Implementation of Connect 4 using TCP

Report submitted to the SASTRA Deemed to be University

as the requirement for the course

CSE302: COMPUTER NETWORKS

Submitted by

THARIK SHERIEF B

(Reg. No.: 124015109 B. Tech – Information Technology)

December 2022

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

SCHOOL OF COMPUTING

THANJAVUR – 613 401

Bona-fide Certificate

This is to certify that the report titled “**Implementation of Connect 4 using TCP**” submitted as a requirement for the course, **CSE302: COMPUTER NETWORKS** for B. Tech is a bona-fide record of the work done by **Shri. Tharik Sherief (Reg. No 124015109, B. Tech Information Technology)** during the academic year 2022-23, in the School of Computing.

Project Based Work *Viva voce* held on ________________

Examiner1 **Examiner2**

LIST OF FIGURES

ABBREVIATIONS

TCP	Transmission Control Protocol
UDP	User Datagram Protocol
GUI	End-to-End Encrypted
IP	Internet Protocol
LAN	Local Area Network

ABSTRACT

Online multiplayer games have always been good entertainment and an interesting mode of communication. Online multiplayer games have connected various people across countries during these quarantine times. Multiplayer online games are gaining more users recently due to the pandemic. Games like “Apex legends”, “Call of Duty”, “Valorant” have been more popular since the beginning of the pandemic. Even most of the multiplayer board games are hitting the internet. Online games are mostly played for chatting, making friends and staying socially connected. Even research says 70% of online games are played with friends. This kind of socialising was called “online socialization”

On the early 1970s, online multiplayer games started emerging. Initially it was LAN connected games. The moves made by the player are sent to a local server and are written back to the other player. The inter-system communication is done by “Socket programming”. The socket programming requires an IP address and a port to connect to a server or a client system. It can use either TCP or UDP protocol in the transport layer. But currently all popular gaming companies have servers across the globe, which enables players to play across countries. The computations are done on servers for complex games, to maintain the uniformity of the game.

Connect 4 is a turn-based game, which can come in variety of grid sizes. The player succeeds in placing 4 of their color coins in a diagonal horizontal or vertical row is the winner. The application is developed to program using sockets in a distributed computing environment based on the basic Client-Server model. The coding was done using Python and the GUI was built using PyGame module . Demonstration of the principles and concepts of socket programming was done in this project. The socket libraries available in python were also learnt and implemented.

KEY WORDS: Connect 4 Game, TCP, Client-Server model, Python, Socket programming.

TABLE OF CONTENTS

1. INTRODUCTION

Computer Networks are formed by the grouping of several computers. In computer networks, the formation of Client Server models is an integral part. We use the client server architecture in this project to build the Connect 4 game where any number of participants can join the quiz as individual threads.

When creating a client-server model, any computer terminal in the network can be made a client or a server. The terminals that create requests and initiate communications are clients of the model. On the other hand, the terminal that responds to client's requests in the model is the server. Every client communicates and connects with each through server.

In this project, we create the Connect 4 Game based on the Transmission Control Protocol (TCP). The TCP is used for the connection between Participants and Host and in the end, Multithreading is used to develop the application.

This Connect 4 System has a host Server and several participant Clients. The communication takes place between them in both the ways. The server is to handle the logic of the game and send it to the client. Each participant receives these questions through the server. The scores and ranks of the participants is then generated by the Server and then displayed to the participant once the exam gets over.

The server starts on a designated computer before the clients and waits for the clients to join the network. When two clients get connected a new thread is created and game starts. The server receives each position the coin is placed from the clients and handles the logic and communication between the two clients.

1.1 CLIENT SERVER MODEL

A model in which the server provide network services to other computers. A system can act as a Server and as a Client simultaneously. This happens when both the client and server process resides in the same machine.

Client Program: The application program that runs in the local machine and requests for services from the other programs running on other remote machine. The client program runs only when there is need for services and thus runs to request for the service from the server. Client programs start when there is service request from user and terminates when the service is over, thus it is a finite program.

Server Program: The application program that provides services for either one or many clients in the model. Client-server follows many to one relationship. The server program runs all the time. It is because; it does not know when a service is required. The server program is an infinite program. It starts runs and waits infinitely for the incoming requests from clients. It responds to the requests whenever raised.

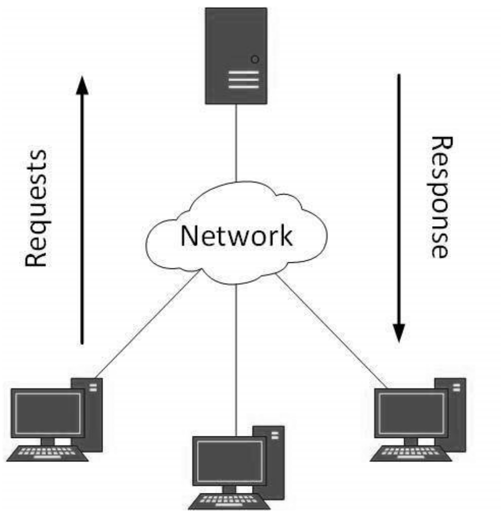


Fig. 1.1.Structure of a Client-Server model

1.1.1 Advantages of Client Server Model:

- All the data is stored in a server. Thus, there is a centralized system with all data in a single place. Data recovery is possible.
- The model is scalable, i.e., the number of clients and servers can be increased separately. Any new node can be added into the network at any time.
- The presence of a dedicated server results in the increase of speed in the sharing of resources. Thus as a result, there is an increase in the performance of the overall system as well.
- Has a less maintenance cost.

1.1.2 Disadvantages of Client Server Model:

- Lacks in robustness. Whenever the server is down, the requests cannot be met by the server and the thus the entire model is down.
- The entire model is prone to viruses, Trojans, if any of this is present or uploaded into the server.
- Spoofing and modification of Data packets is possible during the transmission.

1.2 COMMUNICATION PROTOCOL

The request-response messaging pattern has to follow a common communication protocol. The rules, dialog pattern, language to be used are formally defined by the protocols. The commonly used protocols in socket programming are the datagram communication and stream communication.

The datagram communication is considered a connectionless protocol and is known as the UDP. In UDP, every time the client sends a datagram, it is also required for it to send the local socket descriptor and as well the address of the receiving socket.

The stream communication is a connection oriented protocol and is known as the TCP. In TCP, firstly a connection has to be established between two identified sockets in order to establish a communication. The server socket listens for the connection request and the other client socket asks for connection. The transmission of data is possible in both the ways only when they have an already established connection.

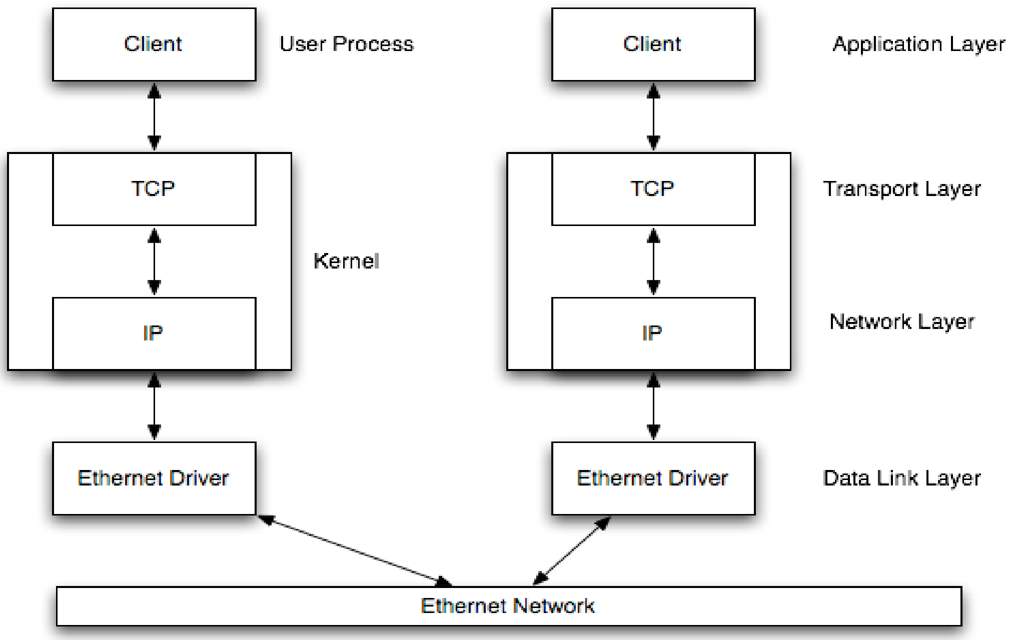


Fig. 1.2.Client and Server communicating using TCP

1.3 SOCKET PROGRAMMING

In a client-server application, the communication link between the client and server programs is called a socket. They are the end point of a two-way communication. The application process sends and receives messages from other application process via the socket. They allow processes lying on different machines or on the same machine to communicate.

The client server program allows user to run the client software to create a query. The client connects and sends query to the identified server through the use of socket (client side). The server once received the request from the client analyses the query and sends the result to the client. Socket is a combination of the port number and the IP address.

The commonly used types of sockets are Stream Sockets and Datagram Sockets. The stream sockets use TCP for data transmission whereas, the datagram sockets use UDP. In this project only Stream sockets have been used.

1.3.1 TCPSocket API

The Connect 4 game was designed using socket programming in Python supported by TCP datagram. The client and server programs were simulated, demonstrated and analyzed.

The steps for establishing a TCP socket on the server side:

- Create a socket using the **socket()** function.
- Bind the socket to an address using the **bind()** function.
- Listen for connections using the **listen()** function.
- Accept a connection using the **accept()** function. This function blocks until client connections are made with the server.
- Send and receive data using **send()** and **recv()** functions.

The steps for establishing a TCP socket on the Client side:

- Create a socket using the **socket()** function.
- Connect the socket to the address of the server using **connect()** function.
- Send and receive data using **send()** and **recv()** functions.

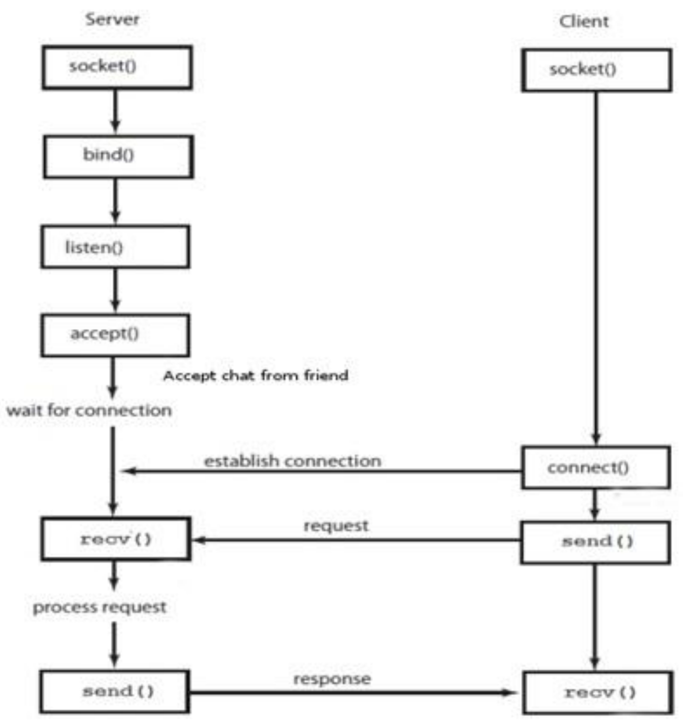


Fig. 1.3.Python Socket Programming Workflow

1.4 MULTITHREADED CONNECT 4 USING TCP

This application is a basic combination of a server application and client application. Server application is made to run on the server computer. The client application is made to run on each of the player's computer. The server socket is created and attached to a port. There, the server listens for connections and waits till two players are connected.

At the server side, a thread is created for the two clients. When the two players are connected, one player can place their coins. The first player after clicking on the column to place their coin, the client send the data to the server telling which column the player their coin. The server receives and retransmits the data to the second player. The above happens till player has won the game.

1.5 DEMERITS

The major demerit of this model is the non-robust nature of the server. Here, the server has to be maintained very well to have the model running well. Whenever the server is down the entire network model is down. Also, the processing power of the server has to be high to support the large number of clients getting connected into the network. When a huge number of clients get connected to the network, the server has to process a large number of threads each one carrying the process of an individual client. Information security can also be an issue here. Spoofing and modification of data may be possible in the application model.

2. SOURCE CODE

The source code consists of python files.

1. server.py
2. client.py
3. logic.py (helper functions for handling logic of the game)

2.1 server.py

```
import socket
import threading

from logic import *

PORT = 9999
SERVER = socket.gethostbyname(socket.gethostname())
ADDR = (SERVER, PORT)

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(ADDR)

waiting_clients = []

def handle_game(conn1:socket, conn2:socket):
    try:
        board = create_board()
        conn1.send("1".encode())
        conn2.send("2".encode())
        while True:
            player1Move = int(conn1.recv(1024).decode())
            row = get_next_open_row(board, player1Move)
            drop_piece(board, row, player1Move, 1)
            conn2.send(str(player1Move).encode())
            if winning_move(board, 1):
                print(f"{conn1.getpeername()} has won the game with {conn2.getpeername()}")
                break

            player2Move = int(conn2.recv(1024).decode())
            row = get_next_open_row(board, player2Move)
            drop_piece(board, row, player2Move, 2)
            conn1.send(str(player2Move).encode())
            if winning_move(board, 2):
```

```
            print(f"{conn2.getpeername()} has won the game with {conn1.getpeername()}")
            break
    except Exception as e:
        print(e)
        print("Connection closed")
    conn1.close()
    conn2.close()


def start():
    server.listen()
    print(f"[LISTENING Server is listening on {SERVER}]")
    while True:
        conn, addr = server.accept()
        print(f"Got connection from {addr}.")
        waiting_clients.append(conn)
        if len(waiting_clients) == 2:
            conn1,conn2 = waiting_clients.pop(),waiting_clients.pop()
            print(f"Created game b/w {conn1.getpeername()} and {conn2.getpeername()}")
            thread = threading.Thread(target=handle_game, args=(conn1,conn2))
            thread.start()
start()
```

2.2 client.py

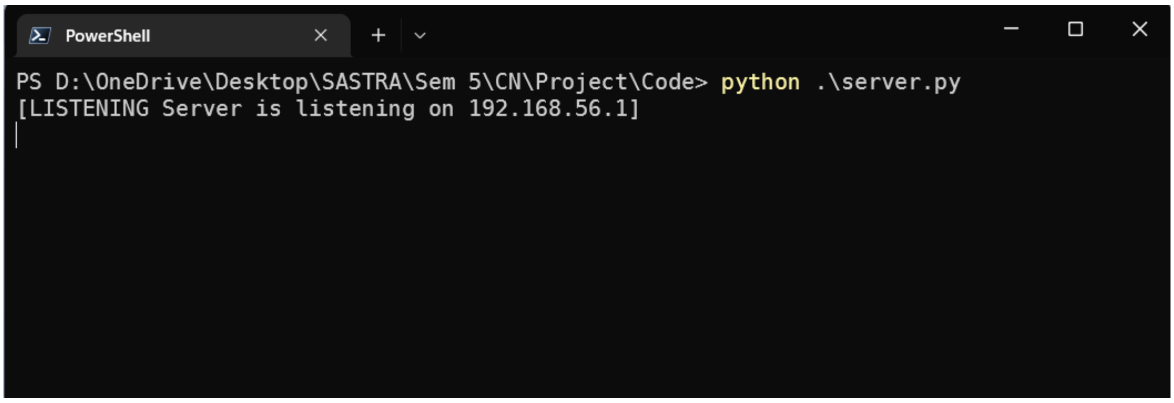
```
import math
import socket
import sys
from threading import Thread
import pygame
import pygame_menu
from logic import *

# Colors
BOARD_BACKGROUND = (0, 0, 255)
BACKROUND_COLOR = (135, 206, 235)
PLAYER1_COLOR = (255, 0, 0)
PLAYER2_COLOR = (255, 255, 0)

# Game Logic Variables
game_over = False
ROW_COUNT = 6
COLUMN_COUNT = 7
```

```
turn = 0
mutex = 0 # 1 for waiting, 0 for not waiting, 1 for got result

# Variables used for rendering
SQUARESIZE = 100
width = COLUMN_COUNT * SQUARESIZE
height = (ROW_COUNT + 1) * SQUARESIZE
size = (width, height)
RADIUS = int(SQUARESIZE / 2 - 5)

# Network Variables
conn = None
op=0 # variable for storing incoming data

pygame.init()
surface = pygame.display.set_mode(size)
font = pygame.font.SysFont("monospace", 50)

# Draws the board for the game using pygame module.draw functions
def draw_board(board):
    for c in range(COLUMN_COUNT):
        for r in range(ROW_COUNT):
            pygame.draw.rect(
                surface,
                BOARD_BACKGROUND,
                (c * SQUARESIZE, r * SQUARESIZE + SQUARESIZE, SQUARESIZE,
SQUARESIZE),
            )
            pygame.draw.circle(
                surface,
                BACKROUND_COLOR,
                (
                    int(c * SQUARESIZE + SQUARESIZE / 2),
                    int(r * SQUARESIZE + SQUARESIZE + SQUARESIZE / 2),
                ),
                RADIUS,
            )

    for c in range(COLUMN_COUNT):
        for r in range(ROW_COUNT):
            if board[r][c] == 1:
                pygame.draw.circle(
                    surface,
                    PLAYER1_COLOR,
                    (
                        int(c * SQUARESIZE + SQUARESIZE / 2),
                        height - int(r * SQUARESIZE + SQUARESIZE / 2),
                    ),
                    RADIUS,
                )
            elif board[r][c] == 2:
                pygame.draw.circle(
                    surface,
                    PLAYER2_COLOR,
                    (
                        int(c * SQUARESIZE + SQUARESIZE / 2),
                        height - int(r * SQUARESIZE + SQUARESIZE / 2),
                    ),
                    RADIUS,
                )
    pygame.display.update()

# Render text on top of the game
def render_txt(txt, color=(0, 0, 0)):
    pygame.draw.rect(surface, BACKROUND_COLOR, (0, 0, width, SQUARESIZE))
    label = font.render(txt, 1, color)
    surface.blit(label, (20, 10))
    pygame.display.update()

def get_other_user_turn(conn):
    global op, mutex
    mutex = 1
    op = int(conn.recv(1024).decode())
    print("Recieved input from Server : ", op)
    mutex = 2

def start_game():
    global surface, board, player, conn, game_over, mutex, turn, op
    board = create_board()
    draw_board(board)
    pygame.display.update()
    while not game_over:
        # Wait for other player input
        if  turn != (player - 1):
            render_txt("Other player's turn")
            if mutex == 0:
                thread = Thread(target=get_other_user_turn, args=(conn,))
                thread.start()
            elif mutex == 2:
                col = op
                row = get_next_open_row(board, col)
                drop_piece(board, row, col, turn + 1)
                draw_board(board)
                render_txt("")
                if winning_move(board, turn + 1):
                    render_txt("You lost :(", PLAYER1_COLOR)
                    game_over = True
                turn = (turn+1) % 2
                mutex = 0

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                sys.exit()

            if turn == (player - 1):
                if event.type == pygame.MOUSEMOTION:
                    pygame.draw.rect(
                        surface, BACKROUND_COLOR, (0, 0, width, SQUARESIZE)
                    )
                    posx = event.pos[0]
                    if player == 1:
                        pygame.draw.circle(
                            surface, PLAYER1_COLOR, (posx, int(SQUARESIZE / 2)), RADIUS
                        )
                    else:
                        pygame.draw.circle(
                            surface, PLAYER2_COLOR, (posx, int(SQUARESIZE / 2)), RADIUS
                        )
                    pygame.display.update()

                if event.type == pygame.MOUSEBUTTONDOWN:
                    pygame.draw.rect(
                        surface, BACKROUND_COLOR, (0, 0, width, SQUARESIZE)
                    )
                    # Local Player user input
                    posx = event.pos[0]
                    col = int(math.floor(posx / SQUARESIZE))
                    if is_valid_location(board, col):
                        row = get_next_open_row(board, col)
                        drop_piece(board, row, col, player)
                        draw_board(board)
                        print("Sending ", col)
                        conn.send(str(col).encode())
                        if winning_move(board, player):
                            render_txt("You win!!", PLAYER1_COLOR)
                            print("You win!!")
                            game_over = True
                        turn = (turn+1) % 2

        if game_over:
            pygame.time.wait(5000)

def play():
    global conn, player
    conn = socket.socket()
    ip = clientGUI.get_widget("ip").get_value()
    port = int(clientGUI.get_widget("port").get_value())
    print("Connecting to ip {} and in port {}".format(ip, port))
    clientGUI.get_widget("label").set_title("Connecting to server. Please wait....")
    pygame.display.update()
    try:
        conn.connect((ip, port))
        player = int(conn.recv(1024).decode())  # Gets player / turn no
        print("Player ID :", player)
        start_game()
        clientGUI.get_widget("label").set_title("")
    except Exception as e:
        print(e)
        clientGUI.get_widget("label").set_title("Can't connect")
    conn.close()

pygame.display.set_caption("Connect 4 Game")
clientGUI = pygame_menu.Menu("Client Menu", size[0], size[1],
theme=pygame_menu.themes.THEME_DARK)
clientGUI.add.label("", label_id="label")
clientGUI.add.text_input("IP to connect :", default="192.168.56.1", textinput_id="ip")
clientGUI.add.text_input(
    "Port to connect :",
    default=9999,
    type="input-int",
    maxchar=5,
    textinput_id="port",
)
clientGUI.add.button("Play", play)
clientGUI.add.button("Quit", pygame_menu.events.EXIT)
clientGUI.mainloop(surface)
```

2.3 logic.py

```
import numpy as np

ROW_COUNT = 6
COLUMN_COUNT = 7

def create_board():
    board = np.zeros((ROW_COUNT, COLUMN_COUNT))
    return board

def drop_piece(board, row, col, piece):
    board[row][col] = piece

def is_valid_location(board, col):
    return board[ROW_COUNT - 1][col] == 0

def get_next_open_row(board, col):
    for r in range(ROW_COUNT):
        if board[r][col] == 0:
            return r

def winning_move(board, piece):
    # Check horizontal locations for win
    for c in range(COLUMN_COUNT - 3):
        for r in range(ROW_COUNT):
            if (
                board[r][c] == piece
                and board[r][c + 1] == piece
                and board[r][c + 2] == piece
                and board[r][c + 3] == piece
            ):
                return True

    # Check vertical locations for win
    for c in range(COLUMN_COUNT):
        for r in range(ROW_COUNT - 3):
            if (
                board[r][c] == piece
                and board[r + 1][c] == piece
                and board[r + 2][c] == piece
                and board[r + 3][c] == piece
            ):
                return True

    # Check positively sloped diaganols
    for c in range(COLUMN_COUNT - 3):
        for r in range(ROW_COUNT - 3):
            if (
                board[r][c] == piece
                and board[r + 1][c + 1] == piece
                and board[r + 2][c + 2] == piece
                and board[r + 3][c + 3] == piece
            ):
                return True

    # Check negatively sloped diaganols
    for c in range(COLUMN_COUNT - 3):
        for r in range(3, ROW_COUNT):
            if (
                board[r][c] == piece
                and board[r - 1][c + 1] == piece
                and board[r - 2][c + 2] == piece
                and board[r - 3][c + 3] == piece
            ):
                return True
```

3. SCREENSHOTS:



```
PS D:\OneDrive\Desktop\SASTRA\Sem 5\CN\Project\Code> python .\server.py
[LISTENING Server is listening on 192.168.56.1]
```

Fig. 3.1.Server terminal listening

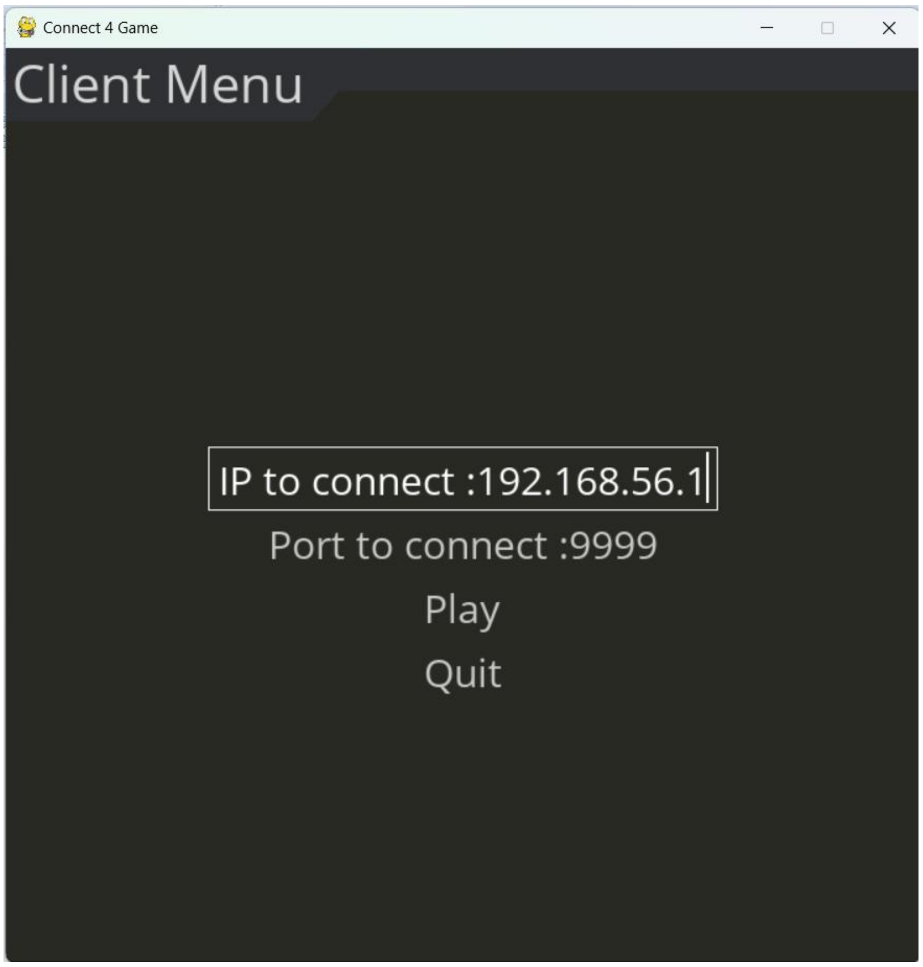
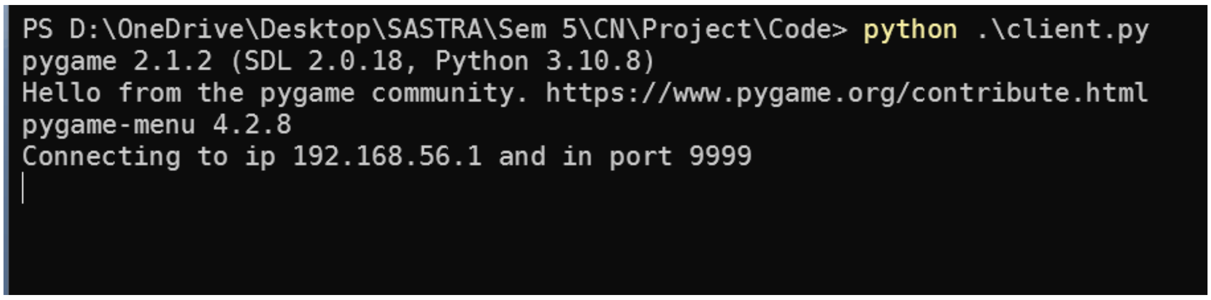
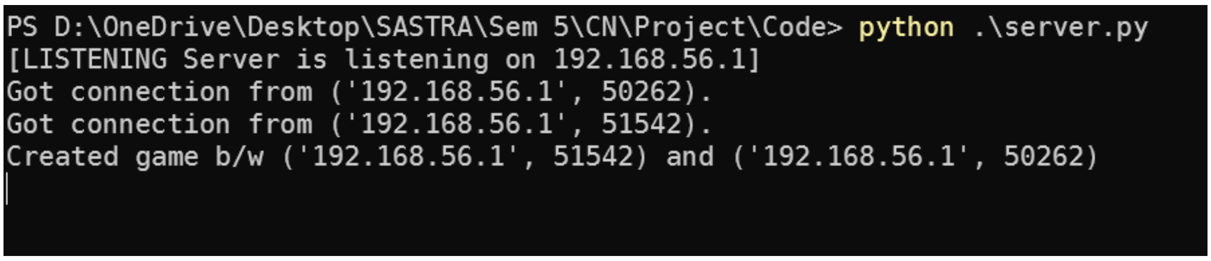


Fig. 3.2.Client menu



```
PS D:\OneDrive\Desktop\SASTRA\Sem 5\CN\Project\Code> python .\client.py
pygame 2.1.2 (SDL 2.0.18, Python 3.10.8)
Hello from the pygame community. https://www.pygame.org/contribute.html
pygame-menu 4.2.8
Connecting to ip 192.168.56.1 and in port 9999
```

Fig. 3.3.Client Terminal connecting to server



```
PS D:\OneDrive\Desktop\SASTRA\Sem 5\CN\Project\Code> python .\server.py
[LISTENING Server is listening on 192.168.56.1]
Got connection from ('192.168.56.1', 50262).
Got connection from ('192.168.56.1', 51542).
Created game b/w ('192.168.56.1', 51542) and ('192.168.56.1', 50262)
```

Fig. 3.4.Server Terminal after game starts

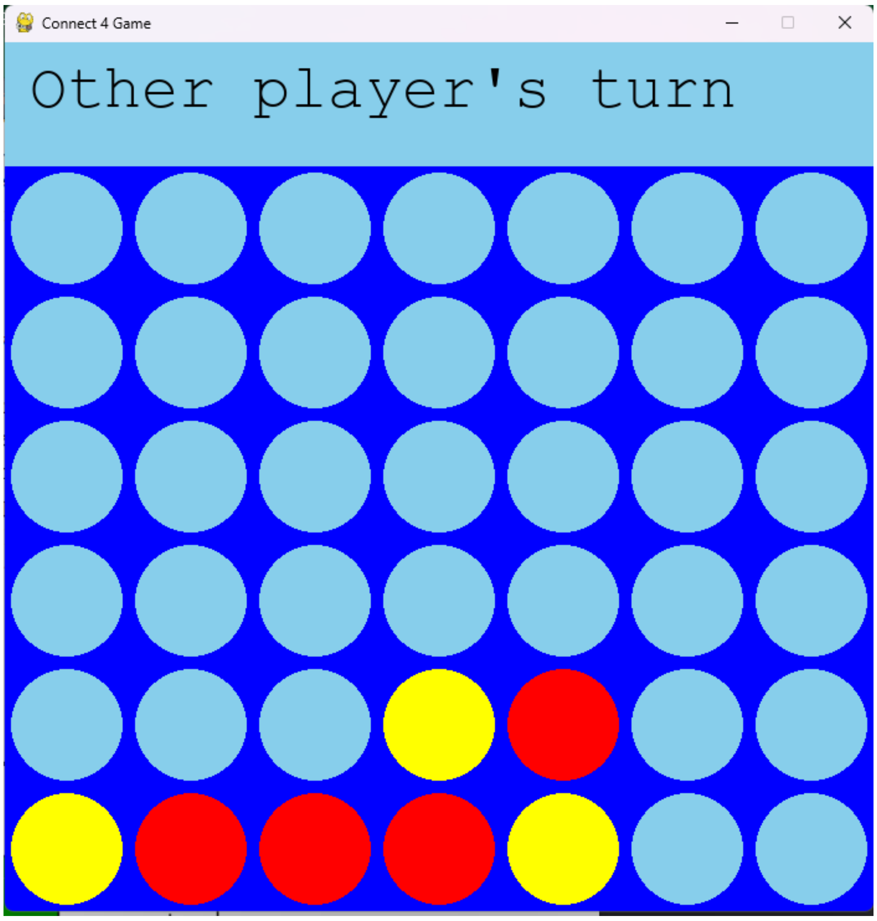
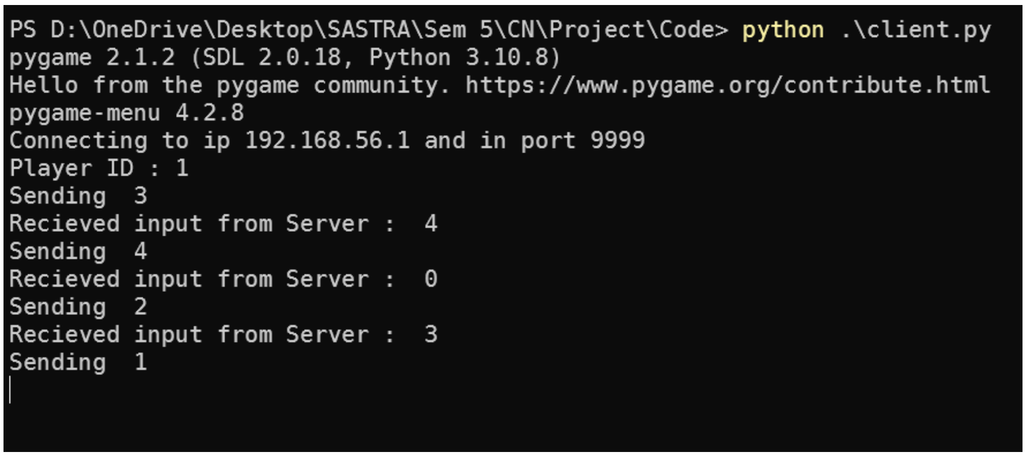
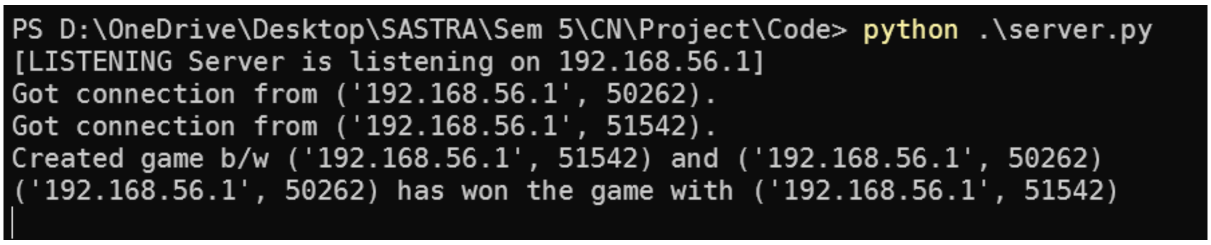


Fig. 3.5.Client waiting for other player move from server



```
PS D:\OneDrive\Desktop\SASTRA\Sem 5\CN\Project\Code> python .\client.py
pygame 2.1.2 (SDL 2.0.18, Python 3.10.8)
Hello from the pygame community. https://www.pygame.org/contribute.html
pygame-menu 4.2.8
Connecting to ip 192.168.56.1 and in port 9999
Player ID : 1
Sending  3
Recieved input from Server :  4
Sending  4
Recieved input from Server :  0
Sending  2
Recieved input from Server :  3
Sending  1
```

Fig. 3.6.Client Terminal when game in progress



```
PS D:\OneDrive\Desktop\SASTRA\Sem 5\CN\Project\Code> python .\server.py
[LISTENING Server is listening on 192.168.56.1]
Got connection from ('192.168.56.1', 50262).
Got connection from ('192.168.56.1', 51542).
Created game b/w ('192.168.56.1', 51542) and ('192.168.56.1', 50262)
('192.168.56.1', 50262) has won the game with ('192.168.56.1', 51542)
```

Fig. 3.7.Terminal of the Server completion of a game

4. CONCLUSION

The application was run successfully and handled more than 1 game at a time. It was clearly seen that the clients were able to play the game. The submitted moves are handled by the server and sent to the other player. The board also tells who the winner is. The detailed implementation of socket programming has helped in an easier run in this application. Thus, the various socket libraries were studied and implemented. The real world simulation and use of socket programming was well analyzed and illustrated. This application could be further developed by the implementation of techniques to maintain the information security while storing and transmitting data packets can also be done. Embedding of other functionalities can also be done in the future for updating a new version of this application.

5. REFERENCES

1. http://gaia.cs.umass.edu/kurose_ross/online_lectures.htm *Computer Networking: A Top-down Approach - 8th Edition* **Jim Kurose,Keith Ross** Authors’ website.
2. **Kurose, James F., and Keith W. Ross.** *Computer Networking: A Top-Down Approach*. 2021.
3. https://www.cs.dartmouth.edu/~campbell/cs60/socketprogramming.html/ - **Andrew T. Campbell’s** *CS 60 Computer Networks Lecture 3 and 4 : Socket Programming*
4. https://www.tutorialspoint.com/unix_sockets/what_is_socket.html - Socket Programming
5. https://www.javatpoint.com/computer-network-client-and-server-model - ClientServer model
6. **Maata, Rolou Lyn & Cordova, Ronald & Sudramurthy, Balaji & Halibas, Alrence**. (2017). *Design and Implementation of Client-Server Based Application Using Socket Programming in a Distributed Computing Environment.* 10.1109/ICCIC.2017.8524573.
7. https://www.techbeamers.com/python-tutorial-essentials-of-python-socket- programming/ - Python Socket programming
8. https://realpython.com/python-sockets/ - Python Socket programming