COMP 90015 Distributed Systems – Assignment 2

Scrabble Game

Team Name: Triple H1

Team Member: Chenyang Lu (951933_Chenyangl5)

Pengcheng Yao(886326_pyao1)

Jing Du(775074_du2)

Yu Dong(928922_dong4)

Tutor: Alisha Aneja

1 System Introduction

The Multi-threaded Scrabble Game is an online game system. It requires a client-server architecture to design and implement a multi-threaded server that allows concurrent clients connect to sever and join the game. If no game is in progress, any user can create a game and invite other online users to join the game. In each round of game, players place letters on tiles of grid to make words in turns. After adding a letter to the grid, player can select a word composed of this letter. This “word” will be judged by all other players, if they all agree this “word” is a real word, this player will get a score equal to the length of the word. At the end of the game, the winner will be announced. To implement this game in the distributed system, the architecture, protocols and messaging format need to be considered. This report will introduce the system in these aspects.

2 Communication Protocol

The communication model between the client and the server of this system is the TCP protocol. TCP is a connection-oriented protocol that guarantees reliable transmission. However, UDP send independent datagrams between server and client without guaranteeing the arrival and sequence. In the TCP protocol, a connection must be established before data is transmitted, and each TCP connection can have only two endpoints. From the socket perspective, the communication parties need to establish a socket, which is composed of IP address and port number, and the data will be sent to the application corresponding to the port after reaching the transport layer.

3 Message Description

In this system, the information is transmitted between client and server in json format and the client and server side start a thread, respectively, to listen for any message. JSON is a light, friendly data format. Messages are written to the json file using a concise key-value pair. The first key value is defined as "Command Type". The remaining key values are other valid information contained in the message. The format of the json file looks like this: { "command_type": Login, "username": xxx }. The server and client will use the same command_type name when processing the same business function. Whether the client or the server receives the information, it will first check the command_type and then transfer to the corresponding business process. For example, when a user enters a username and attempts to log in to the game, the client writes the user information to the Json file and sends it to the client. The specific file content looks like this: { "command_type": "login": "username" : "xxx"}. After the server determines the command type, it will process the login service and return the confirmation message of the login to the client. The specific file content looks like this: { "command_type": "login", "check_username": True or False, "username": "xxx" }. If the Boolean check_username is True, the system will store the username. Otherwise, the user will be prompted to re-enter another username. This ensures clear business processing logic.

4 Architecture

4.1 Threading model

This Dictionary system chooses the Many to Many Model. In this model, we have multiple client connection threads mapped to many threads in server. Each connection has one thread, and all connections are managed in one list in ServerState. Concurrency of this model is improved up to the number of client threads, but performance is not gained because kernel can schedule only one thread at a time.

4.2 Class Implementation

	Class Name	Description
Server	Server	In Server class, A socket to listen for client connection requests is created. After accepting the incoming

		connection request from the client, the server creates one thread per connection. Each thread is responsible for listening the message from the client and passing the message to the client manager.
	Game	The purpose of the Game class is to manage the real-time record of the game state and manage the game process. The game status includes: whether there is an ongoing game, the game player list, the score list, the player permissions, and the vote results. And the game status will be broadcast to all players in time to ensure that all users update synchronously.
	ClientManager	The purpose of the ClientManager class is to manage all user threads, which are stored in a list and added or deleted in real time. The functions that realize broadcast information to users are also be defined in this class.
	ClientConnection	The ClientConnection class is mainly responsible for receiving and processing the message coming from clients, and timely updating the system status by analyzing these messages. After that the corresponding responses were returned to the clients.
Client	Client	The purpose of Client class is to create the client information listener.
	MessageListener	The MessageListener class is mainly responsible for creating input and output streams.
	Controller	Analyze the information sent by the server and make corresponding updates or calls to the different client GUI.
	LoginGUI	This class implements the user login interface， which allows the user to enter the appropriate user name and complete the login. If the user enters a user name that is not adopted by the system, the corresponding prompt message will be displayed on the interface
	WaitRoomGUI	This class implements a user interface, which is shown after the user has logged in. This interface allows the user to see a list of all online users, create a game or exit the system.
	InviteGUI	This class implements a user interface, which is shown after the user has press the “create” button and server also allows him/her to create a game. In this user interface， all users who can be invited will be listed. The client can send the game invitation to the corresponding player by clicking the user name.

	StartGameGUI	This class implements a game interface, client can add a letter to the grid and circle the words when they have the appropriate authority.

4.3 Program Framework

The operation of the client is implemented by seven java classes： Client, Controller, MessageListener, LoginGUI, WaitRoomGUI, InviteGUI, startGameGUI. The UML of client is showed in *Fig.1*.

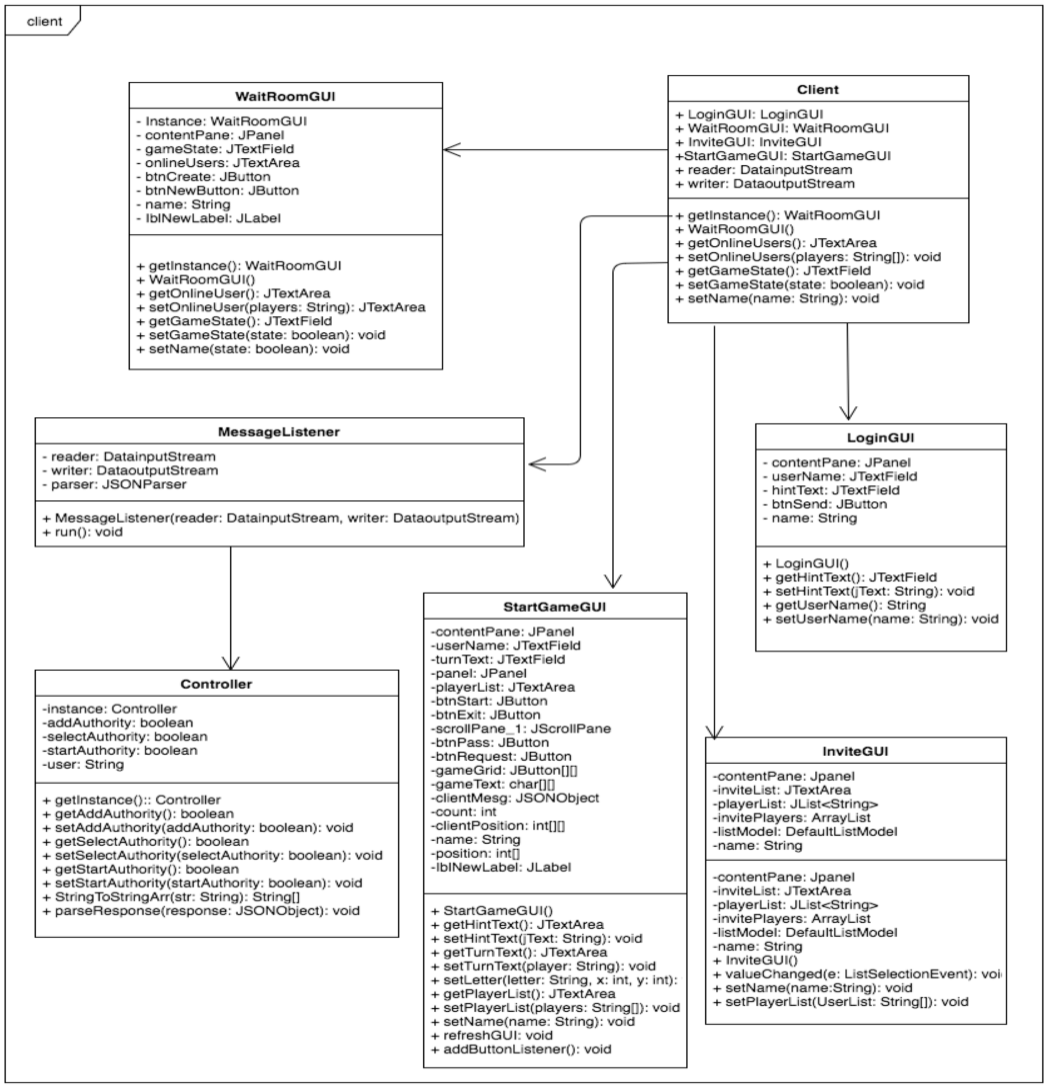

client
WaitRoomGUI
- Instance: WaitRoomGUI
- contentPane: JPanel
- gameState: JTextField
- onlineUsers: JTextArea
- btnCreate: JButton
- btnNewButton: JButton
- name: String
- lblNewLabel: JLabel
+ getInstance(): WaitRoomGUI
+ WaitRoomGUI()
+ getOnlineUser(): JTextArea
+ setOnlineUser(players: String): JTextArea
+ getGameState(): JTextField
+ setGameState(state: boolean): void
+ setName(state: boolean): void
Client
+ LoginGUI: LoginGUI
+ WaitRoomGUI: WaitRoomGUI
+ InviteGUI: InviteGUI
+StartGameGUI: StartGameGUI
+ reader: DatainputStream
+ writer: DataoutputStream
+ getInstance(): WaitRoomGUI
+ WaitRoomGUI()
+ getOnlineUsers(): JTextArea
+ setOnlineUsers(players: String[]): void
+ getGameState(): JTextField
+ setGameState(state: boolean): void
+ setName(name: String): void
MessageListener
- reader: DatainputStream
- writer: DataoutputStream
- parser: JSONParser
+ MessageListener(reader: DatainputStream, writer: DataoutputStream)
+ run(): void
LoginGUI
- contentPane: JPanel
- userName: JTextField
- hintText: JTextField
- btnSend: JButton
- name: String
+ LoginGUI()
+ getHintText(): JTextField
+ setHintText(jText: String): void
+ getUserName(): String
+ setUserName(name: String): void
StartGameGUI
-contentPane: JPanel
-userName: JTextField
-turnText: JTextField
-panel: JPanel
-playerList: JTextArea
-btnStart: JButton
-btnExit: JButton
-scrollPane_1: JScrollPane
-btnPass: JButton
-btnRequest: JButton
-gameGrid: JButton[][]
-gameText: char[][]
-clientMesg: JSONObject
-count: int
-clientPosition: int[][]
-name: String
-position: int[]
-lblNewLabel: JLabel
+ StartGameGUI()
+ getHintText(): JTextArea
+ setHintText(jText: String): void
+ getTurnText(): JTextArea
+ setTurnText(player: String): void
+ setLetter(letter: String, x: int, y: int):
+ getPlayerList(): JTextArea
+ setPlayerList(players: String[]): void
+ setName(name: String): void
+ refreshGUI: void
+ addButtonListener(): void
Controller
-instance: Controller
-addAuthority: boolean
-selectAuthority: boolean
-startAuthority: boolean
-user: String
+ getInstance():: Controller
+ getAddAuthority(): boolean
+ setAddAuthority(addAuthority: boolean): void
+ getSelectAuthority(): boolean
+ setSelectAuthority(selectAuthority: boolean): void
+ getStartAuthority(): boolean
+ setStartAuthority(startAuthority: boolean): void
+ StringToStringArr(str: String): String[]
+ parseResponse(response: JSONObject): void
InviteGUI
-contentPane: Jpanel
-inviteList: JTextArea
-playerList: JList<String>
-invitePlayers: ArrayList
-listModel: DefaultListModel
-name: String
-contentPane: Jpanel
-inviteList: JTextArea
-playerList: JList<String>
-invitePlayers: ArrayList
-listModel: DefaultListModel
-name: String
+ InviteGUI()
+ valueChanged(e: ListSelectionEvent): voi
+ setName(name:String): void
+ setPlayerList(UserList: String[]): void

Fig.1: client UML

The operation of the server is implemented by five java classes： Server, ServerGUI, and ClientManager, ClientConnection, Game. The UML of server is showed in *Fig.2*.

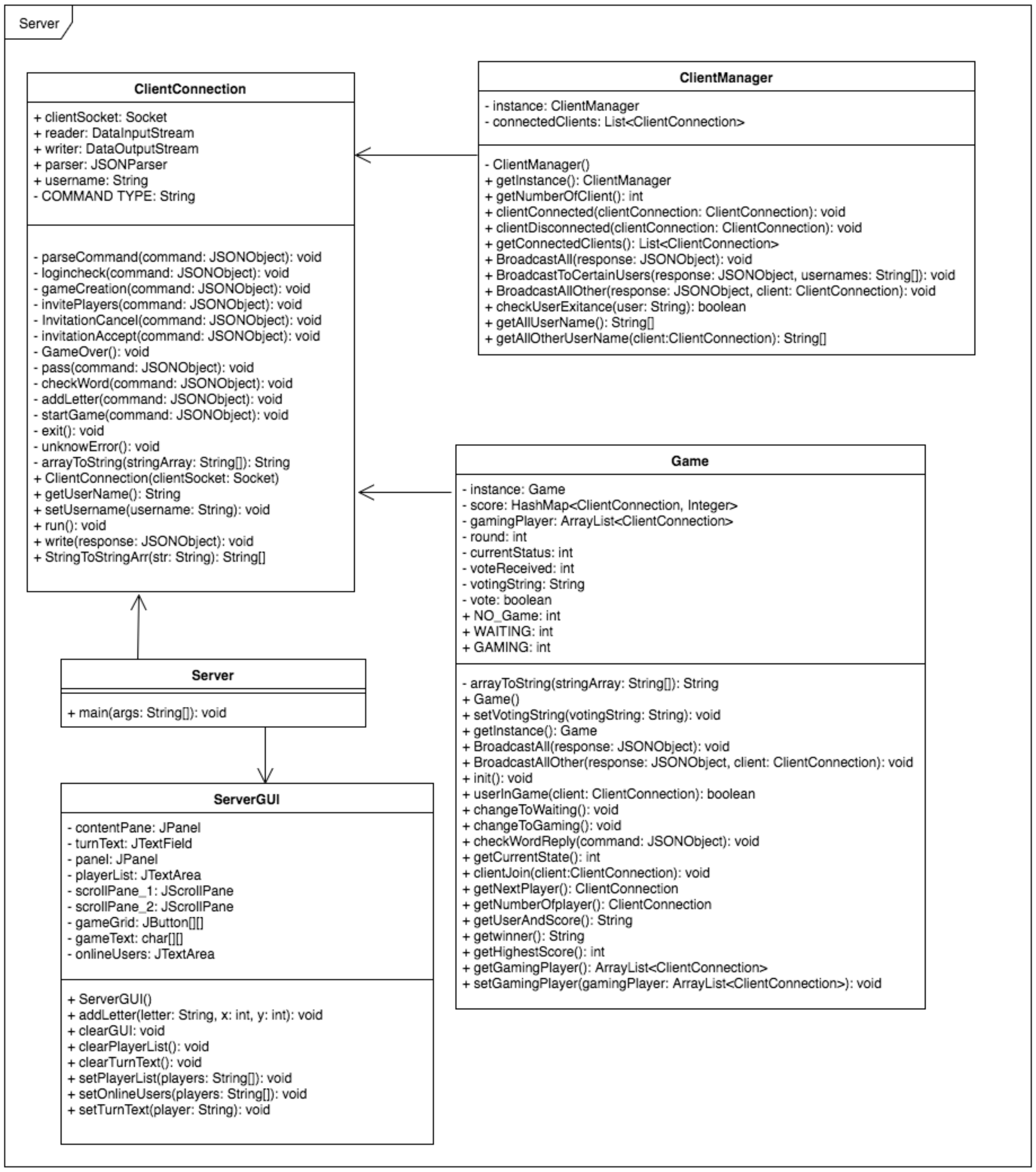

Server
ClientConnection
+ clientSocket: Socket
+ reader: DataInputStream
+ writer: DataOutputStream
+ parser: JSONParser
+ username: String
- COMMAND TYPE: String
- parseCommand(command: JSONObject): void
- logincheck(command: JSONObject): void
- gameCreation(command: JSONObject): void
- invitePlayers(command: JSONObject): void
- InvitationCancel(command: JSONObject): void
- invitationAccept(command: JSONObject): void
- GameOver(): void
- pass(command: JSONObject): void
- checkWord(command: JSONObject): void
- addLetter(command: JSONObject): void
- startGame(command: JSONObject): void
- exit(): void
- unknowError(): void
- arrayToString(stringArray: String[]): String
+ ClientConnection(clientSocket: Socket)
+ getUserName(): String
+ setUsername(username: String): void
+ run(): void
+ write(response: JSONObject): void
+ StringToStringArr(str: String): String[]
ClientManager
- instance: ClientManager
- connectedClients: List<ClientConnection>
- ClientManager()
+ getInstance(): ClientManager
+ getNumberOfClient(): int
+ clientConnected(clientConnection: ClientConnection): void
+ clientDisconnected(clientConnection: ClientConnection): void
+ getConnectedClients(): List<ClientConnection>
+ BroadcastAll(response: JSONObject): void
+ BroadcastToCertainUsers(response: JSONObject, usernames: String[]): void
+ BroadcastAllOther(response: JSONObject, client: ClientConnection): void
+ checkUserExitance(user: String): boolean
+ getAllUserName(): String[]
+ getAllOtherUserName(client:ClientConnection): String[]
Game
- instance: Game
- score: HashMap<ClientConnection, Integer>
- gamingPlayer: ArrayList<ClientConnection>
- round: int
- currentStatus: int
- voteReceived: int
- votingString: String
- vote: boolean
+ NO_Game: int
+ WAITING: int
+ GAMING: int
- arrayToString(stringArray: String[]): String
+ Game()
+ setVotingString(votingString: String): void
+ getInstance(): Game
+ BroadcastAll(response: JSONObject): void
+ BroadcastAllOther(response: JSONObject, client: ClientConnection): void
+ init(): void
+ userInGame(client: ClientConnection): boolean
+ changeToWaiting(): void
+ changeToGaming(): void
+ checkWordReply(command: JSONObject): void
+ getCurrentState(): int
+ clientJoin(client:ClientConnection): void
+ getNextPlayer(): ClientConnection
+ getNumberOfplayer(): ClientConnection
+ getUserAndScore(): String
+ getwinner(): String
+ getHighestScore(): int
+ getGamingPlayer(): ArrayList<ClientConnection>
+ setGamingPlayer(gamingPlayer: ArrayList<ClientConnection>): void
Server
+ main(args: String[]): void
ServerGUI
- contentPane: JPanel
- turnText: JTextField
- panel: JPanel
- playerList: JTextArea
- scrollPane_1: JScrollPane
- scrollPane_2: JScrollPane
- gameGrid: JButton[][]
- gameText: char[][]
- onlineUsers: JTextArea
+ ServerGUI()
+ addLetter(letter: String, x: int, y: int): void
+ clearGUI: void
+ clearPlayerList(): void
+ clearTurnText(): void
+ setPlayerList(players: String[]): void
+ setOnlineUsers(players: String[]): void
+ setTurnText(player: String): void

Fig.2: server UML

The sequence diagram of this system is showed in *Fig.3*:

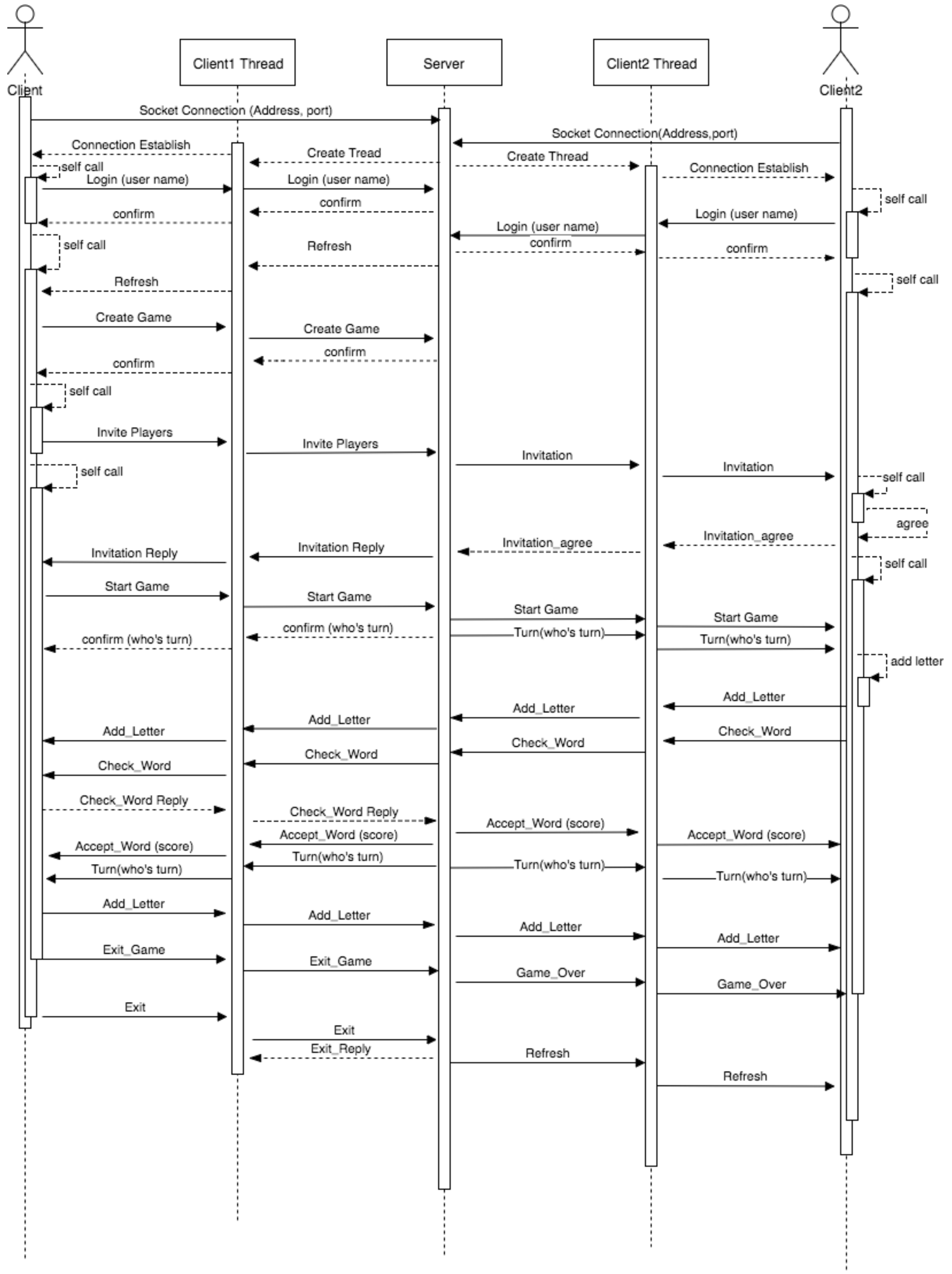


Fig.3: Sequence Diagram

5 Interactive interface

The graphical interface for both client side and server side are all provided in this system.

5.1 Client GUI

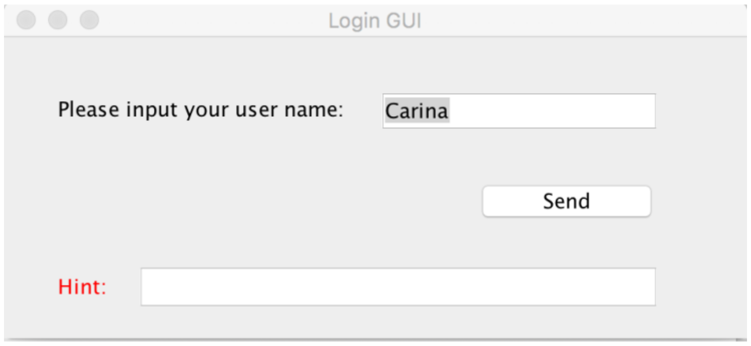


Fig.4: Client Login GUI

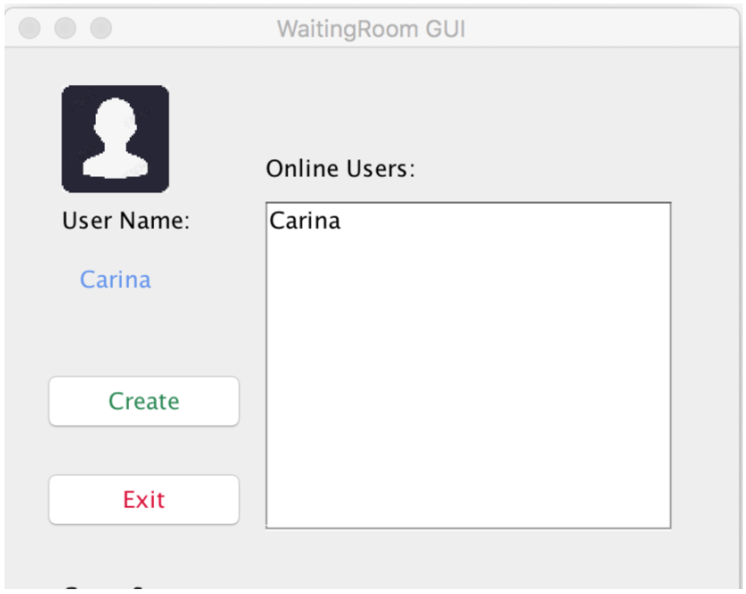


Fig.5: Client Wait Room GUI

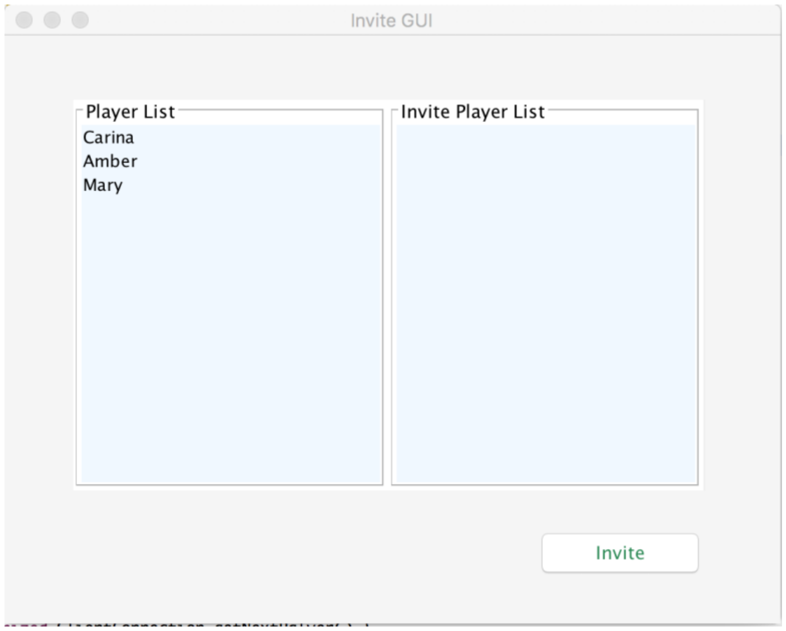
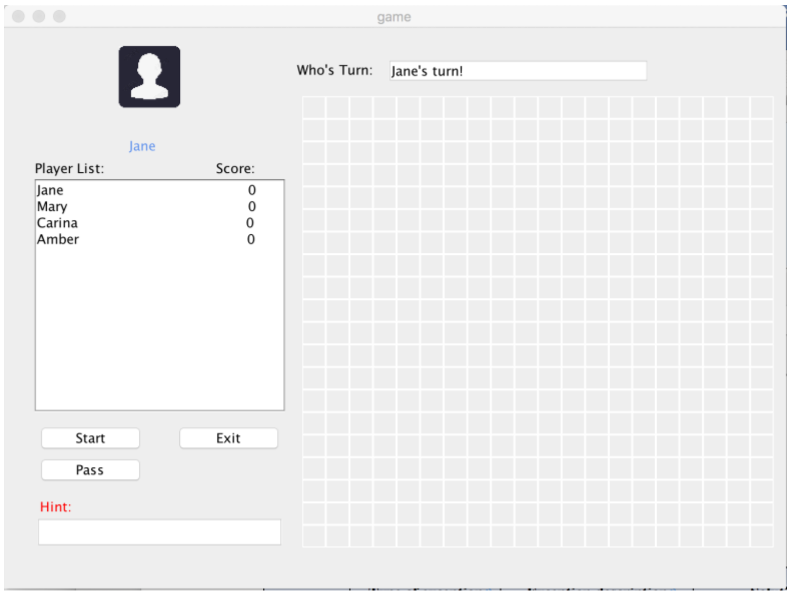


Fig.6: Client Invitation GUI



Player List:	Score:
Jane	0
Mary	0
Carina	0
Amber	0

Fig.7: Client Game GUI

5.2 Server GUI

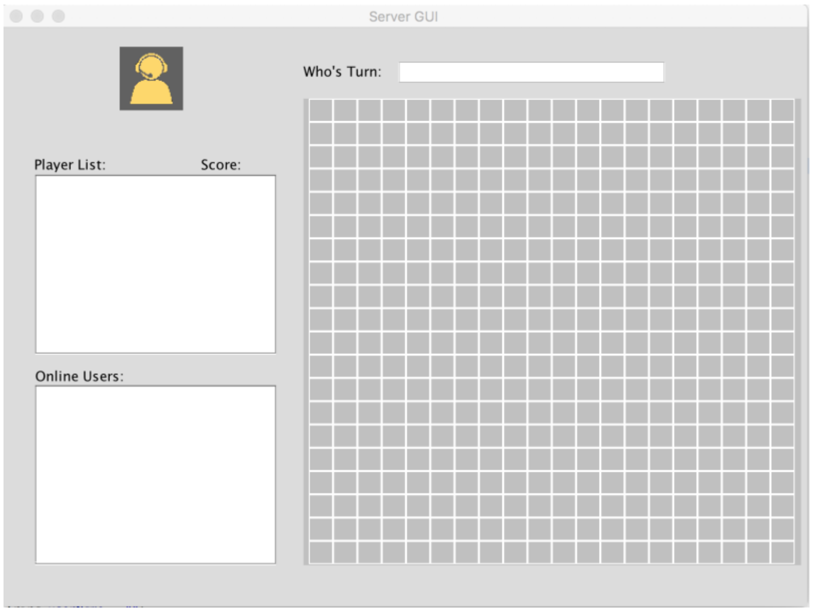


Fig.8: Server Game GUI

6 Exception Handling

There are several exception been considered and handled in this project.

	Type of exception	Exception description	Solution
Client	Connection error	The user can not connect to the server because the server is not open.	Pop up a prompt window to the client and exit the system
		The user cannot connect to the server because an incorrect port number or IP address was entered	Pop up a prompt window to the client and exit the system
		The server is closed after the connection is established between the server and the client	Pop up a prompt to the client that server is closed

	User name input error	The user clicks the button without entering the user name.	System will randomly generate a user name for user
		The user enters the illegal user name	Prompt the user that the username is illegal, and allow him to re-enter the username
		The username entered by the user already exists in the system	Prompt the user that the username already used by others, and allow him to re-enter the username
	game operating error	The user wants to create a new game while a game is in progress	Prompt the user that a game is in progress
		The user wants to start a game but he's not the host of the game	Prompt the user that he has no authority
		The user wants to add a letter without his turn	Prompt the user that he has no authority
		The user wants to highlight a word without his turn	Prompt the user that he has no authority
		The user wants to add a letter to a grid that already filled with letter	Prompt the user that the grid already filled with a letter
		The user wants to add more than one letter on the grid	Prompt the user that the grid can only be filled with one letter

		The user wants to highlight the word, which does not contain the letter he just added	Prompt the user that this is illegal operation
Server	connection error	User disconnected from the server	Refresh online user list
		Open the server repeatedly	Popup prompt open error
	Array Index Out Of Bounds Exceptin	One client logs off, the thread will be removed from Arraylist<ClientConnection> If the index value of an array is less than 0 or greater than the size of the array of Client Connection, the index boundary exception will be thrown.	Popup prompt error
Server and Client	stream execption	Error transferring data stream	Popup prompt error
	socket exception	This exception is caused by a read and write operation after either of the sever and client are disconnected.	Popup prompt error
	IO Exception	Read or write message exception may occur on both the client and server sides	Popup prompt error

7 Analysis

7.1 TCP has great vitality in network communication.

This system chose the connection-oriented transmission protocol (TCP). TCP uses timeout retransmission and piggybacking confirmation mechanism to ensure the reliability of data transmission. However, the verification of the correctness of the data content will inevitably occupy the processing time of the computer and the bandwidth of the network, so the efficiency of TCP transmission is not as high as that of UDP.

7.2 Singleton Pattern

All user threads on the server side are managed by a “ClientManager”. To ensure only one “ClientManager” instance, the system uses the singleton pattern. The singleton pattern provides a globally unique access entry and is easy to control possible conflicts. It is an improvement on the static method. Firstly, it avoids the pollution of the global variable to the system. Secondly, it can have subclasses, and can define virtual methods with polymorphism.

7.3 Client log off abnormally

The server updates the online users list in real time by listening to information sent by client when the client GUI window close. However, if client log off abnormally rather than client click to close the window, the server will not know this client has log off. This problem can be solved by the combination of broadcast and timeout mechanisms. The server continuously broadcasts the message and listens for the response. If the response is not received within the specified time, the connection to the client is disconnected. Unfortunately, this mechanism is not implemented in this project.

7.4 Consistency

A concurrent distributed system should make various processes be able to share access to resources and these processes are scheduled fairly. In this system, the “Controller” can well guarantee the consistency of function implementation with the GUI windows. The “clientManager” can guarantee the consistency of the connection between server and client in the ongoing game.

7.5 Fault Tolerance

A well fault tolerance distributed system should be able to handle exceptions when the data transmission between server and client. This system can handle exceptions and popup prompt to users.

8 Contribution

Name	Student No.	Contribution
Chenyang Lu	951933	● System design ● Server-side code programming ● Server-side code testing
Jing Du	775074	● System design ● Server and Server GUI programming ● Server and Server GUI code testing ● Report writing
Pengcheng Yao	886326	● System design ● Client-side code programming ● Client-side code testing
Yu Dong	928922	● System design ● Server GUI testing ● Report writing