with each other(connecting via IP address).

The TCP/IP and multi-threading has been implemented and tested successfully. The input moves were transmitted fast and there was no lag found during the compilation. Both the players were able to communicate with each other. We have implemented using Client Server architecture, Socket programming, TCP protocol, java socket

In order to optimize the current application:

1. We can make the server connect to more than 2 clients as the server is capable of handling many clients at the same time. The client connecting to each other can be randomised or a specific socket can be given as a key to join a private room of 2 players, to play the game.
2. Make the game available via web application and a frontpage using GUI.
3. Have player history for all the previous matches by creating an account.
4. Create a mobile application using Flutter(dart language similar to java) and play across multi-platform

## CHAPTER 8

### REFERENCES

- Kalita, L. (2014). Socket programming. *International Journal of Computer Science and Information Technologies*, 5(3), 4802-4807.
- Li, Y., Li, D., Cui, W., & Zhang, R. (2011, May). Research based on OSI model. In *2011 IEEE 3rd International Conference on Communication Software and Networks* (pp. 554-557). IEEE.
- Leung, K. C., Li, V. O., & Yang, D. (2007). An overview of packet reordering in transmission control protocol (TCP): problems, solutions, and challenges. *IEEE transactions on parallel and distributed systems*, 18(4), 522-535.
- Xue, M., & Zhu, C. (2009, May). The socket programming and software design for communication based on client/server. In *2009 Pacific-Asia Conference on Circuits, Communications and Systems* (pp. 775-777). IEEE.
- Karamchandani, S., Gandhi, P., Pawar, O., & Pawaskar, S. (2015, January). A simple algorithm for designing an artificial intelligence based Tic Tac Toe game. In *2015 International Conference on Pervasive Computing (ICPC)* (pp. 1-4). IEEE.

<https://theconversation.com/social-video-games-to-play-during-the-coronavirus-quarantine-134880>

<https://www.geeksforgeeks.org/introducing-threads-socket-programming-java/>