COMP90015: Distributed Systems
Assignment 2 Report

Group Name: Around the corner

Tutor: Xunyun Liu

Group member: Weiguang Ma 864752
Tianning Sun 898328
Yu Chao 962031
Zihe Han 922799

Content

1. Introduction

In this project, our team implements the multiplayer online game Scrabble. The system is designed based on client-server architecture, and supports concurrent games by multiple players. To login into the system, players should register a username and access the game lobby from a welcome page. In the game lobby interface, the information of all the room is shown to the player and player can select one to enter by inputting the room number. The player who becomes the room master has the right to invite others to join the room and start the game of this room. In the game, a game interface consisting of a letter board, a scoreboard, and a chat field is presented to all players in a room. Each player in this room fills the letter board with a letter and submits a horizontal string and a vertical string in order. Both the strings should that contain the new letter. Every time when a new letter is entered, all the players in the should vote for the horizontal and vertical strings submitted. Only if all the players agree that the string is a word, the player who fills in the words will get scores equal to the length of the string. If a player clicks the pass button, he/she will skip this round of game. Only if all players choose “pass”, or a player exits, the game will end and the scoreboard will be displayed. After that, all the players will return to the game room and wait for a new round of the game. Our team structures the system and develops the protocol to deal with the network communication. Furthermore, a neat GUI is designed and implemented.

2. System Architecture

The system is designed based on client-server architecture.

2.1 Server

The server is presented by a graphical interface that shows the number of users currently connected and the messages received. The implementation of the server uses multi-thread approach. When a client connection request arrives, the server will store the connection in a ClientConnection List and create a listening thread for each connection. The server processes received messages and returns corresponding messages. If a client disconnects, the server will respond in time and remove the connection of this client from the List. In addition, the server also defines the properties of players and game rooms.

2.2 Client

The client is launched by creating a TCP socket with the server and all client-server communication is based on this socket. This socket remains open until the user exits the system. The client implements the interface design of user registration, game lobby, game room, game interface and score board. Any operation of the player on different interfaces will be converted into a corresponding message and sent to the server. Correspondingly, a thread is

created to listen to messages from the server and perform the relevant actions based on the message content.

2.3 Communication Between Components

Figure 1 briefly shows how client and server have concurrent interaction.

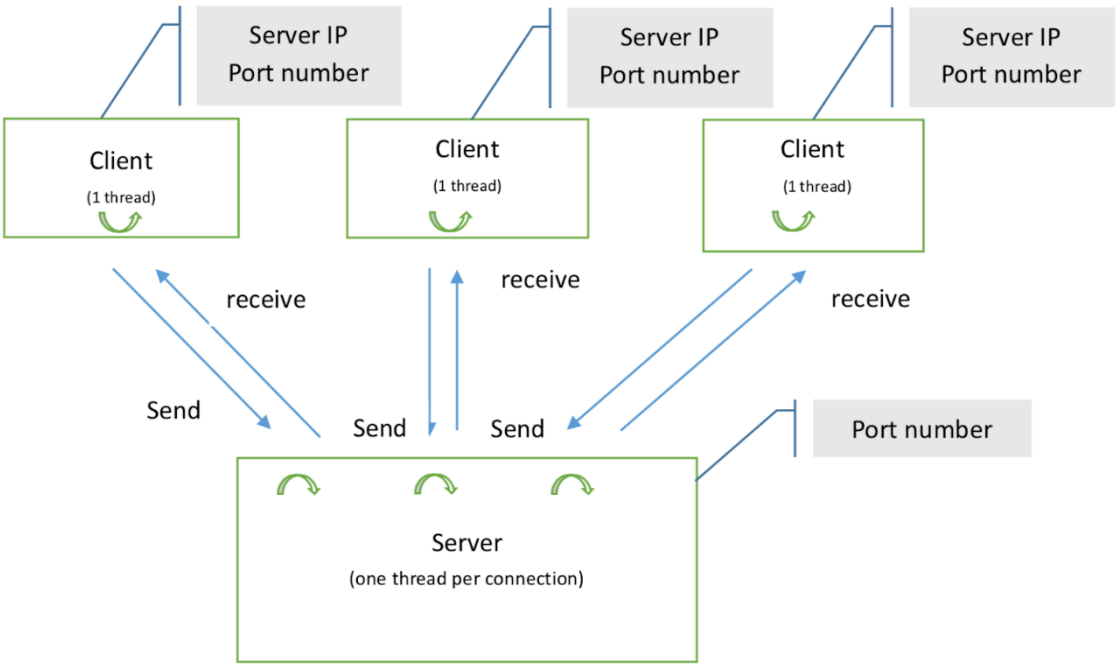

Figure 1 Client-server interaction

3. Class Design

3.1 Server Design

The server is implemented by seven classes that are encapsulated in the package server.

3.1.1 Class Implementation

(1) *Server.class*: the main class for establishing connections with requested clients. When the server is launched, it initializes a ServerSocket with the port number and waits for incoming client connection requests. It creates a thread per connection with clientSocket as parameter to perform interaction with connected clients.

(2) *ServerWindow.class*: the class for server interface. It is designed with WindowBuilder. The main components in this window are the frame, a textArea for recording sent/received messages and a textField for showing the number of online clients.

(3) *AllPlayer.class*: this class contains two ArrayLists for storing all player and players who located in the game lobby, distinguishing between the two types of players in order to send messages to different players.

(4) *ClientConnection.class*: the class for receiving all incoming messages and sending messages to corresponding clients. This class extends Thread class and overrides run() method. Its instance variables include a clientSocket, BufferedReader reader, BufferedWriter writer, etc. In the run() method, messages from the client is continuously received by a while loop, and the corresponding operation is performed according to the content of the received message. If a client disconnects, the connection corresponding to the client is removed from the list and the message that the client disconnects is displayed. In addition, this class also includes definitions of the format and content of the messages to be sent.
(5) *ClientManager.class*: This class is used to manage client connections and includes a List for storing connected clients. When a new client connects to the server, the connection is added in the List. When a client disconnects, the connection is removed from the List.
(6) *Player.class*: This class encapsulates and defines the player's properties, including the username, socket, BufferedWriter object used to send the message, and the room number of the player's room.
(7) *Room.class*: This class encapsulates and defines the properties of the game room, including the state of the room, the number of players who pressed the PASS button in the same turn, the vote confirmation, the game end confirmation, and three ArrayLists, which are used to store player information in the room, the words to be voted and the results of voting.

3.1.2 Class Diagram

The UML diagram of classes in server are shown as below.

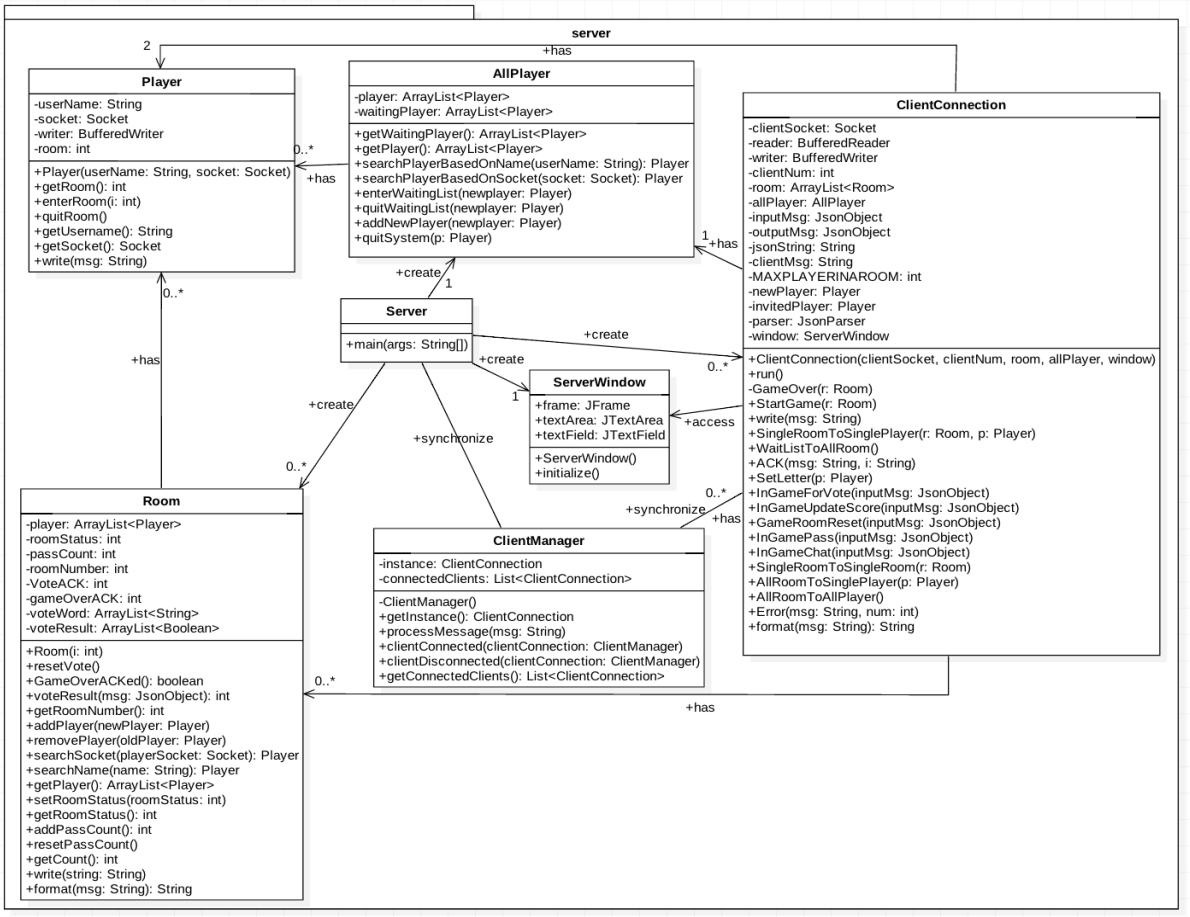

Figure 2 Server class diagram

3.1.3 Interaction between Classes

Firstly, Server.class launches an object of Class ServerWindow *window*. If a connection request arrives, the main class creates a Class ClientConnection object *clientConnection* for this connection and invokes the method getInstance() in Class ClientManager to update connection list. An arraylist of Class Room object *room* and a Class AllPlayer object *registration* are created.

ClientConnection.class invokes the methods in Class AllPlayer or Class Room according to functionality options. For example, if the function is “Register”, then the method searchPlayerBasedOnName() of Class AllPlayer object *allPlayer* will be invoked and if the username does not exist, a Class Player object *newPlayer* will be created, then adding to the *allPlayer* by method addNewPlayer(). If the client closes the connection, it closes the clientSocket and invokes the getInstance() in ClientManager class. ClientConnection.class also has the access to Class ServerWindow object *window* in order to show all messages.

The server interaction diagram is shown in Figure 3.

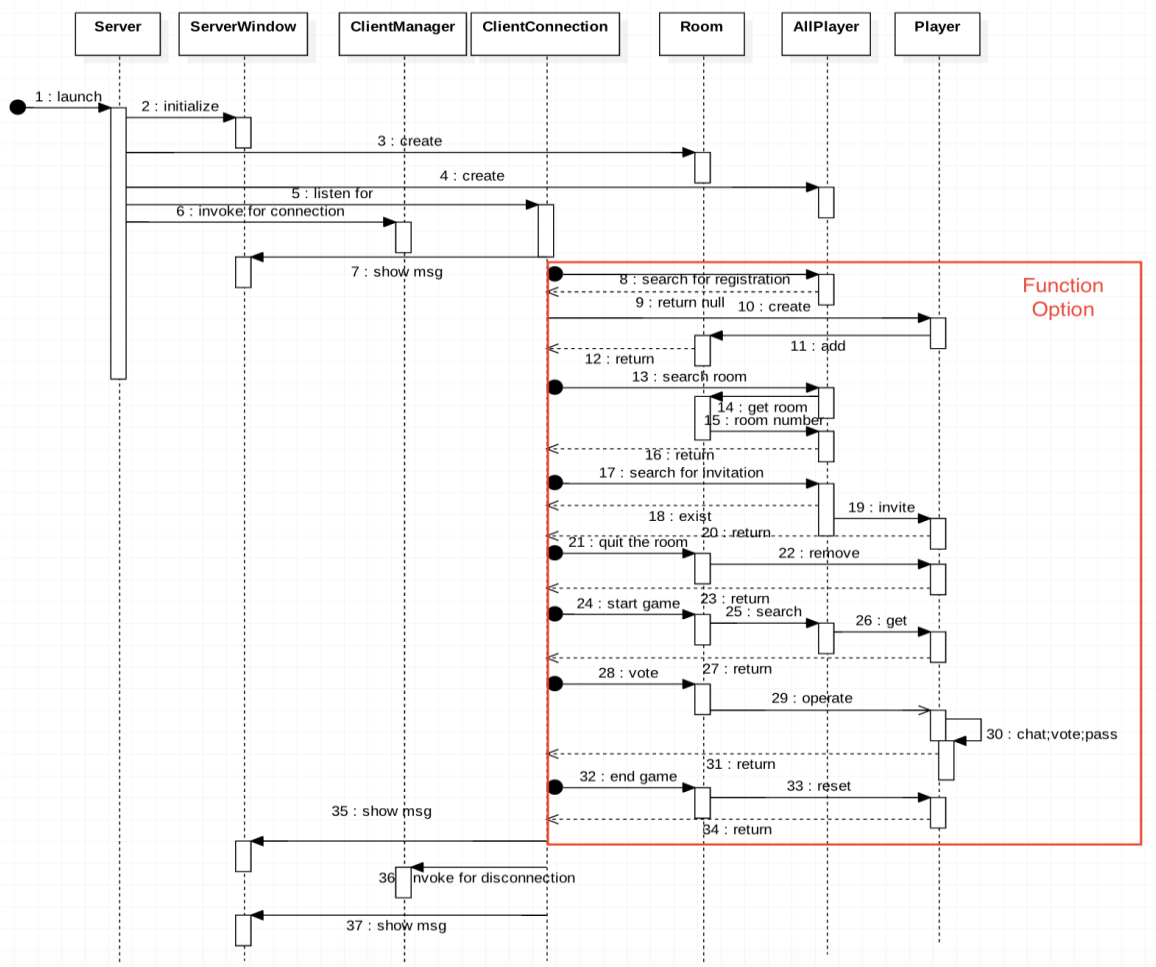

Figure 3 Server interaction diagram

3.2 Client Design

To implement a server, nine classes are encapsulated in the package client.

3.2.1 Class Implementation

(1) *Client.class*: the main class for connecting to the server. This class initializes client socket with server address and port number when it is launched. After the connection with the server establishes, BufferedReader gets the input streams for reading data from the socket and BufferedWriter gets the output streams for writing data to the socket. It creates a thread for incoming messages from the server.

(2) *ClientRegister.class*: the class for client register interface. It is designed with WindowBuilder. Players enter the game lobby by entering his or her player name.

(3) *ClientRoomList.class*: the class for game lobby interface. It is designed with WindowBuilder. All game room information is displayed. Players can enter the game room by entering the room number.

(4) *ClientGameRoom.class*: the class for game room interface. It is designed with WindowBuilder. In the game room interface, the players in the game lobby could be displayed. The room master has the ability to invite players to enter the room or start a new game. Each game room can accommodate up to six players.

(5) *GameBoard.class*: the class for game main interface. It is designed with WindowBuilder. This interface includes a letter board, a score bar, and a chat area. In each player's turn, the player can choose to fill in a letter or skip the round. The score field updates the score of all players, and the players in the game can chat with each other.
(6) *GameResult.class*: the class for game result interface. It is designed with WindowBuilder. When the game is over, this interface displays the winning players and ranks all players in descending order of score.
(7) *GamePlayer.class*: this class defines the properties of the player in the game, including the player's name and score.
(8) *MessageLister.class*: a class used to receive all messages from the server. This class extends the Thread class and overrides the run() method. Its instance variables include BufferedReader reader, BufferedWriter writer and JSON data format object inputMsg. In the run() method, messages from the server are continuously received by the while loop, and the corresponding operations are performed according to the contents of the received message.
(9) *ErrorPage.class*: this class is used to process and display various types of error messages.

3.2.2 Class Diagram

The UML diagram of classes in client are shown as below.

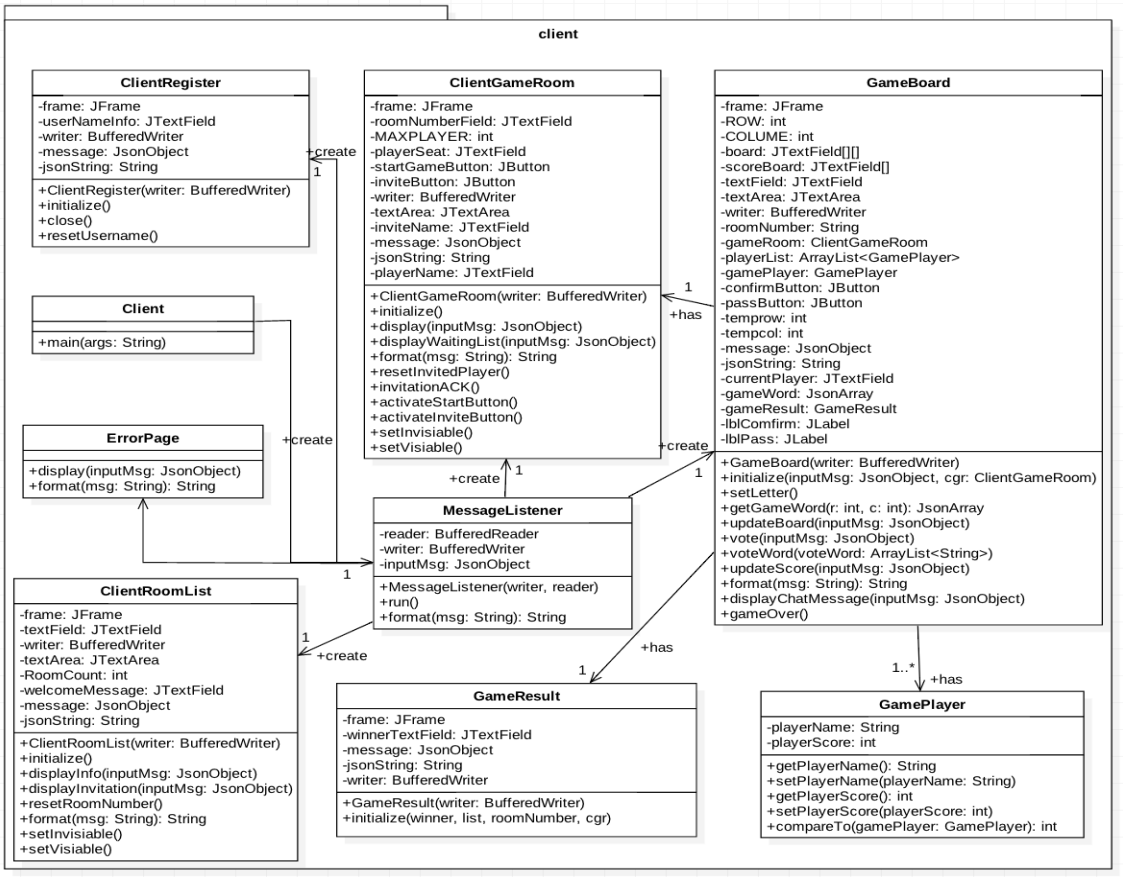

Figure 4 Client class diagram

3.2.3 Interaction Between Classes

Client.class creates a Class MessageListener object *ml* to listen for messages. MessageListener. class creates a Class ErrorPage object *errorPage,* a Class ClientRegister object *clientRegister*, a Class ClientRoomList object *clientRoomList*, a Class ClientGameRoom object *clientGameRoom* and a Class GameBoard object *gameBoard*. Then Class MessageListener initializes *clientRegister*, *clientRoomList*, *clientGameRoom* and gameBoard according to order. If gameBoard is initialized, a Class GameResult object *gameResult* and an arraylist of Class GamePlayer object *playerList* will be created. During the process of game, relating methods of Class GamePlayer or GameResult will be invoked.

The client interaction diagram is shown in Figure 5.

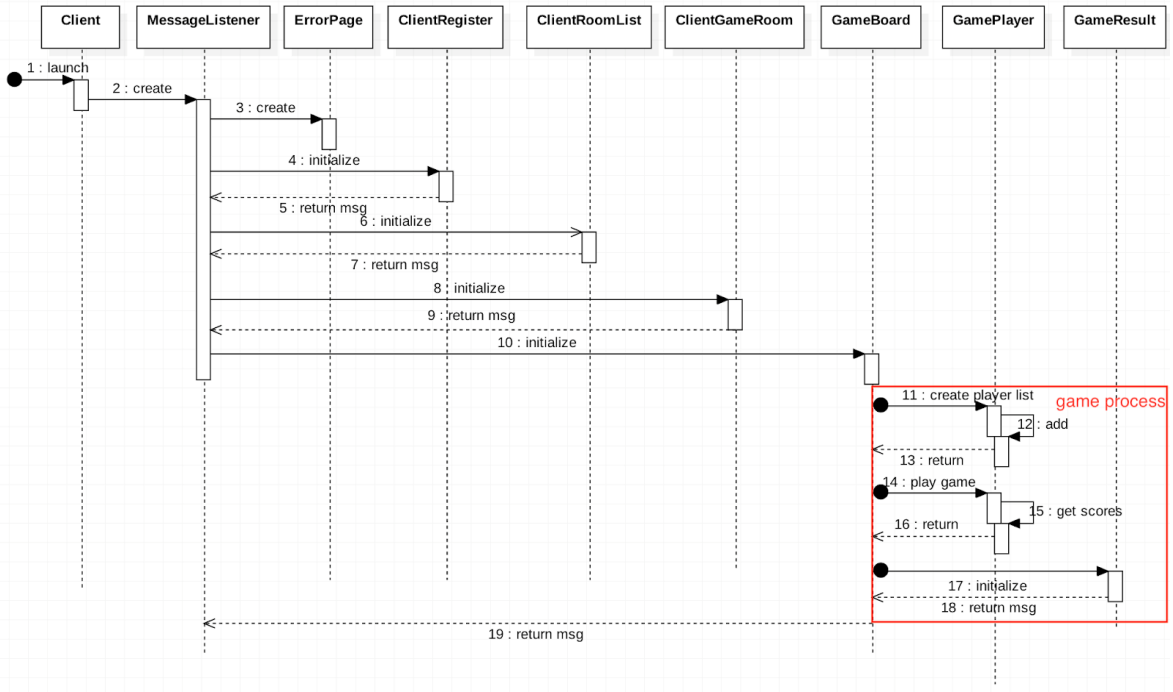

Figure 5 Client interaction diagram

3.3 Interaction Between Server and Clients

There are 9 types of transactions of the system, and the interaction between server and clients is shown in the Figures below:

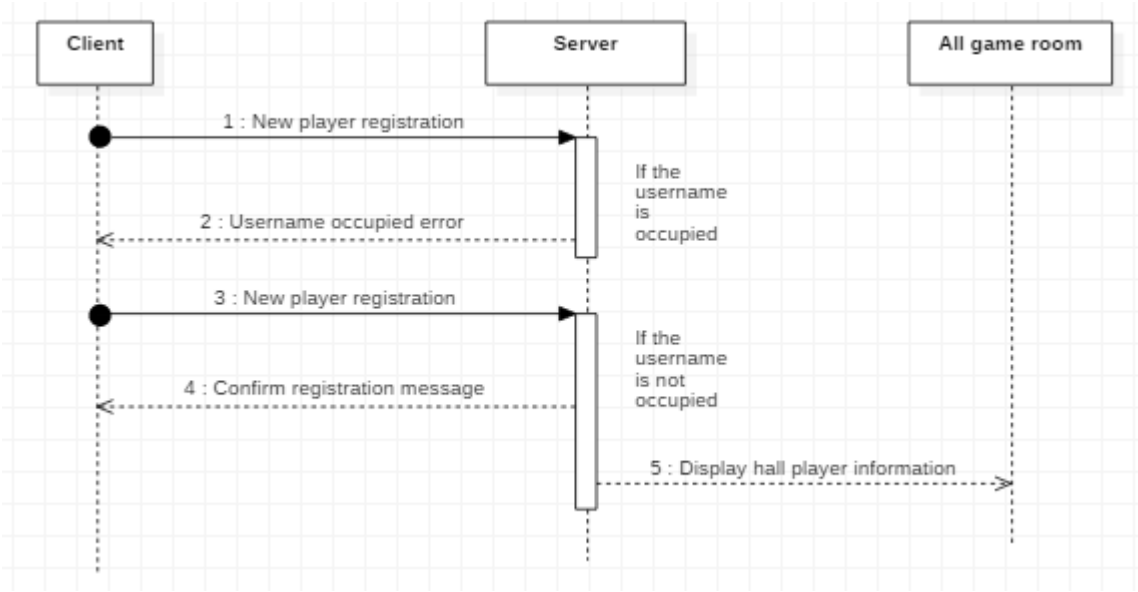

Figure 6 register transaction diagram

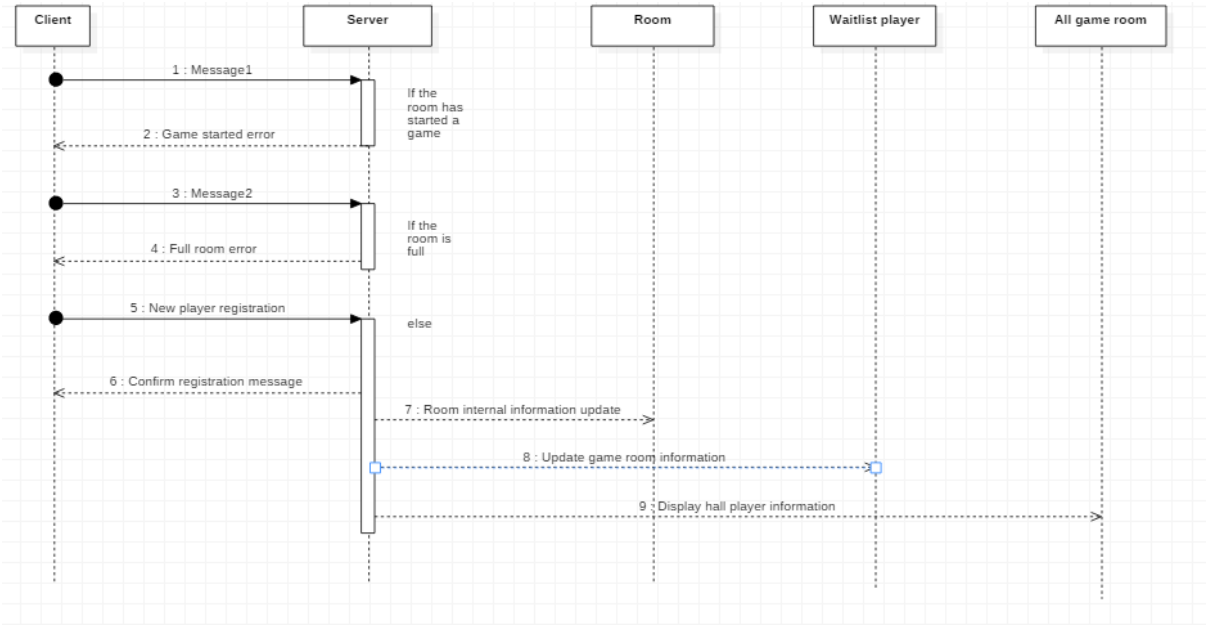

Figure 7 enter room transaction diagram

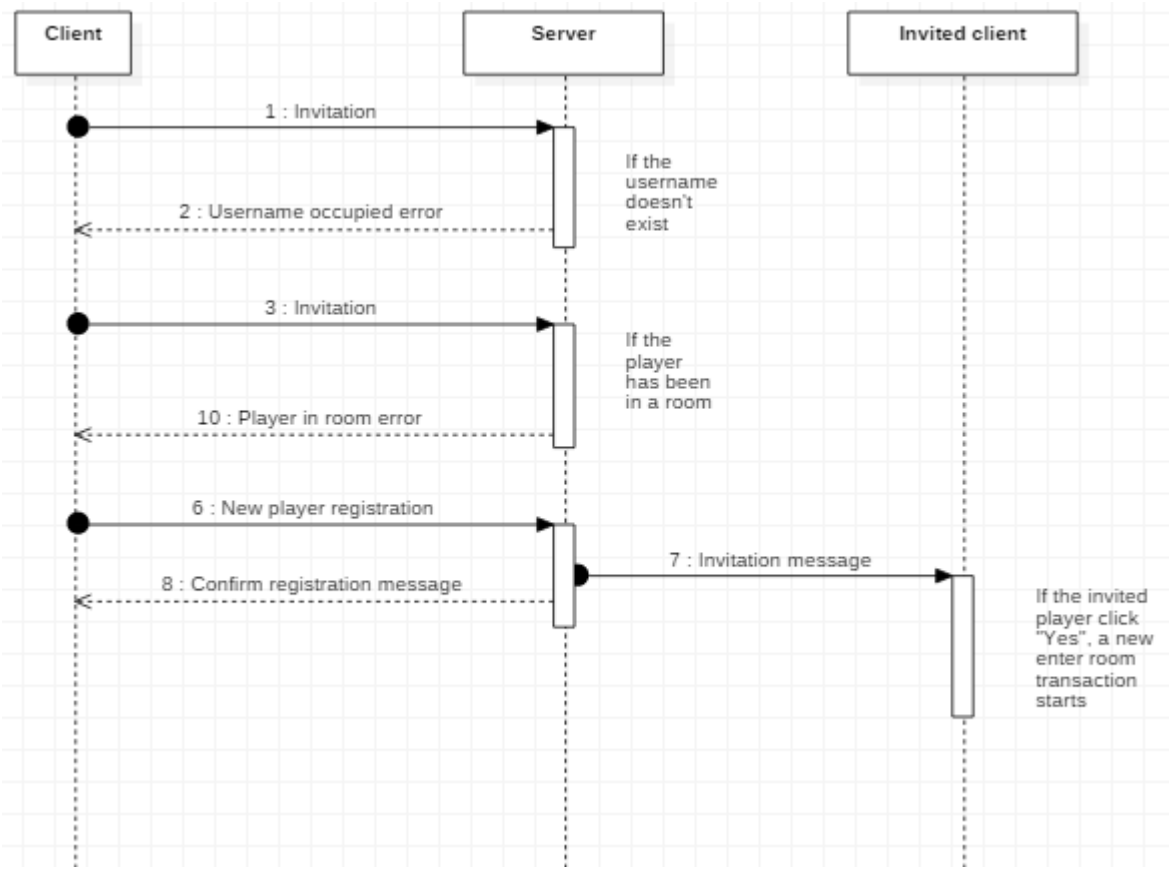

Figure 8 invitation transaction diagram

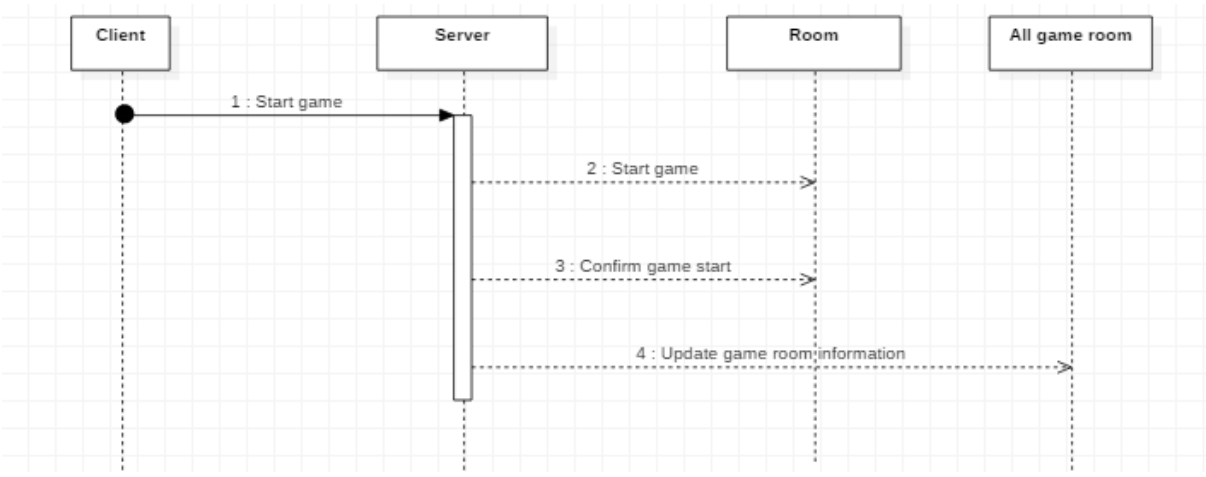

Figure 9 start game transaction diagram

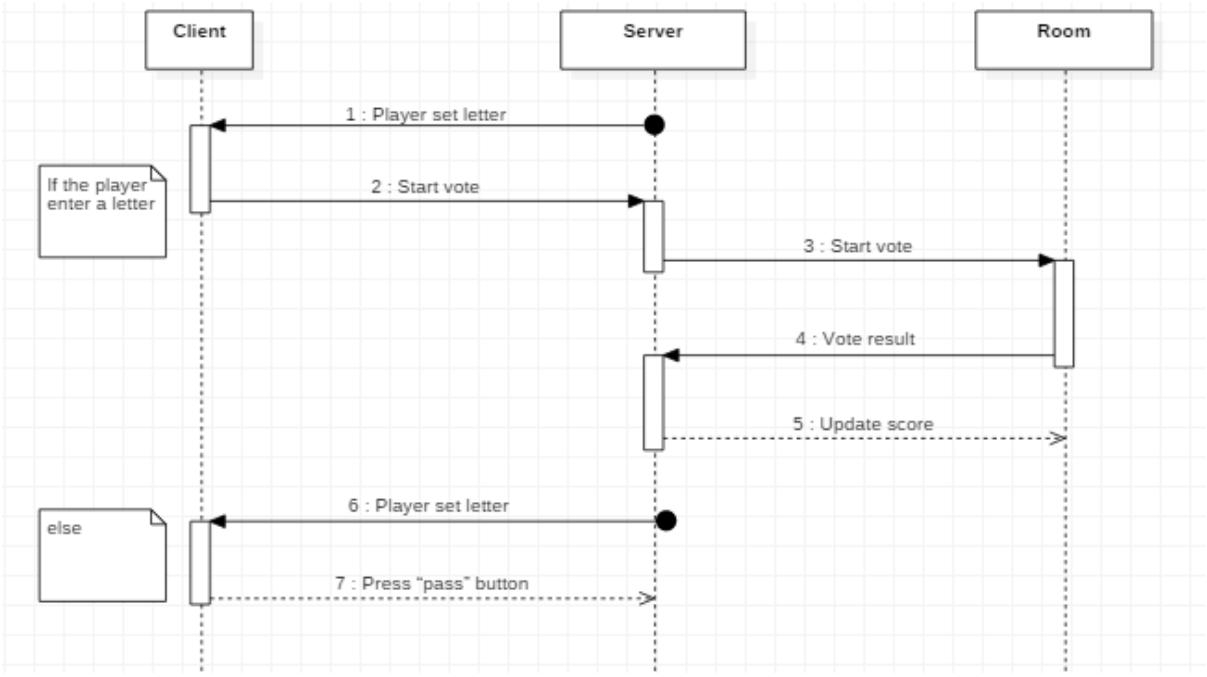

Figure 10 game transaction diagram

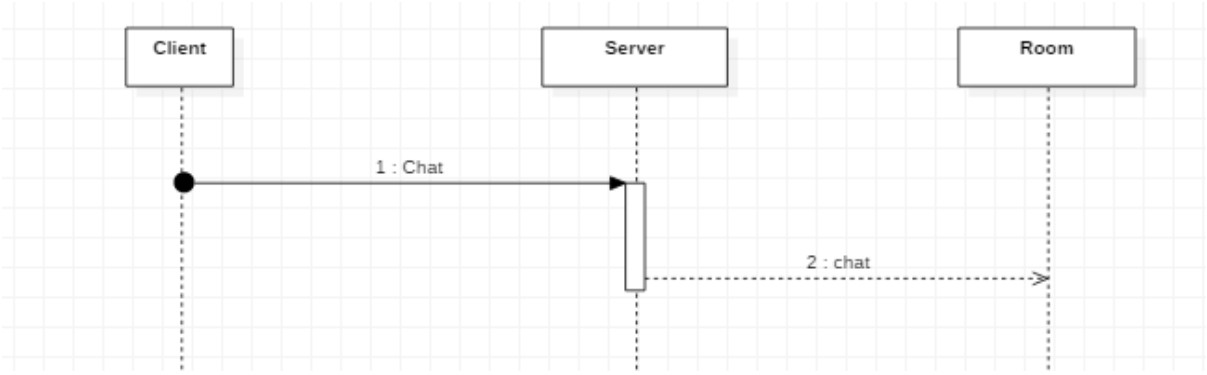

Figure 11 chat transaction diagram

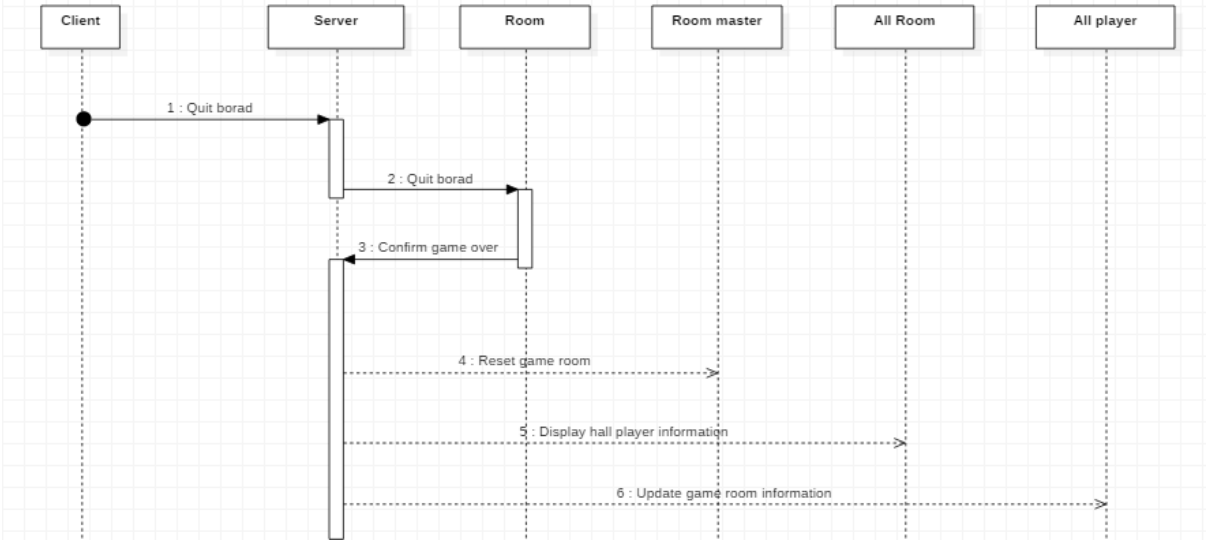

Figure 12 quit game transaction diagram

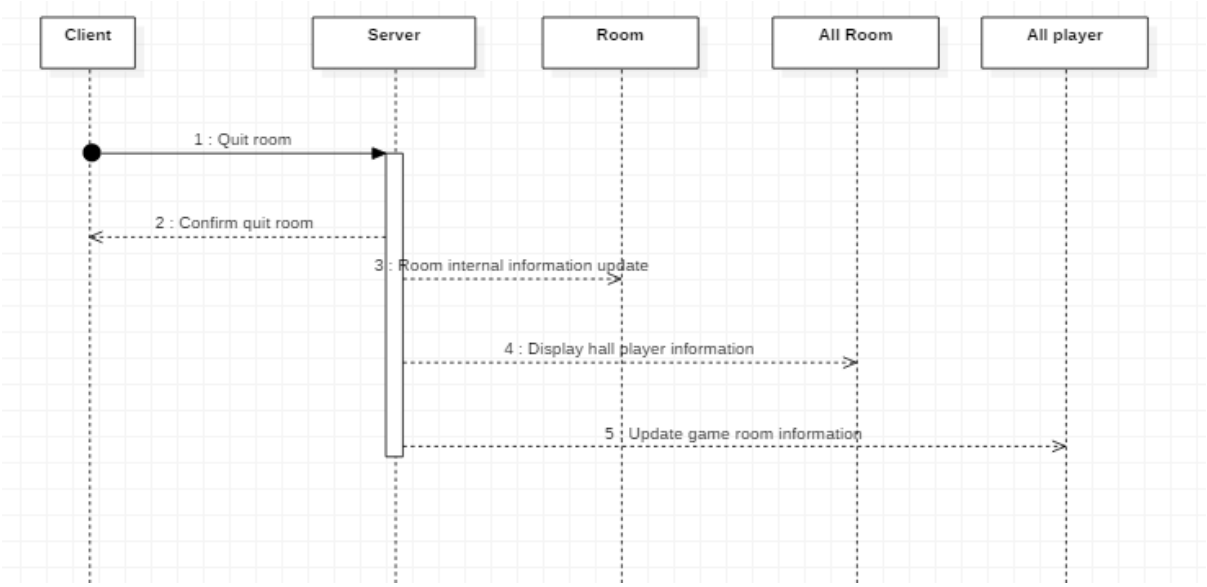

Figure 13 quit room transaction diagram

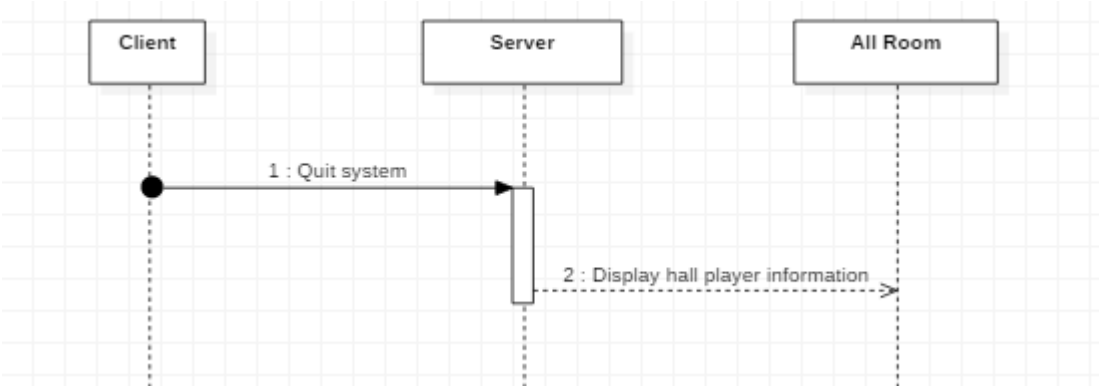

Figure 14 quit system transaction diagram

4. Protocol and Messaging Description

The system uses the Socket TCP connection mode, and the sent message types are unified into the JSON format. The message types of all communication are as follows:

New player registration	{"Direction":"ClientToServer","Function":"Register","Username":"alice"}
Enter Room	{"Direction":"ClientToServer","Function":"RequestForRoomDetail","RoomNumber":"1"}

Invitation	{"Direction":"ClientToServer","Function":"Invitation","Player":"Bob","RoomNumber":"1"}
Start game	{"Direction":"ClientToServer","Function":"StartGame","RoomNumber":"1"}
Start vote	{"Direction":"ClientToServer","Function":"InGame","RoomNumber":"1","Operation":"ForVote","Row":"8","Column":"8","Letter":"a"}
Vote result	{"Direction":"ClientToServer","Function":"InGame","Operation":"VoteResult","RoomNumber":"1","ScoredPlayer":"alice","VoteWord":[{"Word":"a","IsAWord":"Yes"}]}
Press “pass” button	{"Direction":"ClientToServer","Function":"InGame","RoomNumber":"1","PassPlayer":"Bob","Operation":"Pass"}
Chat	{"Direction":"ClientToServer","Function":"InGame","Operation":"Chat","RoomNumber":"1","Speaker":"Alice","ChatContent":"Hi"}"
Confirm game over	{"Direction":"ClientToServer","Function":"GameOverACK","RoomNumber":"1"}
Quit game	{"Direction":"ClientToServer","Function":"QuitRequest","Layer":"Board","RoomNumber":"1","QuitPlayer":"alice"}
Quit room	{"Direction":"ClientToServer","Function":"QuitRequest","Layer":"Room","RoomNumber":"1"}
Quit system	{"Direction":"ClientToServer","Function":"QuitRequest","Layer":"System"}

Table 1: Client message

Confirm registration message	{"Direction":"ServerToClient","Type":"ACK","ACKType":"0","Message":"{\"Direction\":\"ClientToServer\",\"Function\":\"Register\",\"Username\":\"alice\"}"}
Display game room information	{"Direction":"ServerToClient","Type":"AllRoomToSinglePlayer","RoomList":[{"RoomNumber":"0","RoomStatus":"Ready","PlayerNumber":"0","RoomMaster":"None"},{"RoomNumber":"1","RoomStatus":"Ready","PlayerNumber":"0","RoomMaster":"None"},{"RoomNumber":"2","RoomStatus":"Ready","PlayerNumber":"0","RoomMaster":"None"}],"Destination":"alice"}

Confirm enter room message	{"Direction":"ServerToClient","Type":"ACK","ACKType":"1","Message":"{\"Direction\":\"ClientToServer\",\"Function\":\"RequestForRoomDetail\",\"RoomNumber\":\"1\"}"}
Room internal information update	{"Direction":"ServerToClient","Type":"SingleRoomToSingleRoom","RoomNumber":"1","PlayerList":[{"PlayerName":"alice"}],"Destination":"alice"}
Display hall player information	{"Direction":"ServerToClient","Type":"WaitListToAllRoom","WaitingList":[],"Destination":"alice"}
Confirm invitation message	{"Direction":"ServerToClient","Type":"ACK","ACKType":"2","Message":"{\"Direction\":\"ClientToServer\",\"Function\":\"Invitation\",\"Player\":\"Bob\",\"RoomNumber\":\"1\"}"}
Invitation message	{"Direction":"ServerToClient","Type":"SingleRoomToSinglePlayer","RoomNumber":"1","Message":"alice invites you to have a game in Room 1","Destination":"Bob"}
Start game	{"Direction":"ServerToClient","Type":"StartGame","RoomNumber":"1","PlayerList":[{"PlayerName":"alice"},{"PlayerName":"Bob"}],"Destination":"alice"}
Update game room information	{"Direction":"ServerToClient","Type":"AllRoomToSinglePlayer","RoomList":[{"RoomNumber":"0","RoomStatus":"Ready","PlayerNumber":"0","RoomMaster":"None"},{"RoomNumber":"1","RoomStatus":"Playing","PlayerNumber":"2","RoomMaster":"alice"},{"RoomNumber":"2","RoomStatus":"Ready","PlayerNumber":"0","RoomMaster":"None"}],"Destination":"alice"}
Player set letter	{"Direction":"ServerToClient","Type":"InGame","Operation":"SetLetter","RoomNumber":"1","Destination":"alice"}
Confirm game start	{"Direction":"ServerToClient","Type":"ACK","ACKType":"3","Message":"Your room has started a game"}
Start vote	{"Direction":"ServerToClient","Type":"InGame","Operation":"ForVote","RoomNumber":"1","Row":"8","Column":"8","Letter":"a","ScoredPlayer":"alice","Destination":"Bob"}
Update score	{"Direction":"ServerToClient","Type":"InGame","Operation":"UpdateScore","RoomNumber":"1","Score":"1","ScoredPlayer":"alice","Destination":"Bob"}

Chat	{"Direction":"ServerToClient","Type":"InGame","Operation":"Chat","RoomNumber":"1","Speaker":"Alice","ChatContent":"Hi","Destination":"Alice"}
Game over	{"Direction":"ServerToClient","Type":"GameOver","RoomNumber":"1","Destination":"Bob"}
Reset game room	{"Direction":"ServerToClient","Type":"GameRoomReset","CanStart":"Yes"}
Confirm game over	{"Direction":"ServerToClient","Type":"ACK","ACKType":"5","Message":"You have quit game in Room 1"}
Confirm quit room	{"Direction":"ServerToClient","Type":"ACK","ACKType":"4","Message":"You have quit game in Room 1"}

Table 2: Server message

5. Error Control

5.1 Command Line Startup

● Port number contains illegal characters: Display error message “You must input a valid port number”.

● Number of incorrect input parameters: Display error message “Number of parameters incorrect”.

● Start another server with the same port number: Display error message “Port is occupied, please try again”.

● Client connected to the wrong IP address or port number or the server is not started: Display error message “Connection Failure”.

5.2 Player Registration

● Player name is empty: Display error message “You have to input a player name”.

● Player name contains invalid characters: Display error message “The username contains invalid characters”.

● Player name exists: Display error message “Username exists”.

5.3 Enter Room

● Room number is empty: Display error message “You have to input room number”.

● Input an invalid room number: Display error message “Invalid room number”.

5.4 Invite Player

● Invite a player who does not exist: Display error message “This player does not exist”.

● Invite a player already in a room: Display error message “This player has been in a room”.

● Player name contains invalid characters: Display error message “Invalid player name format”.

5.5 Input Letter

● Do not enter any letters or enter multiple letters: Display error message “You should only input 1 letter”.

● Enter illegal characters: Display error message “You should only input 1 letter”.

5.6 Abnormal Exit

● The client quits abnormally: Even if the client computer goes down or directly ends the process, the client's connection can be removed from the List normally. If the player is in the game, the game will end normally.

● The server quits abnormally: When the client sends a message to the server again, it will display "Fail to connect server" and exit the interface.

6. Innovative Function

6.1 Server UI Design

A user interface for server is included in the project. All messages exchanged between the server and clients are shown in the text area and the number of current connections is shown at the top right-hand corner.

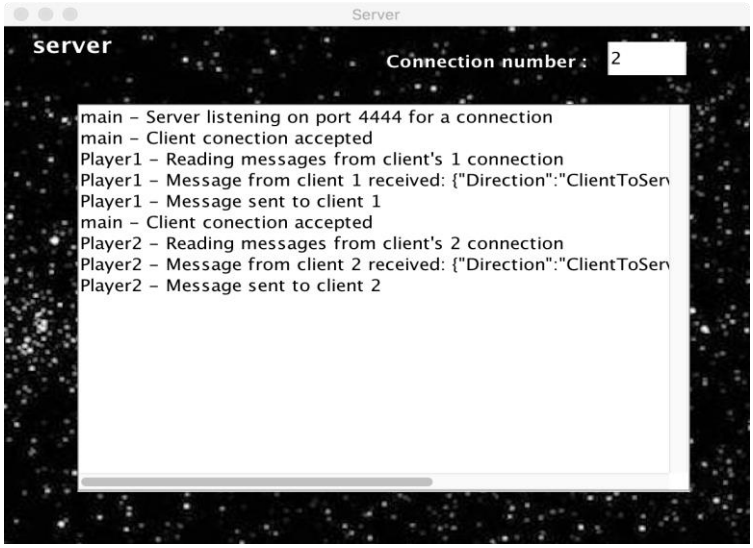

Figure 15 Server UI design

6.2 Game Lobby

The design of game lobby is shown in Figure 16. If a player registers successfully, he/she will enter the game lobby, where a series of game rooms are listed. The player can try to enter any room by inputting the room number. Game Lobby updates the status of room players in real time.

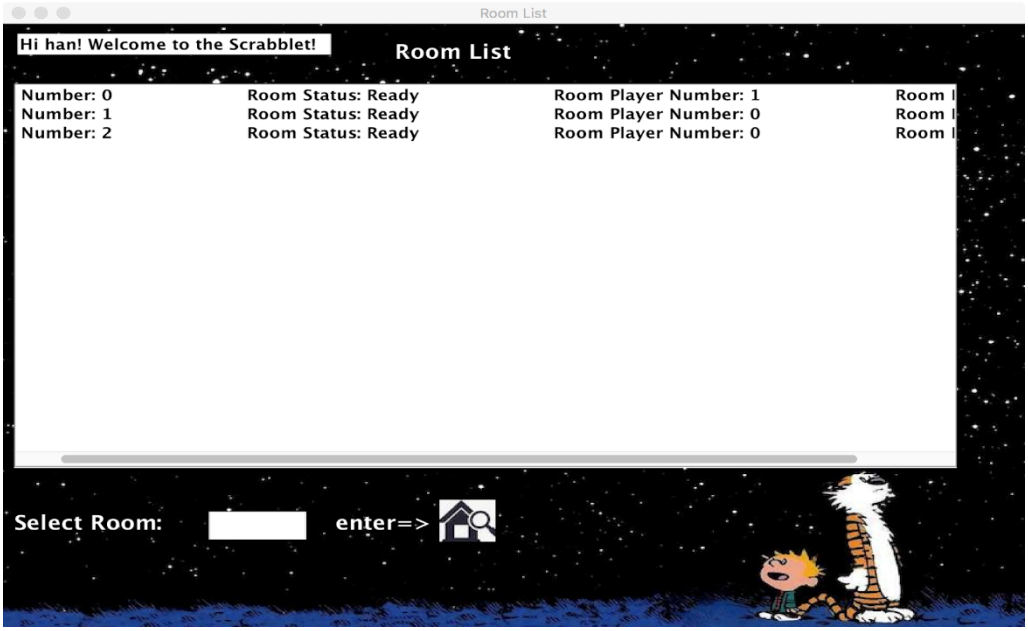

Figure 16 Game lobby

6.3 Game Room

The design of game room is another innovative function of the project. It can be seen in Figure 17. The first person entering each room is deemed to be the room master and have the right to invite other players to this room and start the game as long as the number of players is more than 2. Multiple rooms can start games concurrently.

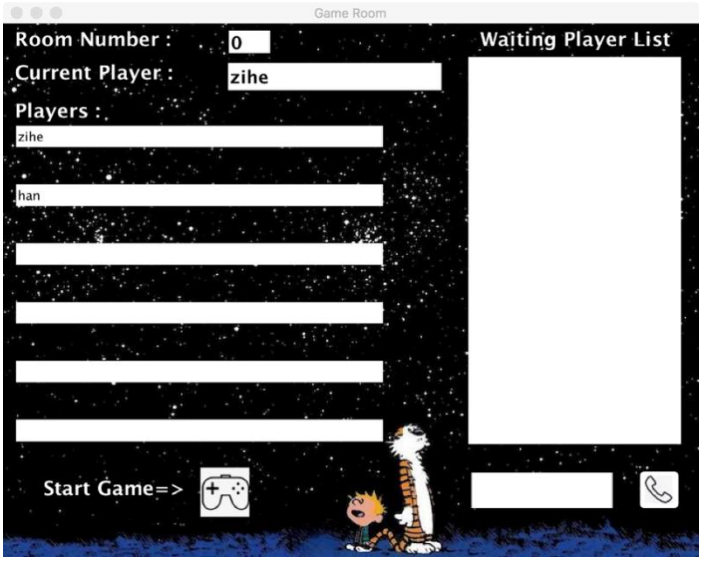

Figure 17 Game room

6.4 Excellent Error Control

Except for general error handling, various kinds of abnormalities are taking into consideration, such as exception in which the player exits the system abruptly during the game. More details about error control describes in section 5.

7. Member contribution

Weiguang Ma: Team Leader of this project, communication protocol development; user interface construction; partial exception handling and report writing.

Tianning Sun: Conceptual and logical model design; system, communication protocol; transaction design and development; report reviewing.

Chao Yu: Data structure definition; underlying implementation; system function test; System, communication protocol; transaction development.

Zihe Han: data structure definition; server graphical interface design; client GUI optimization; report writing and class diagram; interaction diagram design.