**JAYPEE INSTITUTE OF INFORMATION AND TECHNOLOGY**

**  
  
  
  
  
  
 COMPUTER NETWORKS AND SECURITY LAB (18B15CS212)**

**PROJECT BASED LEARNING(PBL)**

**PROJECT TITLE: Tic- Tac- Toe using Socket Programming**

**SUBMITTED TO: Dr. Raghu Vamsi Pothukuchi**

**ACKNOWLEDGEMENT**

# We would like to express our deepest gratitude to Dr. Raghu Vamsi Pothukuch, our professor-in-charge for their tremendous support and guidance in completing our project on the topic- Tic- Tac- Toe game using Socket Programming. It was a great learning experience for the group to learn the basic applications of the subject Computer Networks and Security and more importantly the spirit to work in a group.

# We would also like to take this opportunity to expand our gratitude to our families, friends, and colleagues. The project would not have been successful without their cooperation and inputs.

# **GROUP MEMBERS**

BATCH: B-14

BRANCH: IT

YEAR: 3

SEMESTER: 5 (ODD SEM 2023)

| **NAME** | **ABHILASH**  **TYAGI** | **OJASVI**  **PANDEY** |
| --- | --- | --- |
| ENROLLMENT  NUMBER | 21104019 | 21104027 |

# 

# **INDEX**

1. Introduction
2. Technologies Used
3. Project Overview
4. Program Code
5. Output
6. Workflow
7. Challenges Faced
8. Conclusion
9. Future Work

**INTRODUCTION:**

The aim of this project was to implement a two-player Tic-Tac-Toe game using Python and socket programming as a practical application of concepts learned in the Computer Networks coursework. The project focused on creating a server-client architecture where two players could connect over a network and play the game.

**TECHNOLOGIES USED:**

* Programming Language: Python
* Libraries:`socket`, `pickle`
* Concepts: Client-Server Architecture, Socket Programming

**PROJECT OVERVIEW:**

The project comprises two main components:

**1. Server Script (`host.py`):**

- Handles the server-side operations.

- Establishes a socket connection and listens for incoming client connections.

- Controls the flow of the game, managing the logic, and facilitating communication between the players.

- Implements the Tic-Tac-Toe game rules and functionalities.

**2. Client Script (`client.py`):**

- Represents a player in the game.

- Connects to the server using socket communication.

- Interacts with the server to play the game by making moves and responding to prompts.

**PROGRAM CODE:**

class TicTacToe():

def \_\_init\_\_(self, player\_symbol):

# initialize the list of symbols

self.symbol\_list = []

# defines all nine symbols; all start off as blank

for i in range(9):

self.symbol\_list.append(" ")

# initializes the player symbol

self.player\_symbol = player\_symbol

def restart(self):

# clears the grid

for i in range(9):

self.symbol\_list[i] = " "

def draw\_grid(self):

# display the column headers

print("\n A B C\n")

# display first row

row\_one = " 1 " + self.symbol\_list[0]

row\_one += " ║ " + self.symbol\_list[1]

row\_one += " ║ " + self.symbol\_list[2]

print(row\_one)

# display divider

print(" ═══╬═══╬═══")

# display second row

row\_two = " 2 " + self.symbol\_list[3]

row\_two += " ║ " + self.symbol\_list[4]

row\_two += " ║ " + self.symbol\_list[5]

print(row\_two)

# display divider

print(" ═══╬═══╬═══")

# display third and last row

row\_three = " 3 " + self.symbol\_list[6]

row\_three += " ║ " + self.symbol\_list[7]

row\_three += " ║ " + self.symbol\_list[8]

print(row\_three, "\n")

def edit\_square(self, grid\_coord):

# swamps coordinates such as "1A" to "A1"

if grid\_coord[0].isdigit():

grid\_coord = grid\_coord[1] + grid\_coord[0]

# divides the coordinate

col = grid\_coord[0].capitalize()

row = grid\_coord[1]

# converts "A1" to 0, "C3" to 8, and so forth

grid\_index = 0

if row == "1":

if col == "A":

grid\_index = 0

elif col == "B":

grid\_index = 1

elif col == "C":

grid\_index = 2

elif row == "2":

if col == "A":

grid\_index = 3

elif col == "B":

grid\_index = 4

elif col == "C":

grid\_index = 5

elif row == "3":

if col == "A":

grid\_index = 6

elif col == "B":

grid\_index = 7

elif col == "C":

grid\_index = 8

if self.symbol\_list[grid\_index] == " ":

self.symbol\_list[grid\_index] = self.player\_symbol

def update\_symbol\_list(self, new\_symbol\_list):

for i in range(9):

self.symbol\_list[i] = new\_symbol\_list[i]

def did\_win(self, player\_symbol):

# local variable to replace unweildy self.symbol\_list

g = []

for i in range(9):

g.append(self.symbol\_list[i])

# likewise to replace self.player\_symbol

sym = player\_symbol

# check top row

if g[0] == sym and g[1] == sym and g[2] == sym:

return True

# check middle row

elif g[3] == sym and g[4] == sym and g[5] == sym:

return True

# check bottom row

elif g[6] == sym and g[7] == sym and g[8] == sym:

return True

# check left column

elif g[0] == sym and g[3] == sym and g[6] == sym:

return True

# check middle column

elif g[1] == sym and g[4] == sym and g[7] == sym:

return True

# check right column

elif g[2] == sym and g[5] == sym and g[8] == sym:

return True

# check top-right to bottom-left

elif g[2] == sym and g[4] == sym and g[6] == sym:

return True

# check top-left to bottom-right

elif g[0] == sym and g[4] == sym and g[8] == sym:

return True

# didn't win... yet!

return False

def is\_draw(self):

# see if all the spaces are used up

num\_blanks = 0

for i in range(9):

if self.symbol\_list[i] == " ":

num\_blanks += 1

# if the player didn't win and no spaces are left, it's a draw

if self.did\_win(self.player\_symbol) == False and num\_blanks == 0:

return True

else:

return False

import socket # for networking

import pickle # for sending/receiving objects

# import the game

from tic\_tac\_toe import TicTacToe

HOST = '127.0.0.1' # the server's IP address

PORT = 12783 # the port we're connecting to

# connect to the host

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

s.connect((HOST, PORT))

print(f"\nConnected to {s.getsockname()}!")

# set up the game

player\_o = TicTacToe("O")

# allow the player to suggest playing again

rematch = True

while rematch == True:

# a header for an intense tic-tac-toe match!

print(f"\n\n T I C - T A C - T O E ")

# draw the grid

player\_o.draw\_grid()

# host goes first, client receives first

print(f"\nWaiting for other player...")

x\_symbol\_list = s.recv(1024)

x\_symbol\_list = pickle.loads(x\_symbol\_list)

player\_o.update\_symbol\_list(x\_symbol\_list)

# the rest is in a loop; if either player has won, it exits

while player\_o.did\_win("O") == False and player\_o.did\_win("X") == False and player\_o.is\_draw() == False:

# draw grid, ask for coordinate

print(f"\n Your turn!")

player\_o.draw\_grid()

player\_coord = input(f"Enter coordinate: ")

player\_o.edit\_square(player\_coord)

# draw grid again

player\_o.draw\_grid()

# pickle the symbol list and send it

o\_symbol\_list = pickle.dumps(player\_o.symbol\_list)

s.send(o\_symbol\_list)

# if the player won with the last move or it's a draw, exit the loop

if player\_o.did\_win("O") == True or player\_o.is\_draw() == True:

break

# wait to receive the symbol list and update it

print(f"\nWaiting for other player...")

x\_symbol\_list = s.recv(1024)

x\_symbol\_list = pickle.loads(x\_symbol\_list)

player\_o.update\_symbol\_list(x\_symbol\_list)

# end game messages

if player\_o.did\_win("O") == True:

print(f"Congrats, you won!")

elif player\_o.is\_draw() == True:

print(f"It's a draw!")

else:

print(f"Sorry, the host won.")

# host is being asked for a rematch, awaiting response

print(f"\nWaiting for host...")

host\_response = s.recv(1024)

host\_response = pickle.loads(host\_response)

client\_response = "N"

# if the host wants a rematch, then the client is asked

if host\_response == "Y":

print(f"\nThe host would like a rematch!")

client\_response = input("Rematch? (Y/N): ")

client\_response = client\_response.capitalize()

temp\_client\_resp = client\_response

# let the host know what the client decided

client\_response = pickle.dumps(client\_response)

s.send(client\_response)

# if the client wants a rematch, restart the game

if temp\_client\_resp == "Y":

player\_o.restart()

# if the client said no, then no rematch

else:

rematch = False

# if the host said no, then no rematch

else:

print(f"\nThe host does not want a rematch.")

rematch = False

spacer = input(f"\nThank you for playing!\nPress enter to quit...\n")

s.close()

import socket # for networking

import pickle # for sending/receiving objects

# import the game

from tic\_tac\_toe import TicTacToe

HOST = '127.0.0.1' # the server's IP address

PORT = 12783 # the port we're connecting to

# connect to the host

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

s.connect((HOST, PORT))

print(f"\nConnected to {s.getsockname()}!")

# set up the game

player\_o = TicTacToe("O")

# allow the player to suggest playing again

rematch = True

while rematch == True:

# a header for an intense tic-tac-toe match!

print(f"\n\n T I C - T A C - T O E ")

# draw the grid

player\_o.draw\_grid()

# host goes first, client receives first

print(f"\nWaiting for other player...")

x\_symbol\_list = s.recv(1024)

x\_symbol\_list = pickle.loads(x\_symbol\_list)

player\_o.update\_symbol\_list(x\_symbol\_list)

# the rest is in a loop; if either player has won, it exits

while player\_o.did\_win("O") == False and player\_o.did\_win("X") == False and player\_o.is\_draw() == False:

# draw grid, ask for coordinate

print(f"\n Your turn!")

player\_o.draw\_grid()

player\_coord = input(f"Enter coordinate: ")

player\_o.edit\_square(player\_coord)

# draw grid again

player\_o.draw\_grid()

# pickle the symbol list and send it

o\_symbol\_list = pickle.dumps(player\_o.symbol\_list)

s.send(o\_symbol\_list)

# if the player won with the last move or it's a draw, exit the loop

if player\_o.did\_win("O") == True or player\_o.is\_draw() == True:

break

# wait to receive the symbol list and update it

print(f"\nWaiting for other player...")

x\_symbol\_list = s.recv(1024)

x\_symbol\_list = pickle.loads(x\_symbol\_list)

player\_o.update\_symbol\_list(x\_symbol\_list)

# end game messages

if player\_o.did\_win("O") == True:

print(f"Congrats, you won!")

elif player\_o.is\_draw() == True:

print(f"It's a draw!")

else:

print(f"Sorry, the host won.")

# host is being asked for a rematch, awaiting response

print(f"\nWaiting for host...")

host\_response = s.recv(1024)

host\_response = pickle.loads(host\_response)

client\_response = "N"

# if the host wants a rematch, then the client is asked

if host\_response == "Y":

print(f"\nThe host would like a rematch!")

client\_response = input("Rematch? (Y/N): ")

client\_response = client\_response.capitalize()

temp\_client\_resp = client\_response

# let the host know what the client decided

client\_response = pickle.dumps(client\_response)

s.send(client\_response)

# if the client wants a rematch, restart the game

if temp\_client\_resp == "Y":

player\_o.restart()

# if the client said no, then no rematch

else:

rematch = False

# if the host said no, then no rematch

else:

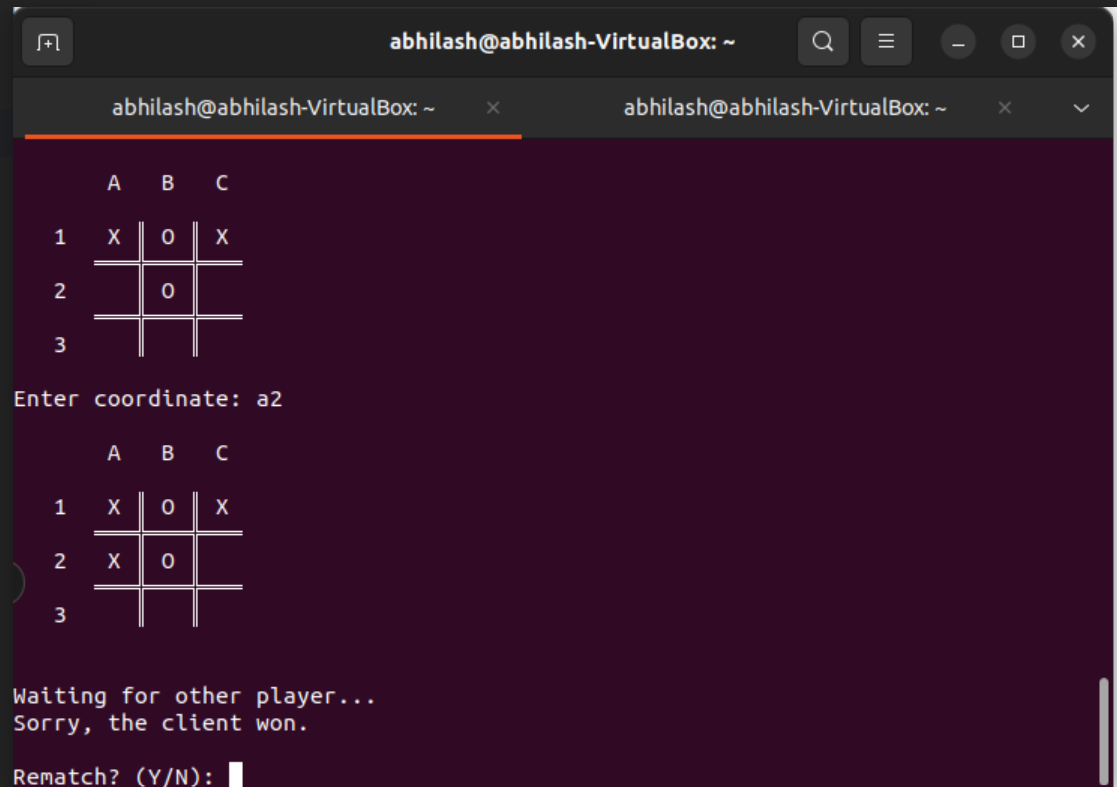
print(f"\nThe host does not want a rematch.")

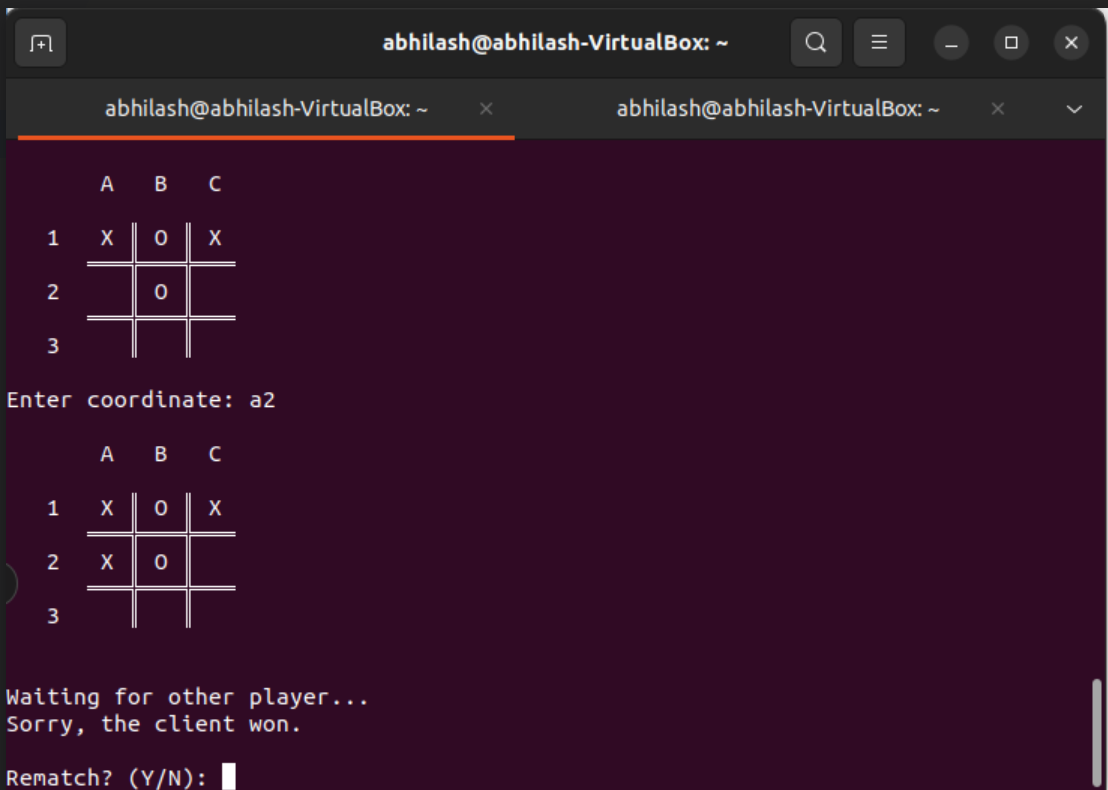
rematch = False

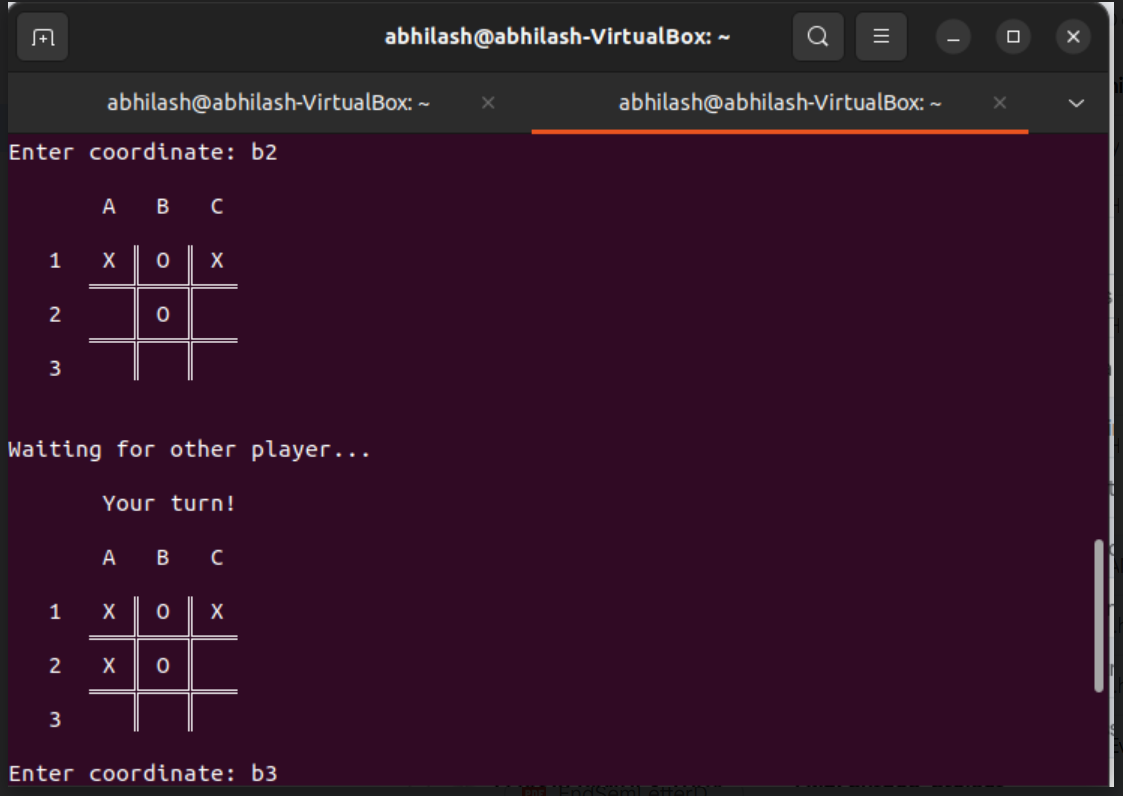
spacer = input(f"\nThank you for playing!\nPress enter to quit...\n")

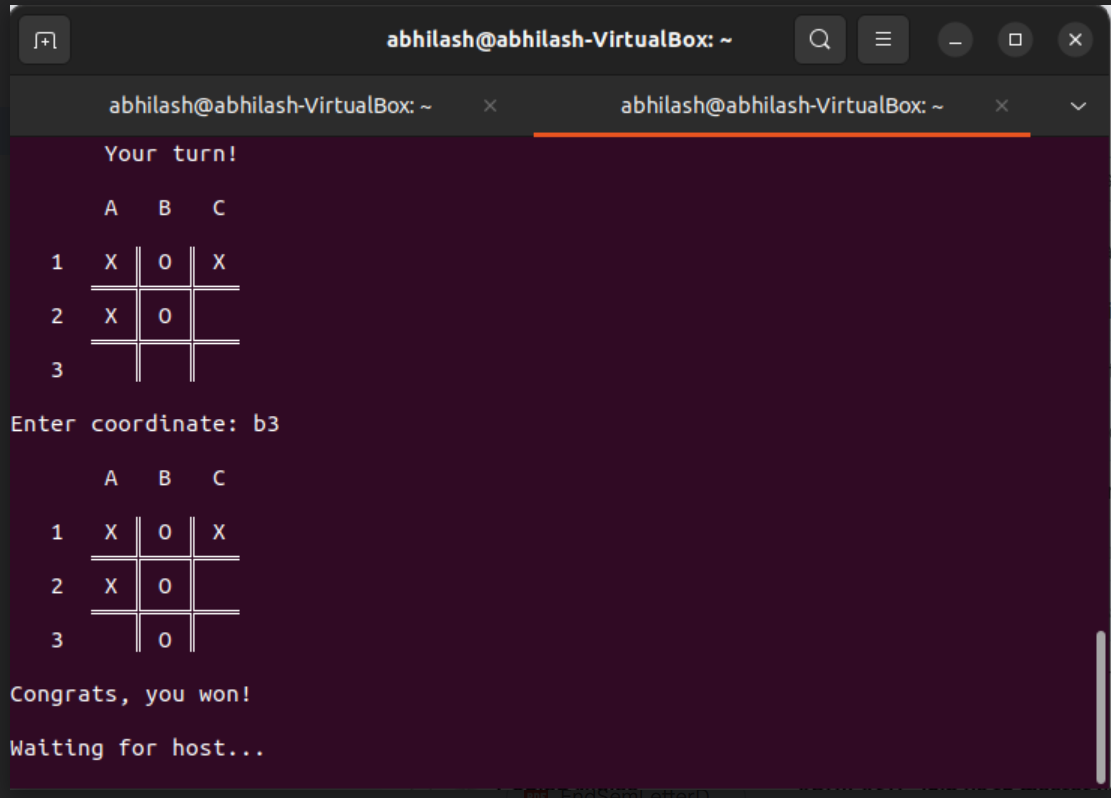
s.close()

**OUTPUT:**

****

****

****

****

**WORKFLOW:**

**1. Server-Side Operations:**

- Creates a socket, binds it to a specific address and port, and listens for incoming connections.

- Accepts client connections and initializes the game session.

- Facilitates the game's logic by coordinating moves between players.

- Manages game state, detects wins or draws, and handles rematch requests.

**2. Client-Side Operations:**

- Connects to the server using socket communication.

- Participates in the game by interacting with the server.

- Sends moves to the server and receives game updates.

- Responds to prompts for rematch requests.

**CHALLENGES FACED:**

* **Socket Communication:** Understanding and implementing socket communication for exchanging game data between the server and clients.
* **Game Logic Implementation:** Ensuring accurate game state management, win-draw detection, and coordinating moves between players.

**CONCLUSION:**

The project successfully demonstrated the implementation of a simple Tic-Tac-Toe game using socket programming in Python. It showcased the practical application of networking concepts learned in the Computer Networks coursework.

**FUTURE WORK:**

- **User Interface:** Incorporating a graphical user interface (GUI) for a more interactive gaming experience.

- **Security Measures:** Implementing secure communication channels for sensitive data exchange.

- **Scalability:** Extending the game to support multiple simultaneous game sessions or additional features.