22AIE204 INTRODUCTION TO COMPUTER NETWORK

Dual Player Battleship Game with Real-Time Chat

Submitted by

Group 8

AIE BATCH A

A. SRIRAM **[AM.SC.U4AIE23014]**

K. BRUHADESH **[AM.SC.U4AIE23039]**

P. VIVEK REDDY **[AM.SC.U4AIE23050]**

SRIVARMA BATTINI **[AM.SC.U4AIE23057]**

ABOUT THE PROJECT

Project Title: Dual Player Battleship Game with Real-Time Chat

Problem Statement:

Develop a Dual Player Battleship Game with Real-Time Chat to provide an engaging multiplayer experience. The game should allow two players to compete in a turn-based strategy game where they attempt to locate and sink each other's ships on a grid. Additionally, a real-time chat feature should be integrated to enhance player interaction, fostering a more social gaming experience. The solution should employ socket programming for robust client-server communication and multithreading for handling simultaneous game and chat activities seamlessly.

Description:

- The Dual Player Battleship Game with Real-Time Chat is an immersive and engaging multiplayer application developed using Python. This project combines elements of strategy, competition, and social interaction to deliver a unique gaming experience. Players participate in a classic battleship game, taking turns to guess the locations of their opponent's ships on a 5x5 grid. The game's objective is straightforward yet challenging: locate and sink all of the opponent's ships before they can sink yours.

- In addition to the strategic gameplay, a real-time chat feature has been integrated to encourage interaction between players. This allows for friendly banter, strategizing, or simply enjoying a conversation while competing.

Key Features:

- Multiplayer Game Setup:
  - Two players take turns guessing the opponent's ship positions on a 5x5 grid.
  - Ships are placed at predefined coordinates on the grid, and players fire shots by sending coordinates to the server.
  - The first player to sink all the opponent's ships wins the game.

- Real-Time Chat:
  - A built-in chat system allows players to send and receive messages while playing, adding a social

layer to the game.

- Socket Programming:
  - The game uses socket programming to establish a connection between the client and server.
  - The server handles two types of communication:
    - Game-related messages (firing shots, turn updates, hit/miss results).
    - Chat messages between the players.
  - Multithreading ensures the server can simultaneously manage both game interactions and chat messages without delays.

Game Mechanics:

- Turn-Based Play: Players alternate turns to fire at the opponent’s grid. The server checks if the shot is a hit or miss and updates both players.
- Server Logic:
  - Maintains the state of the game (player turns, ship positions, hit/miss results).
  - Ensures smooth transitions between player turns.
  - Sends game status updates (e.g., "Your turn," "Opponent's turn," "Game over") to the players.

- GUI Interface:
  - The Tkinter GUI displays the 5x5 game grid and a chat window.
  - Players interact with the grid to fire at the opponent's ships and communicate via the chat feature.
  - The client’s interface dynamically updates based on the server's responses, showing hit/miss results and turn changes.

Technical Aspects:

- Multithreading: Ensures the server handles both game actions and chat messages concurrently, providing a seamless user experience.
- Socket Communication:
  - The server listens for incoming connections on separate ports for the game and chat functionalities.
  - The client communicates with both the game server and the chat server to send moves and messages.

- Game State Management:
  - The server keeps track of ship positions, firing coordinates, and hit/miss results for each player.
  - The game progresses turn by turn, with the server validating each action and providing immediate feedback.

Future Enhancements:

- Expansion of grid size for more complex gameplay.
- Addition of more ship types and special abilities.
- AI opponent for solo play.
- Enhanced chat features (voice chat, emojis).

CODE

CLIENT CODE:

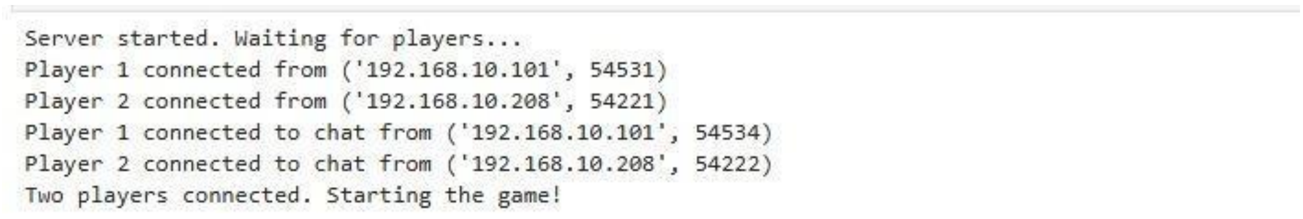
```
import socket
import threading
import tkinter as tk
from tkinter import messagebox, simpledialog

class BattleshipClient:
    def _init_(self, root):
        self.root = root
        self.root.title("Dual Player Battleship with Chat")
        self.root.configure(bg='#2C3E50')

        # Network setup
        self.client_game = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.client_chat = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        # Connect to the server
        self.server_ip = simpledialog.askstring("Server Connection", "Enter Server 
IP:", initialvalue="192.168.10.253")
        self.server_port = simpledialog.askinteger("Server Connection", "Enter Game 
Server Port:", initialvalue=8000)
        self.chat_port = self.server_port + 1  # Chat port assumed to be +1 of game 
port

        try:
            self.client_game.connect((self.server_ip, self.server_port))
            self.client_chat.connect((self.server_ip, self.chat_port))
        except Exception as e:
            messagebox.showerror("Connection Error", f"Could not connect to server: 
{e}")
            self.root.quit()
            return

        # Game variables
        self.grid_size = 5
        self.buttons = [[None for _ in range(self.grid_size)] for _ in 
range(self.grid_size)]
        self.is_turn = False
        self.hits = 0
        self.misses = 0

        # Setup styling
        self.setup_styles()

        # Layout setup
        self.setup_gui()

        # Start listening to server messages
        self.start_threads()

    def setup_styles(self):
        """Set up custom styles for the game."""
        self.root.option_add("*Font", "Arial 10")
        self.style = {
            'button_default': {
                'width': 4,
                'height': 2,
                'font': ('Arial', 10, 'bold'),
                'relief': tk.RAISED,
                'borderwidth': 3
            },
            'hit': {
                'bg': '#E74C3C',  # Bright red
                'fg': 'white'
            },
            'miss': {
                'bg': '#95A5A6',  # Gray
                'fg': 'white'
            },
            'opponent_hit': {
                'bg': '#F39C12',  # Orange
                'fg': 'black'
            },
            'opponent_miss': {
                'bg': '#BDC3C7',  # Light gray
                'fg': 'black'
            }
        }

    def setup_gui(self):
        """Sets up the game GUI."""
        # Title Frame
        self.title_frame = tk.Frame(self.root, bg='#2C3E50')
        self.title_frame.pack(pady=10)

        title_label = tk.Label(
            self.title_frame,
            text="Multiplayer Battleship with Chat",
            font=("Arial", 16, "bold"),
            fg='white',
            bg='#2C3E50'
        )
        title_label.pack()

        # Info Frame
        self.info_frame = tk.Frame(self.root, bg='#34495E')
        self.info_frame.pack(pady=10)

        self.turn_label = tk.Label(
            self.info_frame,
            text="Connecting to server...",
            font=("Arial", 14, "bold"),
            fg='#ECF0F1',
            bg='#34495E'
        )
        self.turn_label.pack(padx=20, pady=10)

        # Grid Frame
        self.grid_frame = tk.Frame(self.root, bg='#2C3E50')
        self.grid_frame.pack()

        for row in range(self.grid_size):
            for col in range(self.grid_size):
                btn = tk.Button(
                    self.grid_frame,
                    text=" ",
                    command=lambda r=row, c=col: self.fire(r, c),
                    **self.style['button_default'],
                    bg='#3498DB',
                    activebackground='#2980B9'
                )
                btn.grid(row=row, column=col, padx=2, pady=2)
                self.buttons[row][col] = btn

        # Chat Frame
        self.chat_frame = tk.Frame(self.root, bg='#34495E')
        self.chat_frame.pack(pady=10, fill=tk.BOTH, expand=True)

        self.chat_log = tk.Text(self.chat_frame, state=tk.DISABLED, height=10,
bg='#ECF0F1', fg='#2C3E50', wrap=tk.WORD)
        self.chat_log.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=10,
pady=10)

        self.chat_entry = tk.Entry(self.chat_frame, bg='#ECF0F1', fg='#2C3E50')
        self.chat_entry.pack(side=tk.LEFT, fill=tk.X, expand=True, padx=10, pady=10)
        self.chat_entry.bind("<Return>", lambda _: self.send_chat())

        send_button = tk.Button(self.chat_frame, text="Send",
command=self.send_chat, bg='#3498DB', fg='white')
        send_button.pack(side=tk.LEFT, padx=10, pady=10)

    def start_threads(self):
        """Start threads to listen for game and chat messages."""
        threading.Thread(target=self.listen_to_server, daemon=True).start()
        threading.Thread(target=self.listen_to_chat, daemon=True).start()

    def listen_to_server(self):
        """Listens for messages from the server."""
        while True:
            try:
                data = self.client_game.recv(1024).decode()

                if data.startswith("HIT"):
                    _, coords = data.split(":")
                    r, c = map(int, coords.split(","))
                    self.buttons[r][c].config(text="X", **self.style['hit'])
                    self.hits += 1

                elif data.startswith("MISS"):
                    _, coords = data.split(":")
                    r, c = map(int, coords.split(","))
                    self.buttons[r][c].config(text="O", **self.style['miss'])
                    self.misses += 1

                elif data.startswith("Opponent HIT"):
                    _, coords = data.split(":")
                    r, c = map(int, coords.split(","))
                    self.buttons[r][c].config(text="O",
**self.style['opponent_hit'])

                elif data.startswith("Opponent MISS"):
                    _, coords = data.split(":")
                    r, c = map(int, coords.split(","))
                    self.buttons[r][c].config(text="O",
**self.style['opponent_miss'])

                elif "Your turn" in data:
                    self.is_turn = True
                    self.turn_label.config(text="Your Turn!", fg="#2ECC71")

                elif "Not your turn" in data:
                    self.is_turn = False
                    self.turn_label.config(text="Opponent's Turn", fg="#E74C3C")

                elif "You won!" in data:
                    messagebox.showinfo("Game Over", "You won!")
                    self.root.quit()

                elif "You lost!" in data:
                    messagebox.showinfo("Game Over", "You lost!")
                    self.root.quit()

            except Exception as e:
                messagebox.showerror("Connection Error", f"Lost connection: {e}")
                break

    def listen_to_chat(self):
        """Listens for chat messages from the server."""
        while True:
            try:
                message = self.client_chat.recv(1024).decode()
                self.display_chat_message(message)
            except Exception as e:
                self.display_chat_message(f"Chat error: {e}")
                break

    def display_chat_message(self, message):
        """Displays a chat message in the chat log."""
        self.chat_log.config(state=tk.NORMAL)
        self.chat_log.insert(tk.END, f"{message}\n")
        self.chat_log.see(tk.END)
        self.chat_log.config(state=tk.DISABLED)

    def send_chat(self):
        """Sends a chat message to the server."""
        message = self.chat_entry.get()
        if message.strip():
            self.client_chat.send(message.encode())
            self.display_chat_message(f"You: {message}")
            self.chat_entry.delete(0, tk.END)

    def fire(self, row, col):
        """Handles the firing action."""
        if not self.is_turn:
            messagebox.showinfo("Wait", "It's not your turn!")
            return

        self.client_game.send(f"{row},{col}".encode())
        self.is_turn = False
        self.turn_label.config(text="Waiting for opponent...", fg="#E67E22")
        self.buttons[row][col].config(state=tk.DISABLED)

if __name___== "__main__":
    root = tk.Tk()
    BattleshipClient(root)
    root.mainloop()
```

SERVER CODE:

```
import socket
import threading

class BattleshipServer:
    def init(self, host="0.0.0.0", port=8000, chat_port=8001):
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server.bind((host, port))
        self.server.listen(2)

        self.chat_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.chat_server.bind((host, chat_port))
        self.chat_server.listen(2)

        print("Server started. Waiting for players...")
        self.clients = []  # Game sockets
        self.chat_clients = []  # Chat sockets
        self.turn = 0  # Player 0 starts
        self.grid_size = 5
         self.ships = [{(1, 1), (2, 2), (3, 3)},  # Player 1's ships
                      {(0, 0), (1, 2), (4, 4)}]  # Player 2's ships

    def broadcast(self, message, exclude_client=None):
        """Send a message to all clients except the excluded one."""
        for client in self.clients:
            if client != exclude_client:
                client.send(message.encode())

    def handle_game(self, client, client_id):
        """Handle player actions."""
        while True:
            try:
                data = client.recv(1024).decode()

                if not data:
                    break

                if self.turn != client_id:
                    client.send("Not your turn".encode())
                    continue

                if data.startswith("CHAT:"):
                    self.handle_chat(data[5:], client_id)
                    continue

                row, col = map(int, data.split(","))
                opponent_id = 1 - client_id
                opponent_ships = self.ships[opponent_id]

                if (row, col) in opponent_ships:
                    opponent_ships.remove((row, col))
                    client.send(f"HIT:{row},{col}".encode())
                    self.clients[opponent_id].send(f"Opponent
HIT:{row},{col}".encode())

                    if not opponent_ships:
                        client.send("You won!".encode())
                        self.clients[opponent_id].send("You lost!".encode())
                        break
                else:
                    client.send(f"MISS:{row},{col}".encode())
                    self.clients[opponent_id].send(f"Opponent
MISS:{row},{col}".encode())

                # Switch turn
                self.turn = opponent_id
                self.clients[self.turn].send("Your turn".encode())
                self.clients[1 - self.turn].send("Not your turn".encode())

            except Exception as e:
                print(f"Error handling client {client_id}: {e}")
                break

        client.close()

    def handle_chat(self, message, client_id):
        """Handle chat messages and forward them to the opponent."""
        opponent_id = 1 - client_id
        try:
            chat_message = f"CHAT:Player {client_id + 1}: {message}"
            self.chat_clients[opponent_id].send(chat_message.encode())
        except Exception as e:
            print(f"Error sending chat message from Player {client_id + 1}: {e}")

    def handle_chat_client(self, chat_client, client_id):
        """Handle chat communication for a specific client."""
        while True:
            try:
                data = chat_client.recv(1024).decode()
                if data:
                    self.handle_chat(data, client_id)
            except Exception as e:
                print(f"Chat error with Player {client_id + 1}: {e}")
                break

        chat_client.close()

    def start(self):
        """Start the game server."""
        while len(self.clients) < 2:
            client, addr = self.server.accept()
            self.clients.append(client)
            print(f"Player {len(self.clients)} connected from {addr}")

        while len(self.chat_clients) < 2:
            chat_client, addr = self.chat_server.accept()
            self.chat_clients.append(chat_client)
            print(f"Player {len(self.chat_clients)} connected to chat from {addr}")

        if len(self.clients) == 2 and len(self.chat_clients) == 2:
            print("Two players connected. Starting the game!")
            self.clients[0].send("Your turn".encode())
            self.clients[1].send("Not your turn".encode())

            threading.Thread(target=self.handle_game, args=(self.clients[0],
0)).start()
            threading.Thread(target=self.handle_game, args=(self.clients[1],
1)).start()

            threading.Thread(target=self.handle_chat_client,
args=(self.chat_clients[0], 0)).start()
            threading.Thread(target=self.handle_chat_client,
args=(self.chat_clients[1], 1)).start()

if __name___== "__main__":
    server = BattleshipServer()
    server.start()
```

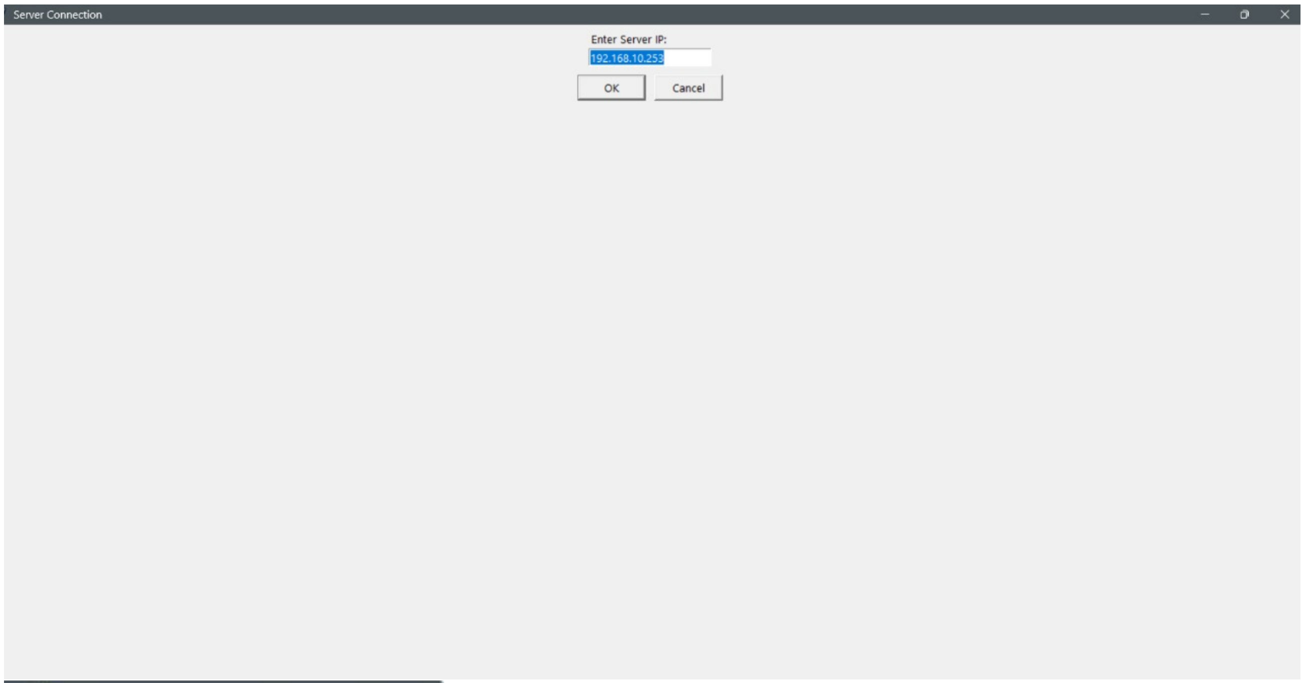
OUTPUT

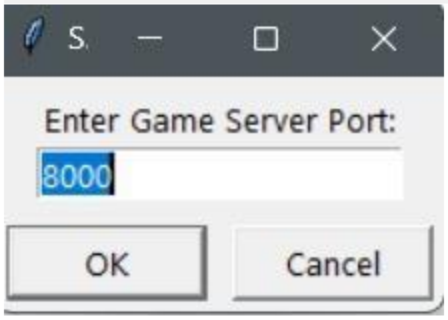
Server:

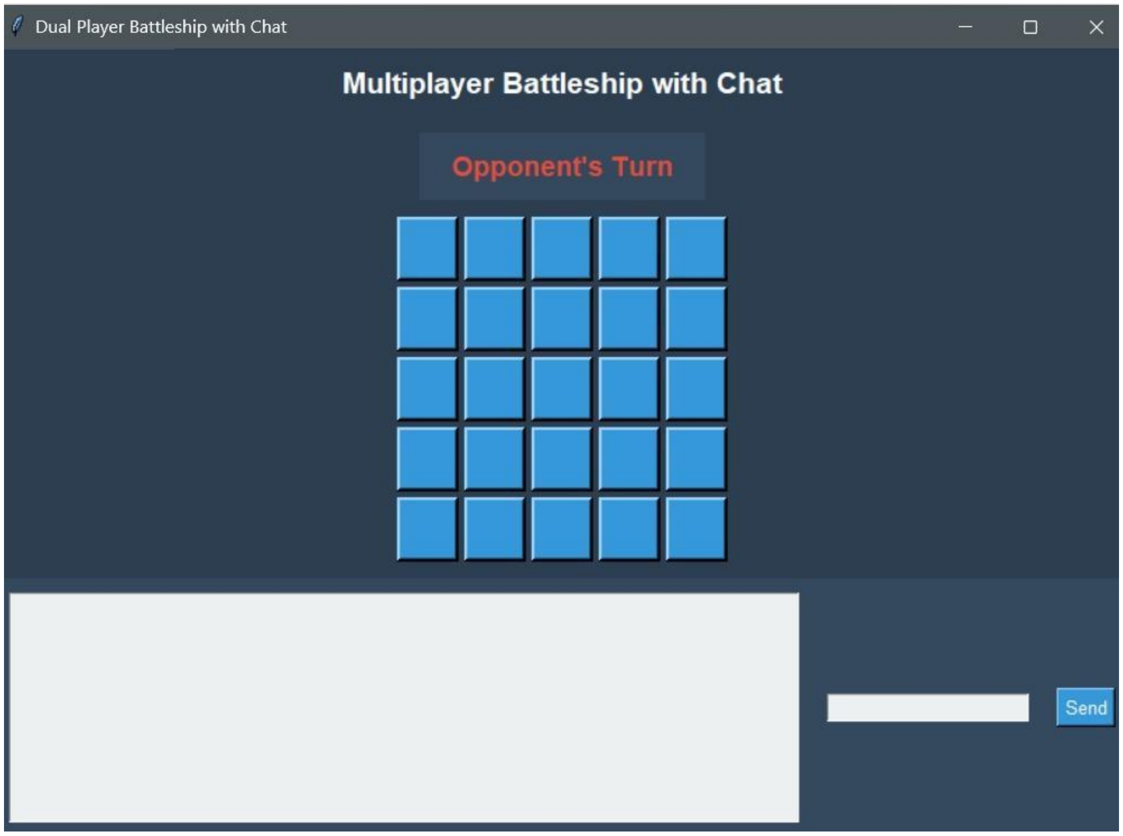


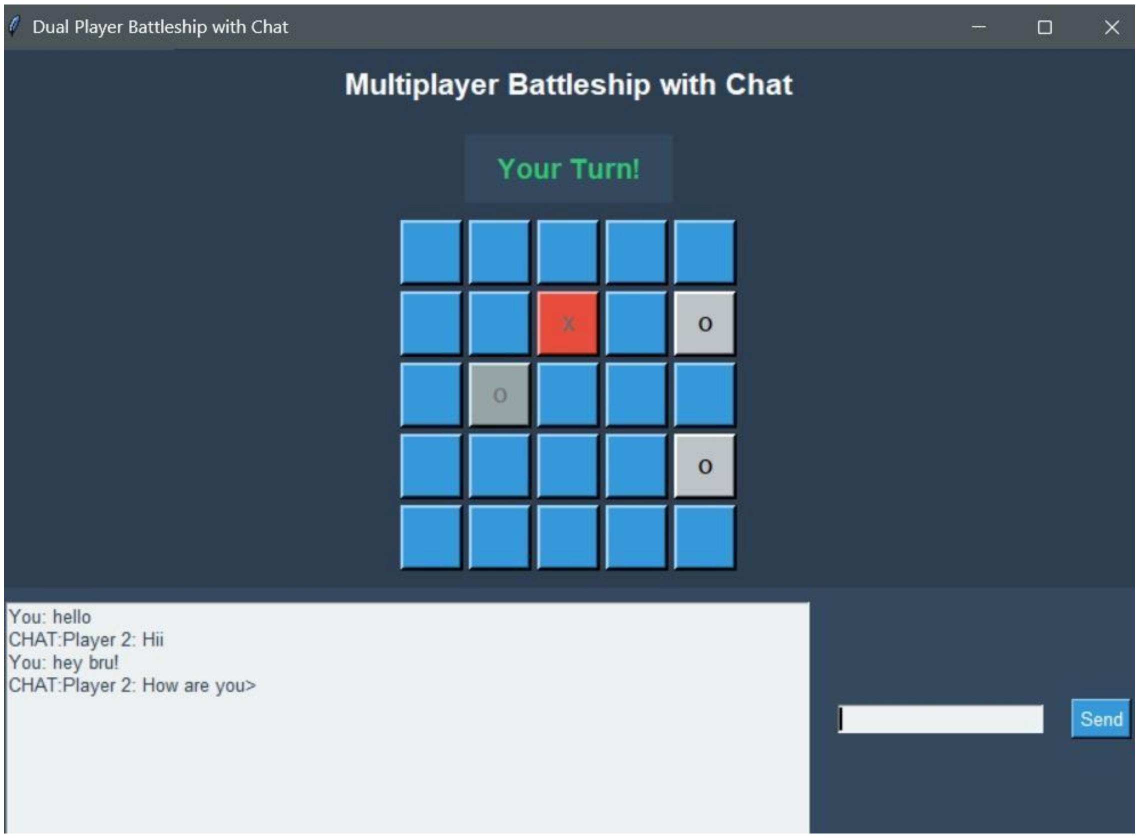
```
Server started. Waiting for players...
Player 1 connected from ('192.168.10.101', 54531)
Player 2 connected from ('192.168.10.208', 54221)
Player 1 connected to chat from ('192.168.10.101', 54534)
Player 2 connected to chat from ('192.168.10.208', 54222)
Two players connected. Starting the game!
```

Client:

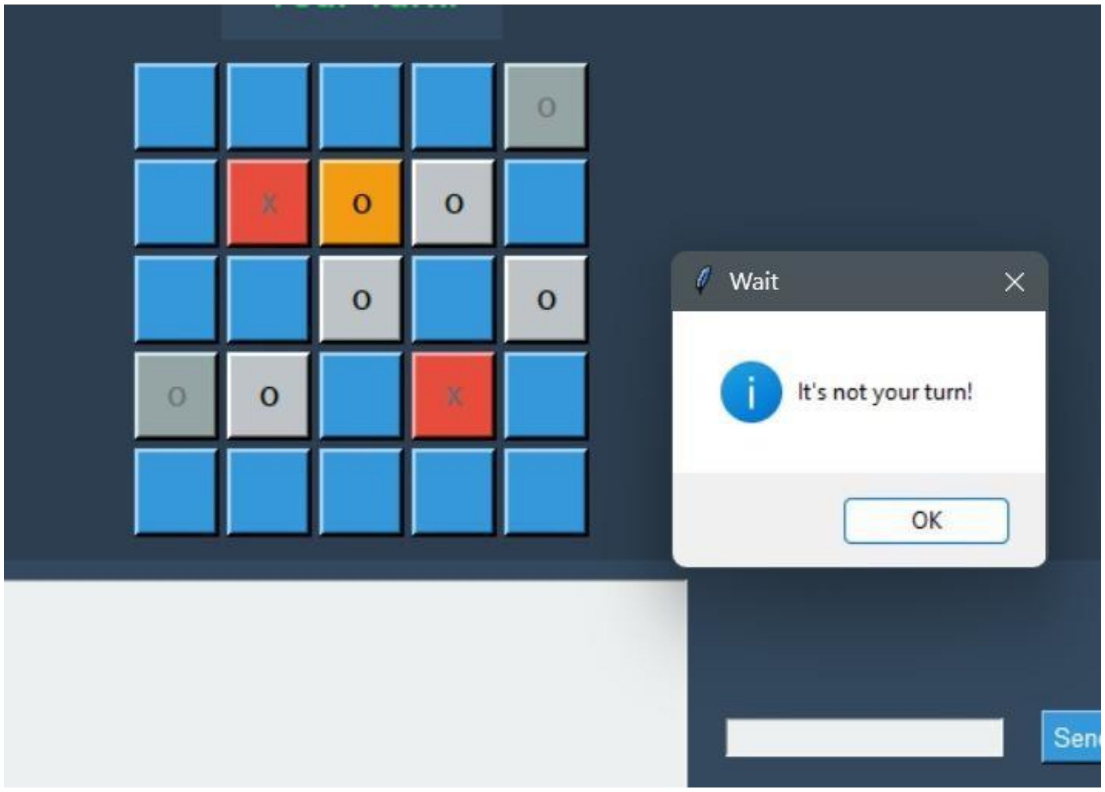


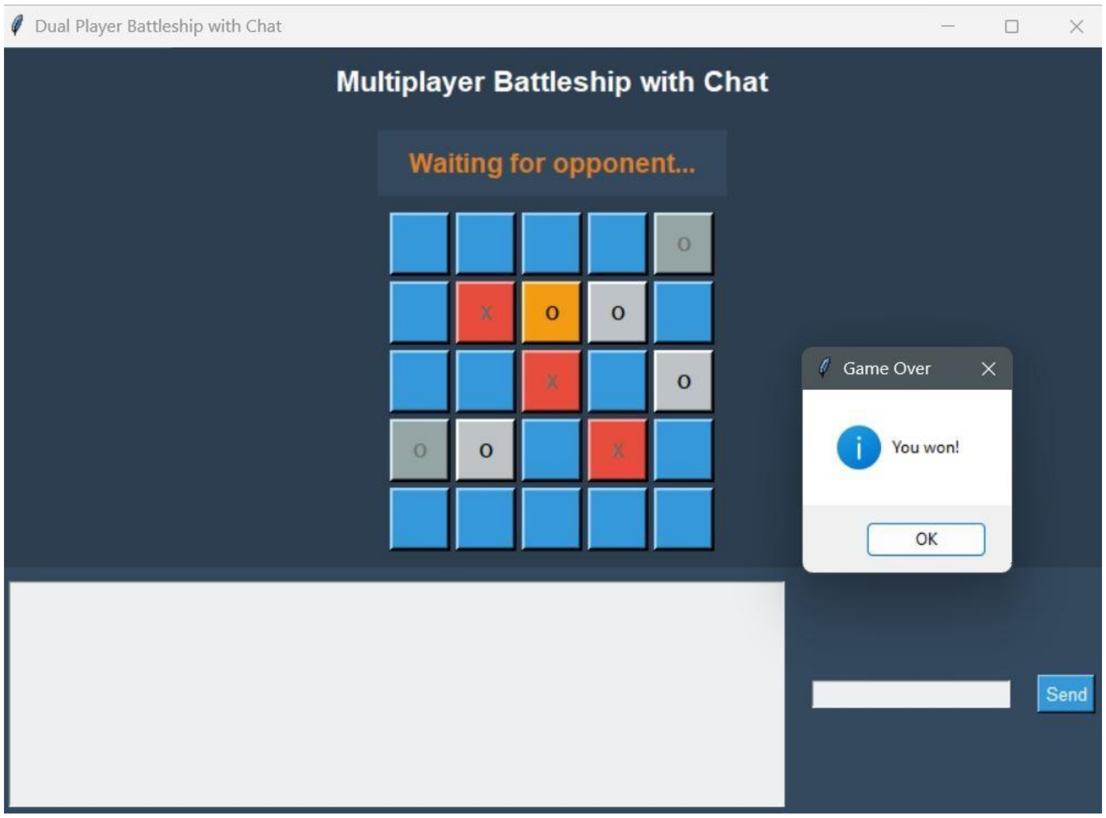



Dual Player Battleship with Chat
Multiplayer Battleship with Chat
Opponent's Turn
Send


Dual Player Battleship with Chat
Multiplayer Battleship with Chat
Your Turn!
X
O
O
O
Send

```
You: hello
CHAT:Player 2: Hii
You: hey bru!
CHAT:Player 2: How are you>
```


O
X
O
O
O
O
O
O
X
Wait
It's not your turn!
OK
Sen


Dual Player Battleship with Chat
Multiplayer Battleship with Chat
Waiting for opponent...
O
X
O
O
X
O
O
O
X
Game Over
You won!
OK
Send

CONCLUSION

The Dual Player Battleship game project successfully demonstrates the integration of socket programming with interactive user interfaces to create a multiplayer gaming experience. By leveraging computer networking concepts, such as TCP sockets and multithreading, the project enables real-time communication between two players for both game actions and chat functionality.

This project highlights the importance of efficient server-client architecture in handling multiple simultaneous connections, ensuring synchronization between players during gameplay. The implementation also showcases how a robust protocol design can manage game states, player turns, and error handling, resulting in a seamless and engaging experience.

Additionally, the game's graphical interface provides an intuitive and user-friendly interaction for players, further enhancing the usability and enjoyment of the application. The inclusion of a chat feature adds a personal touch, enabling players to communicate and strategize during the game.

Overall, the project serves as a practical application of theoretical computer networking concepts, demonstrating their real-world usage in multiplayer gaming. It not only meets the objective of creating a functional game but also reinforces the understanding of network programming, multithreading, and client-server communication. This project can be expanded in the future with features like larger grids, multiple ship types, or additional players, making it a versatile and scalable platform for further development.