Multiplayer Tic-Tac-Toe: A Client-Server Project
CCN COURSE PROGRAMMING ASSIGNMENT REPORT

Submitted by

ARUN KUMAR SUTHARI
21MEB0B62
Mechanical Engineering

1. Description:

The project is a server-client-based implementation of a Tic-Tac-Toe game using Python. The server handles multiple client connections, maintains game states, and facilitates communication between players. The server GUI, built with tkinter, displays key information like the number of active players, ongoing games, and server messages, allowing the server administrator to monitor and manage the game status.

2. Design of modules and algorithms using pseudocode

❖ **Server Initialization:**

- Initialize server socket and start listening on a given port.
- Create GUI with labels for "Active Players" and "Ongoing Games".
- Start a background thread to handle new connections.
- Display logs and update GUI dynamically as players join or games are created.

❖ **connection Handling:**

while True:
  Wait for two client connections.
  Assign player IDs and notify players of their role.
  Start a new thread to run the Tic-Tac-Toe game with these two clients**.**

❖ **Game Loop**

Initialize an empty Tic-Tac-Toe board.
While game is not over:
  Send “WAT” to the waiting player.
  Prompt the current player for a move.
  Validate move:
    If valid, update the board and notify both players.
    If invalid, prompt again.
  Check if current move results in a win or draw.
  Update game state and notify clients accordingly.

❖ **Move Validation**

For a move to be valid:
  Move position should be within bounds (0–8).
  The chosen cell on the board must be empty.

❖ **Winning Condition Check**

For a win, check:
  - If any row, column, or diagonal has three matching symbols.
If no winner and all moves are made:
  Declare a draw.

3. The Python program code

- **Server.py**

```
import socket
import threading
import struct
import tkinter as tk
from tkinter import messagebox

# Global player count with thread lock
player_count = 0
mutex = threading.Lock()

class TicTacToeServer:
    def __init__(self, port):
        self.port = port
        self.server_socket = None
        self.setup_gui()

    def setup_gui(self):
        self.root = tk.Tk()
        self.root.title("Tic-Tac-Toe Server")
        self.root.geometry("400x300")
        self.root.configure(bg="lightgray")

        # Server title label
        self.title_label = tk.Label(
            self.root, text="Tic-Tac-Toe Server - ARUN KUMAR", font=("Arial",
16, "bold"), bg="lightgray"
        )
        self.title_label.pack(pady=10)

        # Player count and server status labels
        self.info_frame = tk.Frame(self.root, bg="lightgray")
        self.info_frame.pack(pady=10)

        self.player_count_label = tk.Label(self.info_frame, text="Active
Players: 0", font=("Arial", 12), bg="lightgray")
        self.player_count_label.grid(row=0, column=0, padx=10, pady=5)

        self.game_count_label = tk.Label(self.info_frame, text="Ongoing Games:
0", font=("Arial", 12), bg="lightgray")
        self.game_count_label.grid(row=1, column=0, padx=10, pady=5)

        # Status message box
        self.status_box = tk.Text(self.root, height=10, width=40,
state="disabled", bg="white")
```

```
        self.status_box.pack(pady=10)

        # Start server in a separate thread
        threading.Thread(target=self.run_server).start()

        # Run the GUI
        self.root.mainloop()

    def log_message(self, message):
        self.status_box.config(state="normal")
        self.status_box.insert("end", f"{message}\n")
        self.status_box.config(state="disabled")
        self.status_box.see("end")

    def update_active_players(self, count):
        self.player_count_label.config(text=f"Active Players: {count}")

    def update_ongoing_games(self, count):
        self.game_count_label.config(text=f"Ongoing Games: {count}")

    def run_server(self):
        global player_count
        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server_socket.bind(('', self.port))
        self.server_socket.listen(5)
        self.log_message(f"Server listening on port {self.port}.")

        game_count = 0
        while True:
            # Accept new connections and handle players
            clients = self.get_clients()
            game_count += 1
            self.update_ongoing_games(game_count // 2)
            game_thread = threading.Thread(target=self.run_game,
args=(clients,))
            game_thread.start()

    def get_clients(self):
        clients = []
        global player_count
        while len(clients) < 2:
            conn, addr = self.server_socket.accept()
            with mutex:
                player_count += 1
                self.update_active_players(player_count)
                self.log_message(f"Player {player_count} connected from
{addr}.")
            clients.append(conn)
```

```
            self.write_client_int(conn, len(clients) - 1)  # Send client ID (0
or 1)
        return clients

    def run_game(self, clients):
        board = [[' ' for _ in range(3)] for _ in range(3)]
        self.log_message("Game started between two players.")
        player_turn = 0
        game_over = False
        turn_count = 0

        while not game_over:
            self.write_client_msg(clients[(player_turn + 1) % 2], "WAT")
            valid_move = False
            move = -1

            while not valid_move:
                move = self.get_player_move(clients[player_turn])
                if move == -1:
                    self.log_message("Player disconnected.")
                    return
                if self.check_move(board, move):
                    valid_move = True
                else:
                    self.write_client_msg(clients[player_turn], "INV")

            self.update_board(board, move, player_turn)
            self.send_update(clients, move, player_turn)

            if turn_count >= 4 and self.check_board(board, move):
                game_over = True
                self.write_client_msg(clients[player_turn], "WIN")
                self.write_client_msg(clients[(player_turn + 1) % 2], "LSE")
                self.log_message(f"Player {player_turn} won.")
            elif turn_count == 8:
                game_over = True
                self.write_clients_msg(clients, "DRW")
                self.log_message("Game ended in a draw.")

            player_turn = (player_turn + 1) % 2
            turn_count += 1

        for conn in clients:
            conn.close()
        global player_count
        with mutex:
            player_count -= 2
            self.update_active_players(player_count)
```

```
    def check_move(self, board, move):
        return 0 <= move <= 8 and board[move // 3][move % 3] == ' '

    def update_board(self, board, move, player_id):
        board[move // 3][move % 3] = 'X' if player_id == 1 else 'O'

    def send_update(self, clients, move, player_id):
        self.write_clients_msg(clients, "UPD")
        self.write_clients_int(clients, player_id)
        self.write_clients_int(clients, move)

    def check_board(self, board, last_move):
        row = last_move // 3
        col = last_move % 3
        player_mark = board[row][col]

        if all(board[row][i] == player_mark for i in range(3)) or
all(board[i][col] == player_mark for i in range(3)):
            return True
        if last_move % 2 == 0:
            return all(board[i][i] == player_mark for i in range(3)) or
all(board[i][2 - i] == player_mark for i in range(3))
        return False

    def write_client_int(self, conn, msg):
        try:
            conn.sendall(struct.pack("!I", msg))
        except Exception as e:
            self.log_message(f"Error writing int to client socket: {e}")

    def write_client_msg(self, conn, msg):
        try:
            conn.sendall(msg.encode())
        except Exception as e:
            self.log_message(f"Error writing msg to client socket: {e}")

    def write_clients_msg(self, clients, msg):
        for client in clients:
            self.write_client_msg(client, msg)

    def write_clients_int(self, clients, msg):
        for client in clients:
            self.write_client_int(client, msg)

    def get_player_move(self, conn):
        self.write_client_msg(conn, "TRN")
        return self.recv_int(conn)
```

```
    def recv_int(self, conn):
        try:
            data = conn.recv(4)
            if not data:
                return -1
            return struct.unpack("!I", data)[0]
        except:
            return -1

def main(port):
    server = TicTacToeServer(port)

if __name__ == "__main__":
    import sys
    if len(sys.argv) < 2:
        print("Error: No port provided.")
        sys.exit(1)
    main(int(sys.argv[1]))
```

- **client.py**

```
import socket
import threading
import struct
import tkinter as tk
from tkinter import messagebox, simpledialog

class TicTacToeClient:
    def __init__(self, hostname, port):
        self.hostname = hostname
        self.port = port
        self.player_id = None
        self.game_number = 1  # Initialize game number
        self.board = [[' ' for _ in range(3)] for _ in range(3)]
        self.sock = None
        self.my_turn = False  # Track if it's this player's turn
        self.create_gui()

    def create_gui(self):
        # Initialize main window
        self.root = tk.Tk()
        self.root.title("Tic-Tac-Toe: ARUN KUMAR")
        self.root.geometry("400x500")
        self.root.configure(bg="lightblue")

        # Title label with your name and game information
        self.title_label = tk.Label(
```

```
            self.root, text="Tic-Tac-Toe Game - ARUN KUMAR (21MEB0B62)",
            font=("Arial", 16, "bold"), bg="lightblue"
        )
        self.title_label.pack(pady=10)

        # Player and game info labels
        self.info_frame = tk.Frame(self.root, bg="lightblue")
        self.info_frame.pack(pady=5)

        self.player_label = tk.Label(self.info_frame, text="Player:
Waiting...", font=("Arial", 12), bg="lightblue")
        self.player_label.grid(row=0, column=0, padx=20)

        self.game_label = tk.Label(self.info_frame, text=f"Game No:
{self.game_number}", font=("Arial", 12), bg="lightblue")
        self.game_label.grid(row=0, column=1, padx=20)

        # Game board buttons
        self.buttons = [[None for _ in range(3)] for _ in range(3)]
        self.board_frame = tk.Frame(self.root, bg="lightblue")
        self.board_frame.pack(pady=10)

        for i in range(3):
            for j in range(3):
                btn = tk.Button(self.board_frame, text="", font=("Arial", 24),
width=5, height=2,
                                command=lambda i=i, j=j: self.send_move(i, j),
state="disabled")
                btn.grid(row=i, column=j, padx=5, pady=5)
                self.buttons[i][j] = btn

        # Status label
        self.status_label = tk.Label(self.root, text="Connecting to
server...", font=("Arial", 12), bg="lightblue")
        self.status_label.pack(pady=10)

        # Connect to the server and start the game loop
        threading.Thread(target=self.connect_to_server).start()

    def connect_to_server(self):
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        try:
            print(f"Attempting to connect to {self.hostname}:{self.port}")
            self.sock.connect((self.hostname, self.port))
            print("Connection established!")
            self.player_id = self.recv_int()
            self.update_player_info()
            threading.Thread(target=self.listen_for_messages).start()
```

```
        except socket.error as e:
            error_message = f"Unable to connect to server at
{self.hostname}:{self.port}\nError: {e}"
            print(error_message)
            messagebox.showerror("Connection Error", error_message)
            self.root.destroy()

    def update_player_info(self):
        # Update player info once connected
        self.player_label.config(text=f"Player: {'X' if self.player_id == 1
else 'O'}")
        self.status_label.config(text="Waiting for the game to start...")

    def listen_for_messages(self):
        while True:
            msg = self.recv_msg()
            if msg == "SRT":
                self.status_label.config(text="Game started! Your turn..." if
self.player_id == 1 else "Waiting for opponent...")
                self.clear_board()
                self.my_turn = (self.player_id == 1)
                self.update_buttons()
            elif msg == "TRN":
                self.my_turn = True
                self.status_label.config(text="Your turn!")
                self.update_buttons()
            elif msg == "WAT":
                self.my_turn = False
                self.status_label.config(text="Waiting for opponent's
move...")
                self.update_buttons()
            elif msg == "UPD":
                player_id = self.recv_int()
                move = self.recv_int()
                self.update_board(player_id, move)
            elif msg == "WIN":
                self.show_game_result("Congratulations! You win!")
            elif msg == "LSE":
                self.show_game_result("You lost. Better luck next time!")
            elif msg == "DRW":
                self.show_game_result("It's a draw!")
            elif msg == "INV":
                messagebox.showwarning("Invalid Move", "This position is
already taken. Try another one.")
            elif msg == "CNT":
                num_players = self.recv_int()
                messagebox.showinfo("Active Players", f"There are currently
{num_players} active players.")
```

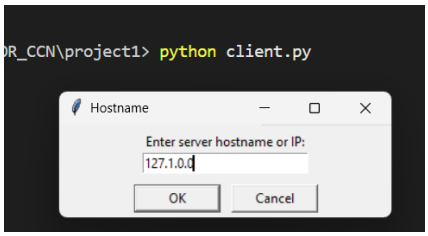
```
    def show_game_result(self, message):
        self.status_label.config(text=message)
        for row in self.buttons:
            for btn in row:
                btn.config(state="disabled")
        self.game_number += 1
        self.update_game_info()

    def send_move(self, row, col):
        if self.my_turn:
            move = row * 3 + col
            self.write_server_int(move)
            self.my_turn = False
            self.update_buttons()
        else:
            messagebox.showwarning("Not Your Turn", "Please wait for your
turn.")

    def update_board(self, player_id, move):
        symbol = 'X' if player_id == 1 else 'O'
        row, col = divmod(move, 3)
        self.board[row][col] = symbol
        self.buttons[row][col].config(text=symbol, state="disabled")

    def clear_board(self):
        self.board = [[' ' for _ in range(3)] for _ in range(3)]
        for i in range(3):
            for j in range(3):
                self.buttons[i][j].config(text="", state="normal")

    def update_buttons(self):
        for row in self.buttons:
            for btn in row:
                btn.config(state="normal" if self.my_turn else "disabled")

    def update_game_info(self):
        self.game_label.config(text=f"Game No: {self.game_number}")

    def recv_msg(self):
        try:
            msg = self.sock.recv(3).decode('utf-8')
            return msg
        except socket.error:
            self.sock.close()
            self.root.quit()
            return None
```

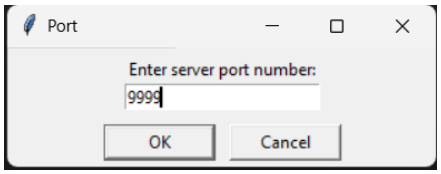
```
    def recv_int(self):
        try:
            data = self.sock.recv(4)
            return struct.unpack("!I", data)[0]
        except socket.error:
            self.sock.close()
            self.root.quit()
            return None

    def write_server_int(self, msg):
        try:
            data = struct.pack("!I", msg)
            self.sock.sendall(data)
        except socket.error:
            self.sock.close()
            self.root.quit()

def main():
    hostname = simpledialog.askstring("Hostname", "Enter server hostname or
IP:")
    port = simpledialog.askinteger("Port", "Enter server port number:")
    if hostname and port:
        client = TicTacToeClient(hostname, port)
        client.root.mainloop()

if __name__ == "__main__":
    main()
```

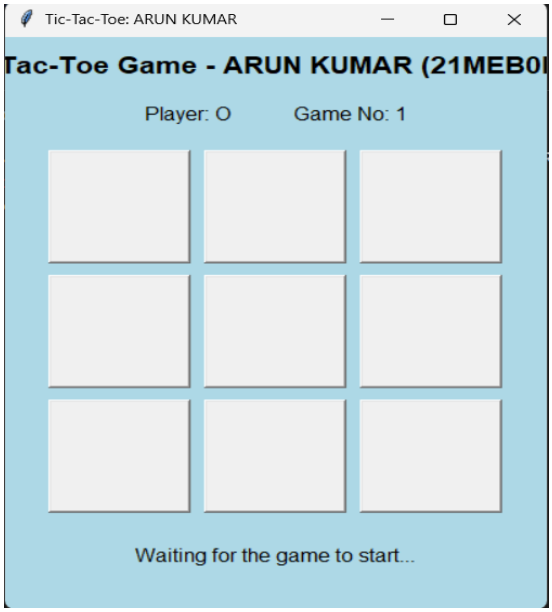
4. Screenshots of the working of client server program.

❖ **Running the server**

Running the client



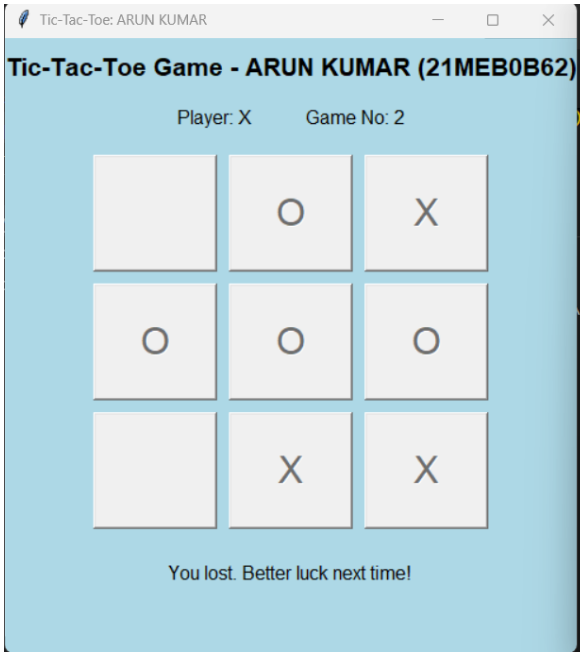


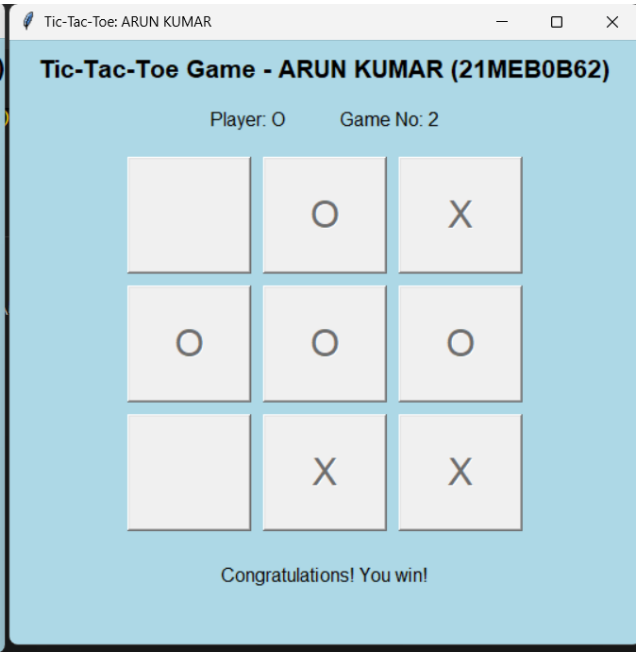


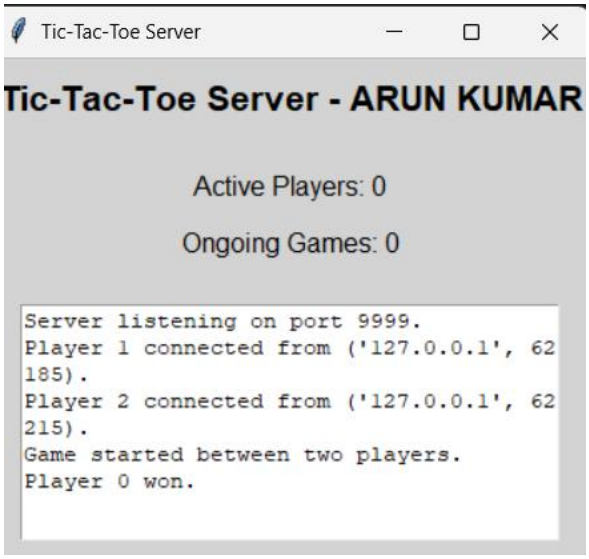
5. A test log including a range of tests and screenshot evidence of the results

- **GUI (made using tkinter) showing the connection establishment between client and server**

a. Output on the GUI of two players, when one player won and other lost.

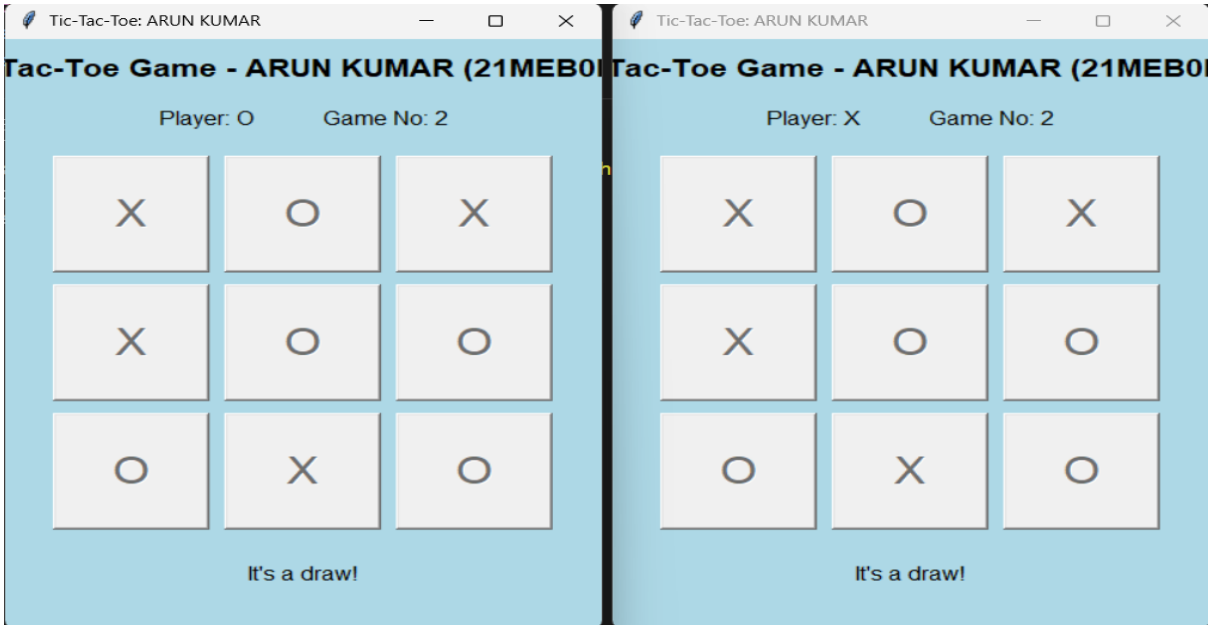


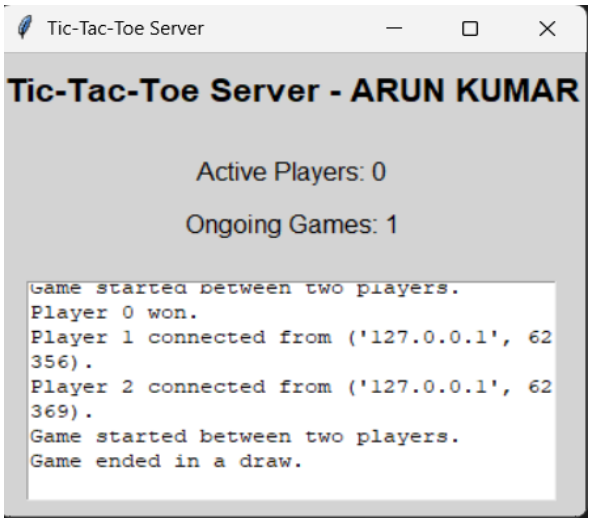




GUI on the server side when one of the player won. It also shows the ports from where they were connected.

b. When the game is DRAW between both the players





Output on the GUI of server side when the game is draw between players.

6. Learning Outcomes

Through the development of this Tic-Tac-Toe client-server application, I achieved the following:

- ❖ **Designing and implementing communication services for client/server** (One of the course outcome).

1. Understanding Sockets and Network Communication: I gained hands-on experience with Python sockets for establishing client-server connections and managing real-time data exchange.
2. Multi-threading for Concurrent Connections: Implementing multi-threading allowed me to handle multiple players and simultaneous games, an essential aspect of networked applications.
3. Error Handling and Robustness: I learned the importance of handling client disconnections and erroneous inputs, ensuring a stable and user-friendly experience.
4. GUI Design with tkinter: Creating a GUI in tkinter enabled me to visualize server activity, offering a valuable perspective on real-time application monitoring.

7. References

1. https://www.digitalocean.com/community/tutorials/python-socket-programming-server-client
2. *Python Socket Programming Documentation* - https://docs.python.org/3/library/socket.html
3. *Python Socket Programming Tutorial* - *Python Socket Programming Tutorial*
4. *Tkinter GUI Programming Reference* - *https://docs.python.org/3/library/tkinter.html*