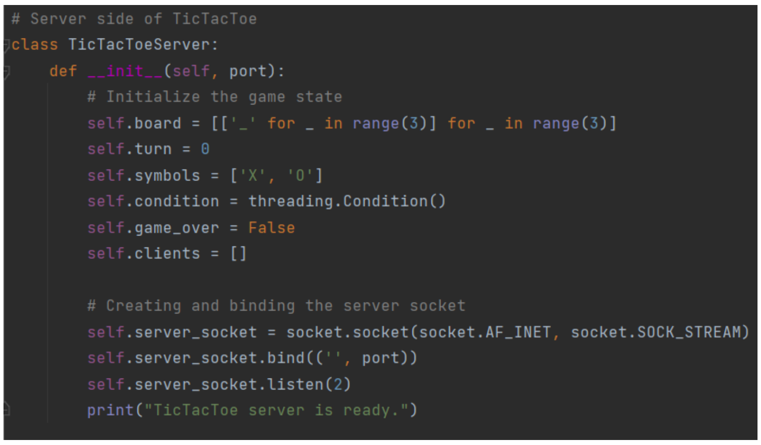
Atahan İyiekici
21801985
CS421-001
PA2

CS421 PA-2 TicTacToe Game over TCP

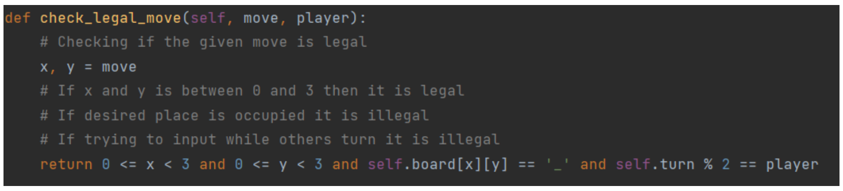
Codes and explanation:



```python
# Server side of TicTacToe
class TicTacToeServer:
    def __init__(self, port):
        # Initialize the game state
        self.board = [['_' for _ in range(3)] for _ in range(3)]
        self.turn = 0
        self.symbols = ['X', 'O']
        self.condition = threading.Condition()
        self.game_over = False
        self.clients = []

        # Creating and binding the server socket
        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server_socket.bind(('', port))
        self.server_socket.listen(2)
        print("TicTacToe server is ready.")
```

Figure 1: Constructor of TicTacToe server class

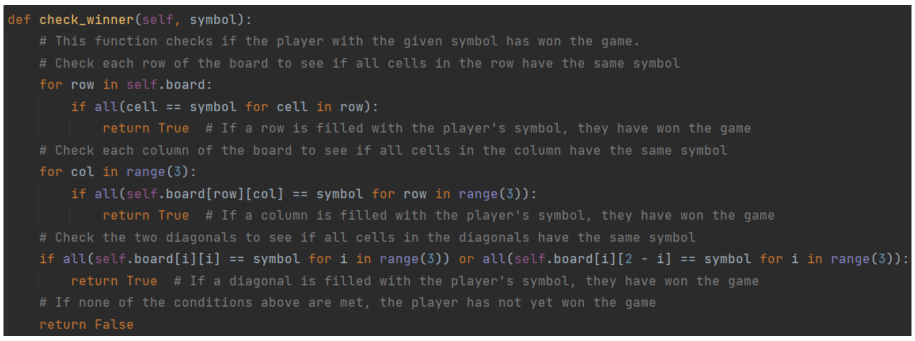
In the TicTacToe server code, first, there are some parameters that is setted in the __init__ function. These are board, turn, symbols, condition(for multi-threading), game_over(boolean for whether the game is going or over), clients and socket. Also, the server socket binding is inside the function.



```python
def check_legal_move(self, move, player):
    # Checking if the given move is legal
    x, y = move
    # If x and y is between 0 and 3 then it is legal
    # If desired place is occupied it is illegal
    # If trying to input while others turn it is illegal
    return 0 <= x < 3 and 0 <= y < 3 and self.board[x][y] == '_' and self.turn % 2 == player
```

Figure 2: check_legal_move function

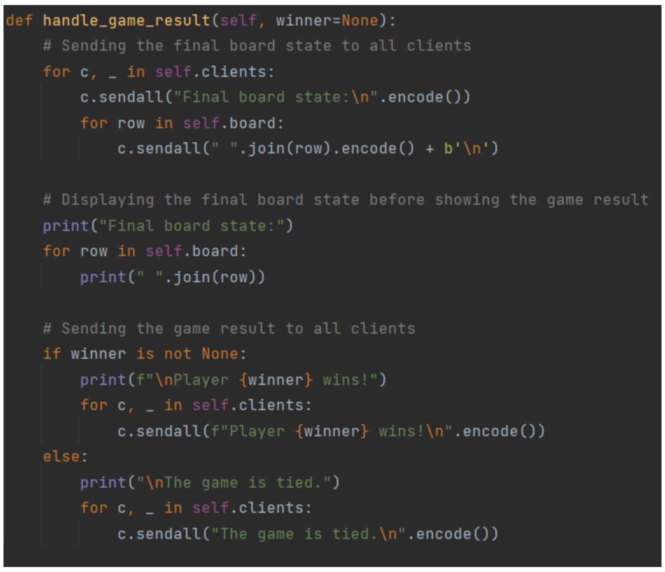
Check_legal_move function simply checks whether the move of a client is legal or illegal.



```python
def check_winner(self, symbol):
    # This function checks if the player with the given symbol has won the game.
    # Check each row of the board to see if all cells in the row have the same symbol
    for row in self.board:
        if all(cell == symbol for cell in row):
            return True  # If a row is filled with the player's symbol, they have won the game
    # Check each column of the board to see if all cells in the column have the same symbol
    for col in range(3):
        if all(self.board[row][col] == symbol for row in range(3)):
            return True  # If a column is filled with the player's symbol, they have won the game
    # Check the two diagonals to see if all cells in the diagonals have the same symbol
    if all(self.board[i][i] == symbol for i in range(3)) or all(self.board[i][2 - i] == symbol for i in range(3)):
        return True  # If a diagonal is filled with the player's symbol, they have won the game
    # If none of the conditions above are met, the player has not yet won the game
    return False
```

Figure 3: check_winner function

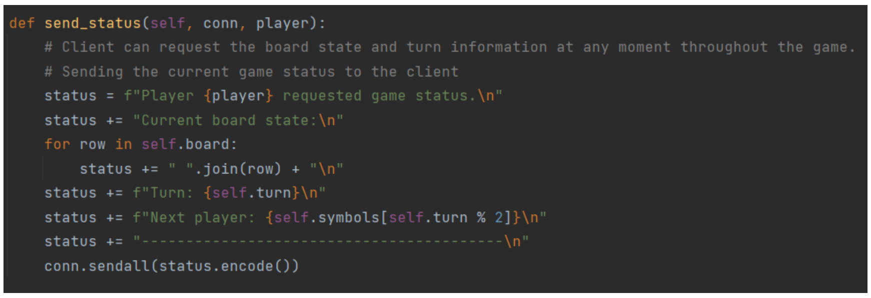
Check_winner function checks the rows and columns to see whether there is a winner or not in terms of TicTacToe conditions of rows and columns.



```python
def handle_game_result(self, winner=None):
    # Sending the final board state to all clients
    for c, _ in self.clients:
        c.sendall("Final board state:\n".encode())
        for row in self.board:
            c.sendall(" ".join(row).encode() + b'\n')

    # Displaying the final board state before showing the game result
    print("Final board state:")
    for row in self.board:
        print(" ".join(row))

    # Sending the game result to all clients
    if winner is not None:
        print(f"\nPlayer {winner} wins!")
        for c, _ in self.clients:
            c.sendall(f"Player {winner} wins!\n".encode())
    else:
        print("\nThe game is tied.")
        for c, _ in self.clients:
            c.sendall("The game is tied.\n".encode())
```

Figure 4: handle_game_result function

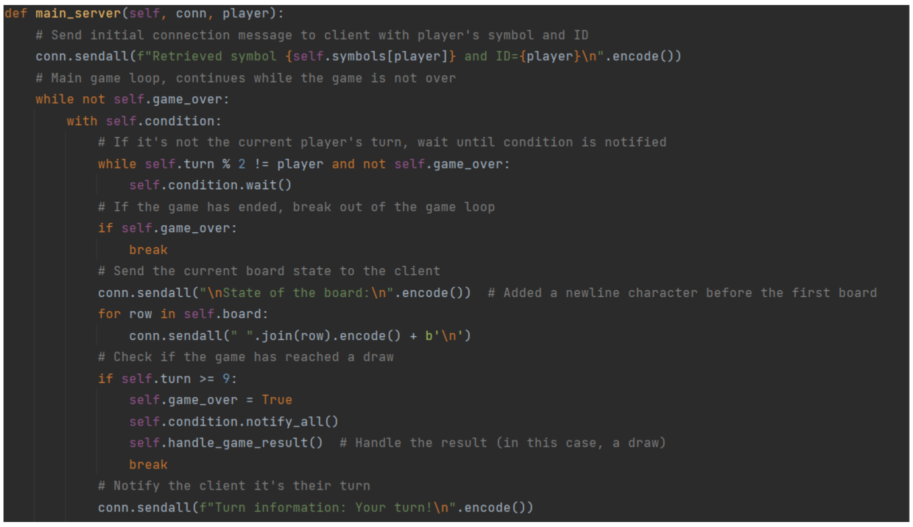
Handle_game_result function takes one parameter as winner which is defined by the main function in the server code. With taking the information about winner this function displays the final state of the board and winner after the game end. If the game is tied the function also displays the result for both clients and the server.



```python
def send_status(self, conn, player):
    # Client can request the board state and turn information at any moment throughout the game.
    # Sending the current game status to the client
    status = f"Player {player} requested game status.\n"
    status += "Current board state:\n"
    for row in self.board:
        status += " ".join(row) + "\n"
    status += f"Turn: {self.turn}\n"
    status += f"Next player: {self.symbols[self.turn % 2]}\n"
    status += "------------------------------------------\n"
    conn.sendall(status.encode())
```

Figure 5: send_status function

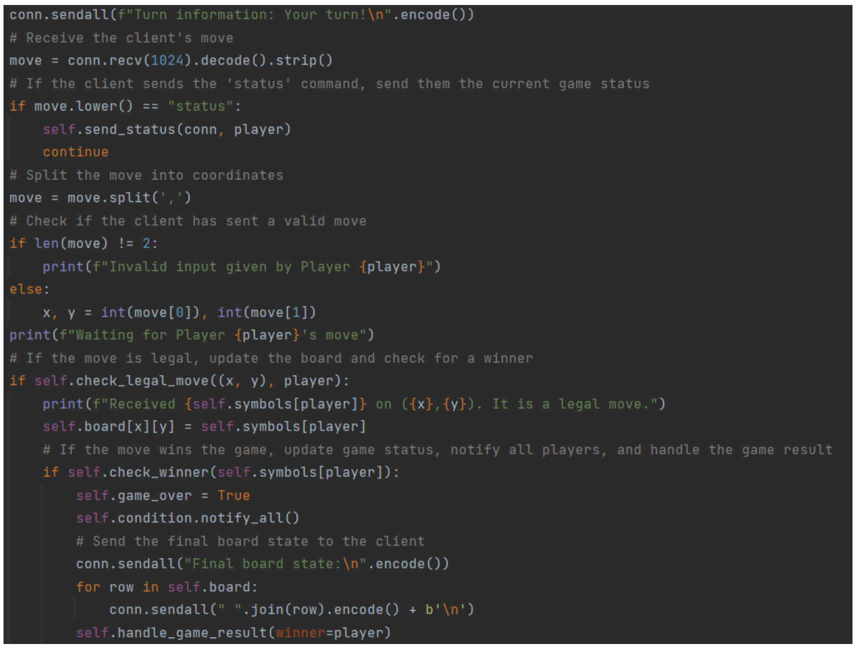
When client inputs “status”, the send_status function activates and displays the current state of board to the client.



```python
def main_server(self, conn, player):
    # Send initial connection message to client with player's symbol and ID
    conn.sendall(f"Retrieved symbol {self.symbols[player]} and ID={player}\n".encode())
    # Main game loop, continues while the game is not over
    while not self.game_over:
        with self.condition:
            # If it's not the current player's turn, wait until condition is notified
            while self.turn % 2 != player and not self.game_over:
                self.condition.wait()
            # If the game has ended, break out of the game loop
            if self.game_over:
                break
            # Send the current board state to the client
            conn.sendall("\nState of the board:\n".encode())  # Added a newline character before the first board
            for row in self.board:
                conn.sendall(" ".join(row).encode() + b'\n')
            # Check if the game has reached a draw
            if self.turn >= 9:
                self.game_over = True
                self.condition.notify_all()
                self.handle_game_result()  # Handle the result (in this case, a draw)
                break
            # Notify the client it's their turn
            conn.sendall(f"Turn information: Your turn!\n".encode())
```

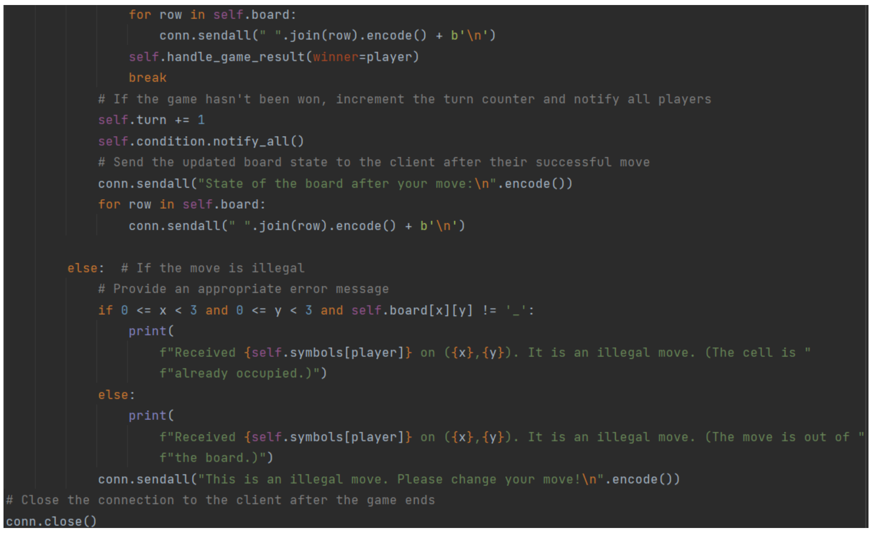
Figure 6: main_server function part 1

Atahan İyiekici
21801985
CS421-001
PA2



```python
conn.sendall(f"Turn information: Your turn!\n".encode())
# Receive the client's move
move = conn.recv(1024).decode().strip()
# If the client sends the 'status' command, send them the current game status
if move.lower() == "status":
    self.send_status(conn, player)
    continue
# Split the move into coordinates
move = move.split(',')
# Check if the client has sent a valid move
if len(move) != 2:
    print(f"Invalid input given by Player {player}")
else:
    x, y = int(move[0]), int(move[1])
print(f"Waiting for Player {player}'s move")
# If the move is legal, update the board and check for a winner
if self.check_legal_move((x, y), player):
    print(f"Received {self.symbols[player]} on ({x},{y}). It is a legal move.")
    self.board[x][y] = self.symbols[player]
    # If the move wins the game, update game status, notify all players, and handle the game result
    if self.check_winner(self.symbols[player]):
        self.game_over = True
        self.condition.notify_all()
        # Send the final board state to the client
        conn.sendall("Final board state:\n".encode())
        for row in self.board:
            conn.sendall(" ".join(row).encode() + b'\n')
        self.handle_game_result(winner=player)
```

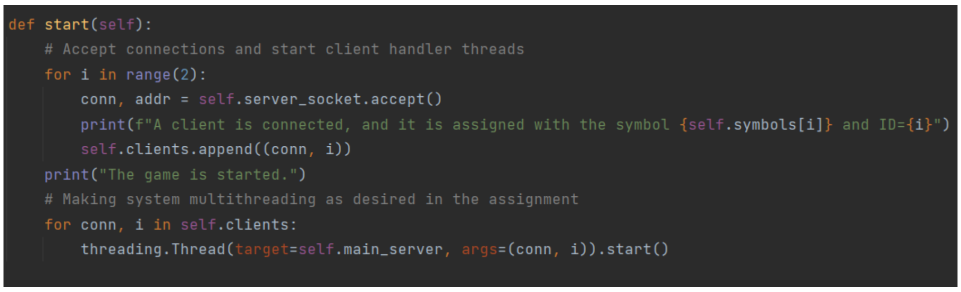
Figure 7: main_server function part 2



```python
                for row in self.board:
                    conn.sendall(" ".join(row).encode() + b'\n')
                self.handle_game_result(winner=player)
                break
            # If the game hasn't been won, increment the turn counter and notify all players
            self.turn += 1
            self.condition.notify_all()
            # Send the updated board state to the client after their successful move
            conn.sendall("State of the board after your move:\n".encode())
            for row in self.board:
                conn.sendall(" ".join(row).encode() + b'\n')

        else:  # If the move is illegal
            # Provide an appropriate error message
            if 0 <= x < 3 and 0 <= y < 3 and self.board[x][y] != '_':
                print(
                    f"Received {self.symbols[player]} on ({x},{y}). It is an illegal move. (The cell is "
                    f"already occupied.)")
            else:
                print(
                    f"Received {self.symbols[player]} on ({x},{y}). It is an illegal move. (The move is out of "
                    f"the board.)")
            conn.sendall("This is an illegal move. Please change your move!\n".encode())
# Close the connection to the client after the game ends
conn.close()
```

Figure 8: main_server function part 3

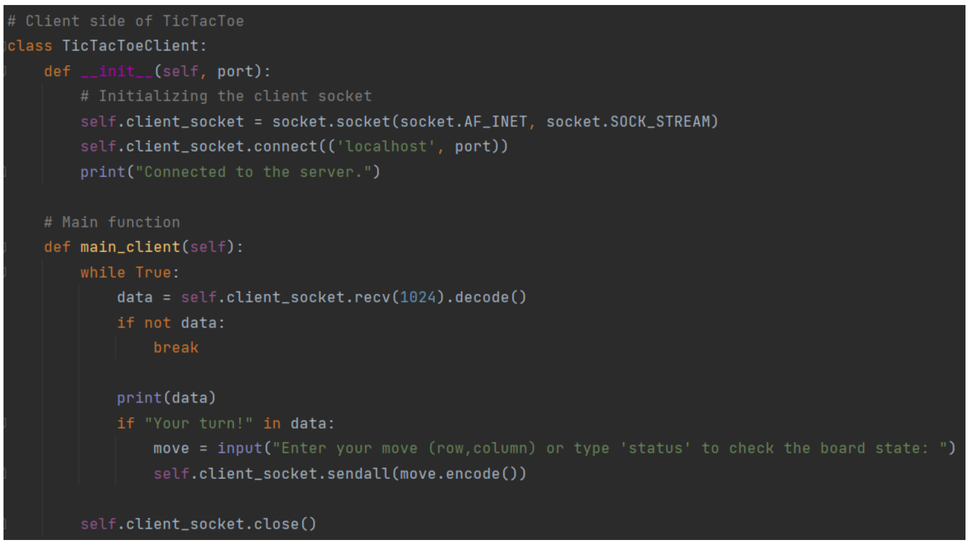
In the Figures 6,7 and 8 the main function of the server part is illustrated. The server_main function starts when the game is started (after there are 2 clients). Firstly, it sends information to clients for their representations in the game as X or O. The game continues until there is a winner or no more move left. The loop also considers the condition of threads, if it is first clients turn the second client won't interrupt, likewise, if it is second clients turn first client won't play, they waits until the other client's turn is over. Then, the state of the board is displayed. Before moving on, the loop checks whether game is tie. If game is tie the final state of board is displayed and game over boolean is setted True which breaks the loop. Then, printing the "your turn!" string to the client that will play that round. After that, the information of coordinates is received from the client. If clients enters "status" then send_status function sends the information. As we are getting the input like a tupple with 2 integers between a comma, if the length is not equal to 2 the systems raises an error of invalid input. After getting the x,y coordinates for the row and column information, firstly, we check whether the move is legal or not with check_legal_move function, then the result is displayed in the server terminal. Then, the system checks whether there is a winner with check_winner function. If there is a winner than the results are displayed. If there is not a winner, the turn goes to other client and takes input from that client. Also, if the move is illegal the system warns client that there is an illegal move and wants another input. After the game is over connection is closed.



```python
def start(self):
    # Accept connections and start client handler threads
    for i in range(2):
        conn, addr = self.server_socket.accept()
        print(f"A client is connected, and it is assigned with the symbol {self.symbols[i]} and ID={i}")
        self.clients.append((conn, i))
    print("The game is started.")
    # Making system multithreading as desired in the assignment
    for conn, i in self.clients:
        threading.Thread(target=self.main_server, args=(conn, i)).start()
```

Figure 9: start function

In Figure 9, start is the function for accepting connections of the clients with using threads.

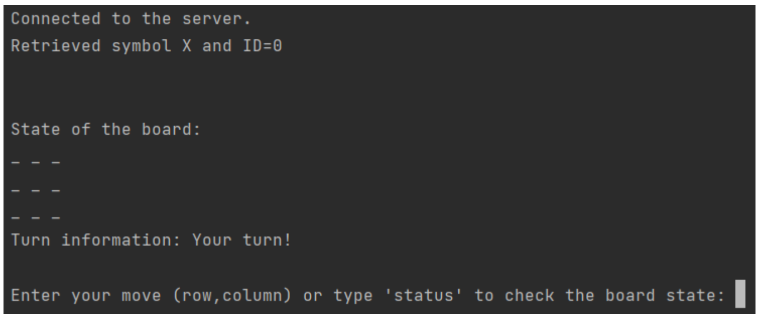


```python
# Client side of TicTacToe
class TicTacToeClient:
    def __init__(self, port):
        # Initializing the client socket
        self.client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.client_socket.connect(('localhost', port))
        print("Connected to the server.")

    # Main function
    def main_client(self):
        while True:
            data = self.client_socket.recv(1024).decode()
            if not data:
                break

            print(data)
            if "Your turn!" in data:
                move = input("Enter your move (row,column) or type 'status' to check the board state: ")
                self.client_socket.sendall(move.encode())

        self.client_socket.close()
```

Figure 10: Client side of the TicTacToe code

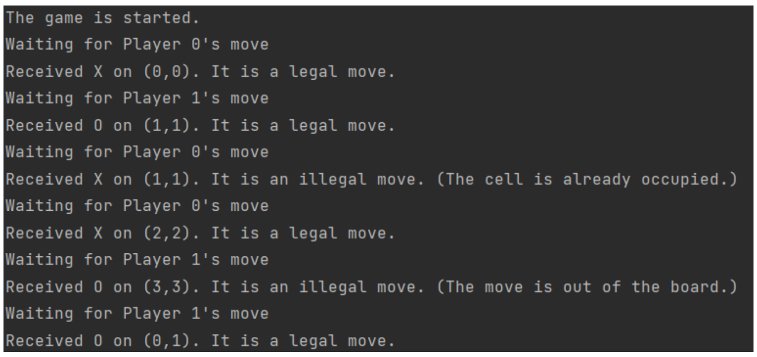
In the client side of the code, Figure 10, the TicTacToe class firstly initizalizes the client socket with binding local host. There is also main_client method inside the class for receiving the data from server and giving input from the client to send it to server to send the move of the clients. After the process client socket will be closed.

Results:



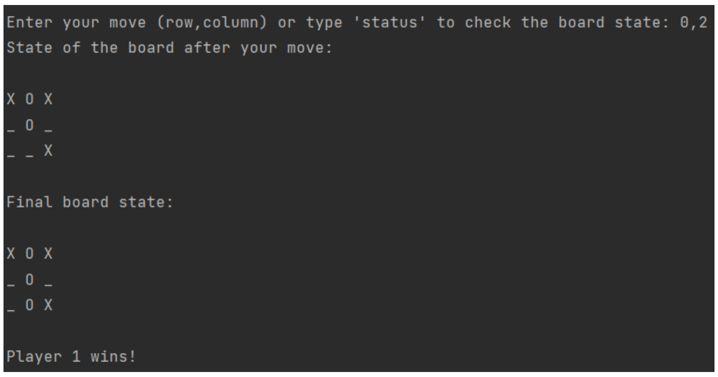
```
Connected to the server.
Retrieved symbol X and ID=0


State of the board:
- - -
- - -
- - -
Turn information: Your turn!

Enter your move (row,column) or type 'status' to check the board state: 
```

Figure 11: Opening screen of first client terminal



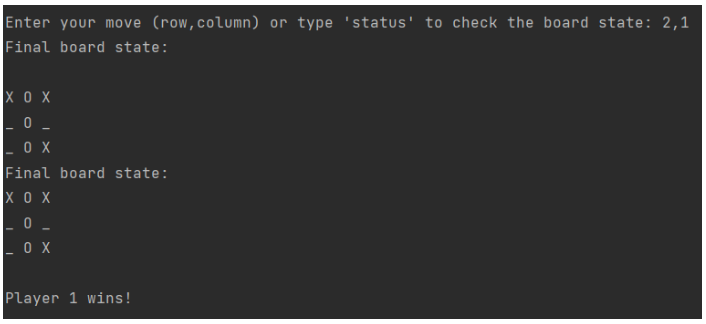
```
The game is started.
Waiting for Player 0's move
Received X on (0,0). It is a legal move.
Waiting for Player 1's move
Received O on (1,1). It is a legal move.
Waiting for Player 0's move
Received X on (1,1). It is an illegal move. (The cell is already occupied.)
Waiting for Player 0's move
Received X on (2,2). It is a legal move.
Waiting for Player 1's move
Received O on (3,3). It is an illegal move. (The move is out of the board.)
Waiting for Player 1's move
Received O on (0,1). It is a legal move.
```

Figure 12: Example of server terminal look



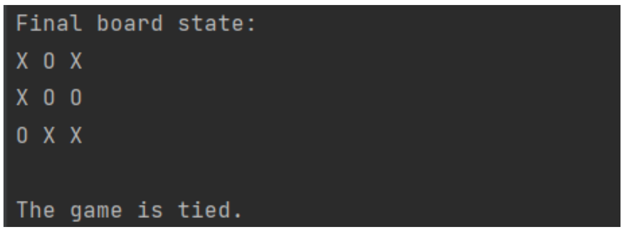
```
Enter your move (row,column) or type 'status' to check the board state: 0,2
State of the board after your move:

X O X
_ O _
_ _ X

Final board state:

X O X
_ O _
_ O X

Player 1 wins!
```

Figure 13: After the losing clients terminal in the end of the game



```
Enter your move (row,column) or type 'status' to check the board state: 2,1
Final board state:

X O X
_ O _
_ O X
Final board state:
X O X
_ O _
_ O X

Player 1 wins!
```

Figure 14: After the winner clients terminal in the end of the game



```
Final board state:
X O X
X O O
O X X

The game is tied.
```

Figure 15: An example of a terminal when game is tied in the end of the game

Note: Since there are limited pages to show, other conditions can be examined by using the code. The code can be executed using three terminal with python3 TicTacToeServer.py [port] and two python3 TicTacToeClient.py [port] messages in the terminals.