**Overview of Socket Programming in the Connect 4 Game**

The game uses **TCP sockets** for communication between a server (server.py) and multiple clients (client.py). The server manages game rooms, user connections, and the Connect 4 game logic, while the client provides a GUI for users to interact with the server (e.g., joining rooms, sending messages, playing the game). Communication is facilitated through **serialized messages** (using pickle) sent over sockets, with each message containing a "Command" key to specify the action and additional data like usernames, room names, or game moves.

The flowchart in connect4\_flowchart.txt outlines the user journey, from connecting to the server to playing the game and leaving. Below, each step is explained in order, detailing the commands sent/received, their purposes, and the key variables involved.

**User Flow with Commands and Variables (Following the Flowchart)**

**1. Create Username**

**Purpose**: The user connects to the server by entering a username, establishing a socket connection.

* **File**: client.py
* **Function**: ClientMenu.Create\_socket
* **User Action**: The user enters a username in the username\_input QLineEdit and clicks the "Connect" button.
* **Command Sent**:
  + **Command**: "Check\_Username"
  + **Data**:
    - User\_Name: The username entered by the user (string).
  + **Purpose**: Requests the server to validate the username and register the client.
* **Command Received** (from server.py: ChatServer.handle\_client):
  + **Command**: "Check\_Username"
  + **Data**:
    - Status: "Valid" (indicating the username is accepted).
    - Users\_In\_Room: [] (empty list, as the user hasn’t joined a room yet).
  + **Purpose**: Confirms the username is valid and provides initial room data.
* **Key Variables**:
  + **Client**:
    - self.username (string): Stores the user’s entered username.
    - self.client\_socket (socket object): The TCP socket used for communication with the server.
    - self.running (boolean): Tracks whether the client is actively connected (True after connection).
    - self.is\_disconnected (boolean): Tracks disconnection status (False after connection).
  + **Server**:
    - self.clients (dict): Maps usernames to their socket objects ({username: client\_socket}).
* **Process**:
  + **Client**: Creates a socket, connects to the server at 127.0.0.1:12345, and sends the "Check\_Username" message. Upon receiving the "Valid" response, it enables UI elements (room\_selector, create\_room\_button, join\_room\_button) and starts a thread (receive\_messages) to listen for server messages.
  + **Server**: Adds the username and socket to self.clients, sends the "Check\_Username" response, and broadcasts the room state to all clients.
* **Socket Role**: Establishes a persistent TCP connection for bidirectional communication.

**2. Create Room**

**Purpose**: The user creates a new game room for others to join.

* **File**: client.py
* **Function**: ClientMenu.Create\_room
* **User Action**: The user enters a room name in room\_input QLineEdit and clicks the "Create Room" button.
* **Command Sent**:
  + **Command**: "Create\_Room"
  + **Data**:
    - Room\_Name: The name of the new room (string).
    - User\_Name: The user’s username.
  + **Purpose**: Requests the server to create a new room.
* **Command Received** (from server.py: ChatServer.handle\_client):
  + **Command**: "Room\_State"
  + **Data**:
    - Available\_Rooms: List of all room names, including the new one.
    - Users\_In\_Room: [] (empty, as no users are in the room yet).
  + **Purpose**: Updates the client with the current list of rooms.
* **Key Variables**:
  + **Client**:
    - self.room\_input (QLineEdit): Holds the room name entered by the user.
    - self.list\_of\_available\_rooms (list): Stores the list of rooms received from the server.
  + **Server**:
    - self.rooms (dict): Maps room names to lists of usernames ({room\_name: [username1, username2, ...]}).
    - self.ready\_users (dict): Maps room names to dictionaries of users and their ready status ({room\_name: {username: ready\_boolean}}).
* **Process**:
  + **Client**: Sends the "Create\_Room" message with the room name and username. Updates room\_selector with new rooms received in "Room\_State".
  + **Server**: Creates a new room in self.rooms with an empty user list and initializes self.ready\_users[room\_name] as an empty dict. Broadcasts the updated room state.
* **Socket Role**: Sends the room creation request and receives the updated room list.

**3. Join Room**

**Purpose**: The user joins an existing or newly created room to chat and play.

* **File**: client.py
* **Function**: ClientMenu.Join\_room
* **User Action**: The user selects a room from room\_selector or enters a room name in room\_input and clicks the "Join Room" button.
* **Commands Sent**:
  + **Command**: "Join\_Room"
    - **Data**:
      * Room\_Name: The selected or entered room name.
      * User\_Name: The user’s username.
    - **Purpose**: Requests to join the specified room.
  + **Command**: "Request\_Room\_State"
    - **Data**:
      * User\_Name: The user’s username.
    - **Purpose**: Requests the current room state (list of rooms and users).
  + **Command**: "Sending\_Message"
    - **Data**:
      * Room\_Name: The room name.
      * User\_Name: The user’s username.
      * Text: "<username> has joined the room."
    - **Purpose**: Notifies other users in the room of the join.
* **Commands Received** (from server.py: ChatServer.handle\_client):
  + **Command**: "Join\_Room"
    - **Data**:
      * Room\_Name: The room name.
      * User\_Name: The joining user’s username.
      * Users\_In\_Room: Updated list of users in the room.
    - **Purpose**: Confirms the user has joined and provides the updated user list.
  + **Command**: "Room\_State"
    - **Data**:
      * Available\_Rooms: List of all rooms.
      * Users\_In\_Room: List of users in the specified room.
    - **Purpose**: Updates the client with the current room state.
  + **Command**: "Sending\_Message"
    - **Data**:
      * Room\_Name: The room name.
      * User\_Name: The joining user’s username.
      * Text: "<username> has joined the room."
    - **Purpose**: Displays the join message in the chat.
* **Key Variables**:
  + **Client**:
    - self.room\_name (string): Stores the name of the joined room.
    - self.list\_of\_users\_in\_room (list): Stores the list of users in the room.
    - self.alreadyinroom (boolean): Tracks if the user is in a room (True after joining).
    - self.chatroom (New\_chat\_room instance): The chat room UI window.
  + **Server**:
    - self.rooms (dict): Updated with the new user in the room.
    - self.ready\_users (dict): Adds the user with False ready status.
* **Process**:
  + **Client**: Sends "Join\_Room", "Request\_Room\_State", and "Sending\_Message". Creates a New\_chat\_room UI instance if not already in a room, and updates the chat with the join message.
  + **Server**: Adds the user to self.rooms[room\_name] and self.ready\_users[room\_name]. Broadcasts the join message and updated room state to the room.
* **Socket Role**: Facilitates joining a room and broadcasting the join event to other clients.

**4. Set Ready Status**

**Purpose**: The user indicates they are ready to play Connect 4.

* **File**: client.py
* **Function**: New\_chat\_room.toggle\_ready
* **User Action**: The user clicks the "Ready" button in the chat room window.
* **Command Sent**:
  + **Command**: "Ready\_Status"
  + **Data**:
    - Room\_Name: The current room name.
    - User\_Name: The user’s username.
    - Ready: True or False (toggles the current ready status).
  + **Purpose**: Informs the server of the user’s ready status.
* **Command Received** (from server.py: ChatServer.handle\_client):
  + **Command**: "Ready\_Update"
  + **Data**:
    - Room\_Name: The room name.
    - Ready\_Users: Dictionary of users and their ready status ({username: ready\_boolean}).
  + **Purpose**: Updates clients with the ready status of all users in the room.
* **Key Variables**:
  + **Client**:
    - self.ready\_users (dict): Stores the ready status of users in the room.
    - self.ready\_button (QPushButton): Toggles between "Ready" and "Not Ready" text and style.
  + **Server**:
    - self.ready\_users (dict): Updates the user’s ready status in the room.
* **Process**:
  + **Client**: Sends "Ready\_Status" with the toggled status. Updates the ready\_button text and style based on the new status.
  + **Server**: Updates self.ready\_users[room\_name][username] and broadcasts "Ready\_Update" to the room.
* **Socket Role**: Sends ready status updates and receives confirmation of all users’ statuses.

**5. Join Game**

**Purpose**: The game starts when two players in a room are ready.

* **File**: server.py
* **Function**: ChatServer.handle\_ready\_status
* **User Action**: Triggered automatically when two players are ready.
* **Command Sent** (server-initiated):
  + **Command**: "Game\_Start"
  + **Data**:
    - Room\_Name: The room name.
    - Game\_State: Dictionary containing:
      * grid: 6x7 game board ([[None, ...], ...]).
      * current\_player: Username of the current player.
      * current\_player\_id: 0 or 1 (player index).
      * game\_over: False (game is active).
      * winner: None (no winner yet).
      * players: List of two usernames.
  + **Purpose**: Notifies clients that the game has started and provides initial game state.
* **Key Variables**:
  + **Client**:
    - self.game\_ui (Connect4GameUI instance): The game UI window.
    - self.grid (list): Local copy of the game board.
    - self.current\_player\_id (int): ID of the current player (0 or 1).
    - self.my\_player\_id (int): The user’s player ID.
    - self.my\_turn (boolean): Whether it’s the user’s turn.
  + **Server**:
    - self.games (dict): Maps room names to Connect4Game instances.
    - Connect4Game variables:
      * self.room\_name (string): The room name.
      * self.players (list): List of two usernames.
      * self.grid (list): 6x7 game board.
      * self.current\_player (int): Current player ID (0 or 1).
      * self.game\_over (boolean): Game status.
      * self.winner (string): Username of the winner, if any.
* **Process**:
  + **Server**: Creates a Connect4Game instance for the room, randomizes player order, and broadcasts "Game\_Start" with the initial game state.
  + **Client**: Creates a Connect4GameUI instance, initializes it with the game state, and starts the game loop in a separate thread.
* **Socket Role**: Broadcasts the game start event and state to both players.

**6. Input Move**

**Purpose**: The user makes a move by selecting a column in the Connect 4 game.

* **File**: client.py
* **Function**: Connect4GameUI.game\_loop
* **User Action**: The user presses a number key (1–7) to select a column when it’s their turn.
* **Command Sent**:
  + **Command**: "Game\_Move"
  + **Data**:
    - Room\_Name: The room name.
    - User\_Name: The user’s username.
    - Column: The selected column (0–6).
  + **Purpose**: Sends the user’s move to the server for validation and processing.
* **Commands Received** (from server.py: ChatServer.handle\_game\_move):
  + **If Successful**:
    - **Command**: "Game\_Update"
    - **Data**:
      * Room\_Name: The room name.
      * Move: Dictionary with:
        + player: Username of the player who made the move.
        + column: Selected column.
        + row: Row where the chip landed.
      * Game\_State: Updated game state (grid, current player, etc.).
    - **Purpose**: Updates all clients with the new move and game state.
  + **If Game Over**:
    - **Command**: "Game\_Over"
    - **Data**:
      * Room\_Name: The room name.
      * Winner: Username of the winner (or None for a draw).
      * Game\_State: Final game state.
    - **Purpose**: Notifies clients that the game has ended and who won.
* **Key Variables**:
  + **Client**:
    - self.grid (list): Updated with the new move.
    - self.my\_turn (boolean): Updated based on game\_state["current\_player"].
  + **Server**:
    - self.grid (list): Updated with the new chip placement.
    - self.current\_player (int): Toggles between 0 and 1.
    - self.game\_over (boolean): Set to True if a win is detected.
    - self.winner (string): Set to the winner’s username if applicable.
* **Process**:
  + **Client**: Validates the move locally (is\_valid\_move), sends "Game\_Move", and updates the UI with the received game state.
  + **Server**: Validates the move (Connect4Game.add\_chip), checks for a win (check\_win), updates the game state, and broadcasts "Game\_Update" or "Game\_Over".
* **Socket Role**: Sends move requests and receives game state updates.

**7. Press Escape (Close Game Window)**

**Purpose**: The user closes the game window, ending the game UI but remaining in the chat room.

* **File**: client.py
* **Function**: Connect4GameUI.game\_loop
* **User Action**: The user closes the Pygame window (triggers pygame.QUIT event).
* **Commands Sent**: None.
* **Commands Received**: None.
* **Key Variables**:
  + **Client**:
    - self.running (boolean): Set to False to stop the game loop.
    - self.game\_ui (Connect4GameUI instance): Closed and set to None in New\_chat\_room.
* **Process**:
  + **Client**: Stops the game loop, calls pygame.quit(), and closes the game UI.
  + **Server**: No action (the game state persists on the server).
* **Socket Role**: None (local action only).

**8. Leave Room**

**Purpose**: The user leaves the chat room, notifying others and cleaning up.

* **File**: client.py
* **Function**: New\_chat\_room.closeEvent
* **User Action**: The user closes the chat room window.
* **Command Sent**:
  + **Command**: "Sending\_Message"
  + **Data**:
    - Room\_Name: The room name.
    - User\_Name: The user’s username.
    - Text: "<username> has left the room."
  + **Purpose**: Notifies the server and other users that the user has left.
* **Commands Received** (from server.py: ChatServer.handle\_client):
  + **Command**: "Join\_Room"
    - **Data**:
      * Room\_Name: The room name.
      * User\_Name: The leaving user’s username.
      * Users\_In\_Room: Updated user list (without the leaving user).
    - **Purpose**: Updates the room’s user list.
  + **Command**: "Room\_State"
    - **Data**:
      * Available\_Rooms: Updated list of rooms.
      * Users\_In\_Room: Updated user list for the room.
    - **Purpose**: Updates all clients with the current room state.
* **Key Variables**:
  + **Client**:
    - self.alreadyinroom (boolean): Set to False.
    - self.chatroom (New\_chat\_room instance): Closed and set to None.
  + **Server**:
    - self.rooms (dict): Removes the user from the room’s user list.
    - self.ready\_users (dict): Removes the user’s ready status.
    - self.games (dict): Deletes the game if the room becomes empty.
* **Process**:
  + **Client**: Sends the leave message, closes the chat room window, and resets alreadyinroom.
  + **Server**: Removes the user from self.rooms and self.ready\_users. If the room is empty, deletes it and any associated game. Broadcasts updated room state.
* **Socket Role**: Sends the leave notification and receives updated room state.

**Summary of Commands and Their Purposes**

| **Command** | **Sender** | **Purpose** |
| --- | --- | --- |
| Check\_Username | Client | Validates and registers a username with the server. |
| Create\_Room | Client | Requests the creation of a new room. |
| Join\_Room | Client | Requests to join a room and notifies others. |
| Request\_Room\_State | Client | Requests the current list of rooms and users. |
| Sending\_Message | Client | Sends chat messages or join/leave notifications. |
| Ready\_Status | Client | Updates the user’s ready status for starting a game. |
| Game\_Move | Client | Sends a Connect 4 move (column selection) to the server. |
| Restart\_Game | Client | Requests to restart the game in the room. |
| Room\_State | Server | Updates clients with the list of rooms and users. |
| Ready\_Update | Server | Updates clients with ready statuses of users in a room. |
| Game\_Start | Server | Notifies clients that a game has started and provides initial state. |
| Game\_Update | Server | Updates clients with a new move and current game state. |
| Game\_Over | Server | Notifies clients that the game has ended and who won. |
| Game\_Restart | Server | Notifies clients that the game has been restarted. |

**Key Variables and Their Roles**

| **Variable** | **File** | **Purpose** |
| --- | --- | --- |
| self.clients | server.py | Maps usernames to client sockets for communication. |
| self.rooms | server.py | Maps room names to lists of usernames in each room. |
| self.ready\_users | server.py | Tracks ready status of users in each room. |
| self.games | server.py | Maps room names to Connect4Game instances for active games. |
| self.grid (Connect4Game) | server.py | Stores the 6x7 Connect 4 game board state. |
| self.current\_player | server.py | Tracks the current player’s ID (0 or 1). |
| self.game\_over | server.py | Indicates if the game has ended. |
| self.winner | server.py | Stores the username of the game winner, if any. |
| self.client\_socket | client.py | The client’s TCP socket for server communication. |
| self.username | client.py | The user’s chosen username. |
| self.room\_name | client.py | The name of the current room (in New\_chat\_room). |
| self.list\_of\_available\_rooms | client.py | List of rooms available to join. |
| self.list\_of\_users\_in\_room | client.py | List of users in the current room. |
| self.alreadyinroom | client.py | Tracks if the user is in a room. |
| self.game\_ui | client.py | The Connect4GameUI instance for the game window. |
| self.grid (Connect4GameUI) | client.py | Local copy of the game board for rendering. |
| self.my\_turn | client.py | Indicates if it’s the user’s turn to play. |

**Flowchart Summary for Presentation**

To present this to your teacher, you can use the flowchart structure to illustrate the user journey and socket communication:

1. **Create Username**:
   * Client sends "Check\_Username" → Server responds with "Valid".
   * Variables: self.username, self.client\_socket, self.clients.
   * Socket: Establishes TCP connection.
2. **Create Room**:
   * Client sends "Create\_Room" → Server creates room and responds with "Room\_State".
   * Variables: self.rooms, self.ready\_users, self.list\_of\_available\_rooms.
   * Socket: Sends creation request, receives room list.
3. **Join Room**:
   * Client sends "Join\_Room", "Request\_Room\_State", "Sending\_Message" → Server updates room and broadcasts state.
   * Variables: self.room\_name, self.list\_of\_users\_in\_room, self.alreadyinroom.
   * Socket: Sends join request, receives room updates.
4. **Set Ready Status**:
   * Client sends "Ready\_Status" → Server updates and broadcasts "Ready\_Update".
   * Variables: self.ready\_users, self.ready\_button.
   * Socket: Sends ready status, receives updates.
5. **Join Game**:
   * Server sends "Game\_Start" when two players are ready.
   * Variables: self.games, self.game\_ui, self.grid.
   * Socket: Broadcasts game start and state.
6. **Input Move**:
   * Client sends "Game\_Move" → Server validates and sends "Game\_Update" or "Game\_Over".
   * Variables: self.grid, self.current\_player, self.my\_turn.
   * Socket: Sends move, receives game state.
7. **Press Escape (Close Game Window)**:
   * No socket communication; local UI cleanup.
   * Variables: self.running, self.game\_ui.
   * Socket: None.
8. **Leave Room**:
   * Client sends "Sending\_Message" (leave) → Server updates and broadcasts "Join\_Room", "Room\_State".
   * Variables: self.rooms, self.alreadyinroom, self.chatroom.
   * Socket: Sends leave notification, receives room updates.