COMP900015 Distributed Systems

Team Project 2: A Distributed Game (Scrabble Game)

Tutorial Time: Thursday 10AM Old Metallurgy - 103
Tutor: Lakshmi Jagathamma Mohan

Yizhou Wang 669026
Yuxin Guo 875698
Kaiyuan Wu 956125
Leewei Kuo 932975

1 Game Introduction

This project aims to design and implement the Scrabble Game System in Java programming language. The game is built on the basis of distributed system, 2-tier client-server architecture and multithreaded mechanism. It inherits typical features of scrabble game and allows multiple players to challenge each other concurrently online. Importantly, the rules of the game system have been adjusted to the version specifically stated in the requirements for this project and its main features are listed below:

- A 20x20 grid is designed as the main GUI of this game, where users on different PCs, at least 2 people will place letters on tiles of this grid to make words in turns.
- Users will take turns to place a character in a tile of the fixed grid mentioned above.
- When tiles that touch each-other make a word, then the person who placed the associated letter will get points equal to the total length (number of tiles) of the word.
- A proper word is judged when all players accept the word through a visible voting GUI.
- The game ends when all users have a turn and cannot find a word/extension to make, basically when all players say pass when their turn comes through a GUI.
- Words are written in English and can only be read from left to right or top to bottom.
- The game allows appropriate controls and information to play as well as to follow it by all players.
- All users are able to see the same view of the game on their machines without differences or errors between them and the game has a clear start and end.
- Users are able to logout from the game any time which would lead to end game as well.
- There is a game membership pool where users come in when they login to the game application and see other potential players.
- Simplifying assumptions are made for membership and game start management such as having at most one game at a time and not allowing people to join games at any time but only at the beginning of a game.
- Any player can initiate a game and invite others who can then accept this.

2 System Architecture

The overall structure of this game system is developed upon client-server architecture.

The server plays a dominant role in listening connection requests, processing message exchanges and conducting business logic of the scrabble game. The functions of the server are designed and implemented in 7 distinct java classes. Processing all the connection

requests and creating and starting the related threads are done in ThreadManager.java. ServerState.java is updated accordingly to store all the connections established between the server and the clients. When it comes to message processing, the ClientConnection.java deals with all the messages received and sent through connections. The game logic is implemented in GameSystem.java and StandardBoard.java, which controls all the information along with the game such as board and turns. Moreover, in terms of GUI, the server has a login interface, which specifies the port number. Since another main function of the server is to monitor the clients' activities, the system also provides a monitor GUI to show the activities. The specific server architecture is shown in Figure 1.

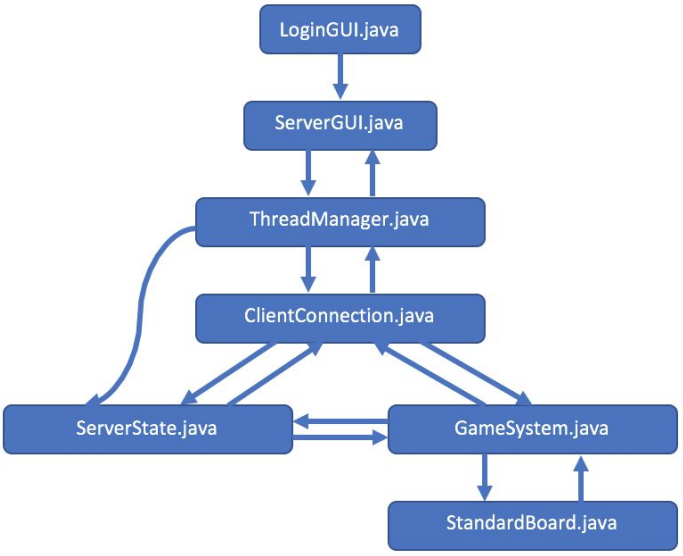

Figure 1. Server Achitecture

The client of scrabble game performs both presentation logic and business logic which includes both GUI and message exchanges. The client is designed as thin client as the main business logic is actually handled in the server. 3 main classes are designed and implemented in the client program of this scrabble game system. The GUI.java is responsible for both the login GUI and login business. ClientGUI.java takes charge of the game lobby GUI, the scrabble game GUI as well as parts of the business logic, especially those for exception handling. The MessageListener.java mainly deals with the exchanged messages and controls parts of the GUI logic. The specific client architecture is shown in Figure 2.

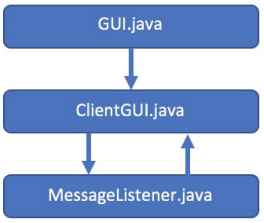

Figure 2. Client Achitecture

3 Communication Protocols

All the connections of this system rely on the construction of the TCP socket. As TCP is connection-oriented and relatively reliable, it is considered as one suitable choice for the system implementation. There are some strengths when using TCP as the communication protocol. Firstly, it can handle the lost packet by ACK and retransmission mechanism. Secondly, it provides flow control. Thirdly, detecting message duplication is also an advantages which helps the system to be kept in consistent status. Last but not least, TCP deals with message ordering which contributes to its reliability as well.

The server also implements a thread-per-connection strategy. It has the advantage of lowering thread management overhead comparing to thread-per-request. This is due to the fact that this particular strategy applied by this system significantly reduces the overhead of thread creation and destruction operations. Therefore, it could process messages more efficiently.

4 Message Formats

The message exchange protocol designed for the system is implemented in JSON format. Both JSONObject and JSONArray are used for message encapsulation and de-encapsulation. This format provides an efficient way for message exchange and because of which, a message must follow the logic to be parsed into two components, either command or the true value that transmit to the server, for performing specific functions. The details of the designed message exchange protocol and its corresponding format are shown below.

Table 1. Message exchange protocol and Message Formats

Key/Command	Value/Parameter	Action Afterwards	Message Format
“username”	client’s name	Put username in connected client’s list if the name is vaild.	JSONObject
“invite”	client who has been invited	Send invite request to client who has been invited.	JSONObject
“inviteAll”	all connected client	Send invite request to all connected clients.	JSONObject
“accept_invite”	the name of client who send this message	Put him/her in player’s list.	JSONObject

“connected or not”	none	Print the connection status of that client.	JSONObject
“exit”	none	Print The client is exit on sever.	JSONObject
“startgame”	none	Start the game and send information to let all other players open their GameUI.	JSONObject
“put”	the character he/she put	Store this character and its location in the server; Send this character and its location to other players.	JSONObject
“pass”	none	Send all payers who is next turn.	JSONObject
“passWithoutPut”	none	Send all payers who is next turn; Store this information. If all players send this key in the same turn, end the game and send all players the game result.	JSONObject
“vote”	the location of word’s first letter and last letter.	Send message to let all player’s client open their VoteUI.	JSONObject
“voteResult”	“approve” or “disapprove”	Store the vote’s information; Send vote result if all player finish their vote; Sned who is next trun.	JSONObject
“endGame”	none	End the game and send all players the game result.	JSONObject JSONArray
“login”	client to server :username server to client: “true” or “false”	Client send username to server, and server check whether the username is already exist or not. If the username is valid, server will send client “true”.	JSONObject
“update_userList”	newest connected client’s list	Update the newest client’s list.	JSONObject JSONArray
“invite_request”	client who has been invited	Send this request information to server.	JSONObject
“player_in_game_List”	newest player’s list	Update the newest player’s list.	JSONObject JSONArray
“openGameUI”	none	Let client open his/her GameUI and put all player’s name with their initial	JSONObject

		mark on GameUI.	
“board”	the character he/she put	DisPlay this character based on its location.	JSONObject
“pass”	who pass vote	Tell the server who pass.	JSONObject
“whoPlay”	who play	Weak the operation interface if this is his/her turn.	JSONObject
“passWithoutPut”	who pass without put	Tell the server who pass wothout put.	JSONObject
“end”	game result	Display the game result.	JSONObject JSONArray
“vote”	the location of word’s first letter and last letter.	Send this message to the server.	JSONObject
“openVoteUI”	none	Open the vote UI; Show the voted word;	JSONObject
“voteResult”	“approve” or “disapprove”	Show the final vote result.	JSONObject
“acceptInvite”	none	Let this client have right to join the game. But if the game is begin, he/she can not join.	JSONObject

5 Design Diagrams

5.1 Class Diagrams

5.1.1 Server UML

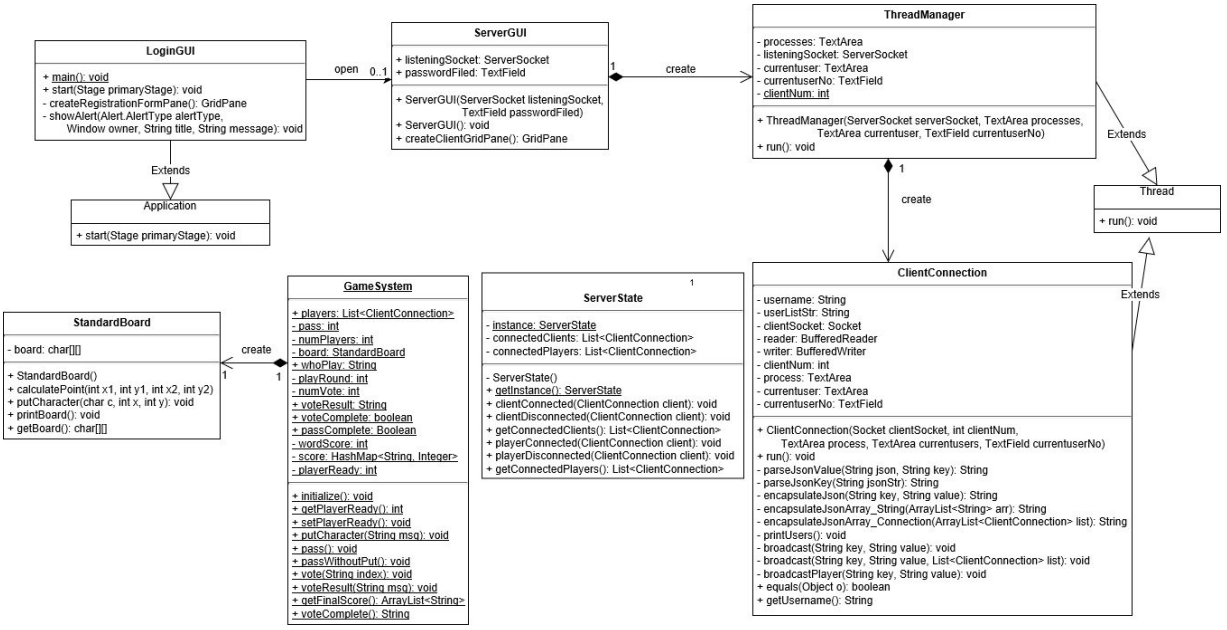

Figure 3. Server UML

In the server UML:

- There is a LoginUI used for server user to login the server system.
- There is a ServerUI used for server user to monitor the messages exchanged between server and clients as well as the statues of the connections.
- There is one main thread listening for incoming connection requests and create one thread per client connection, which is handled by ThreadManager.
- Each thread is responsible for listening for incoming client messages, processing and sending back the results. This is designed and implemented in ClientConnection.
- There is a singleton holding all the state information of the connections, namely server statues, which is the Serverstate in the figure.
- The GameSystem stores all the information about the game status such as players, turns, board, passing, voting and scores.

5.1.2 Client UML

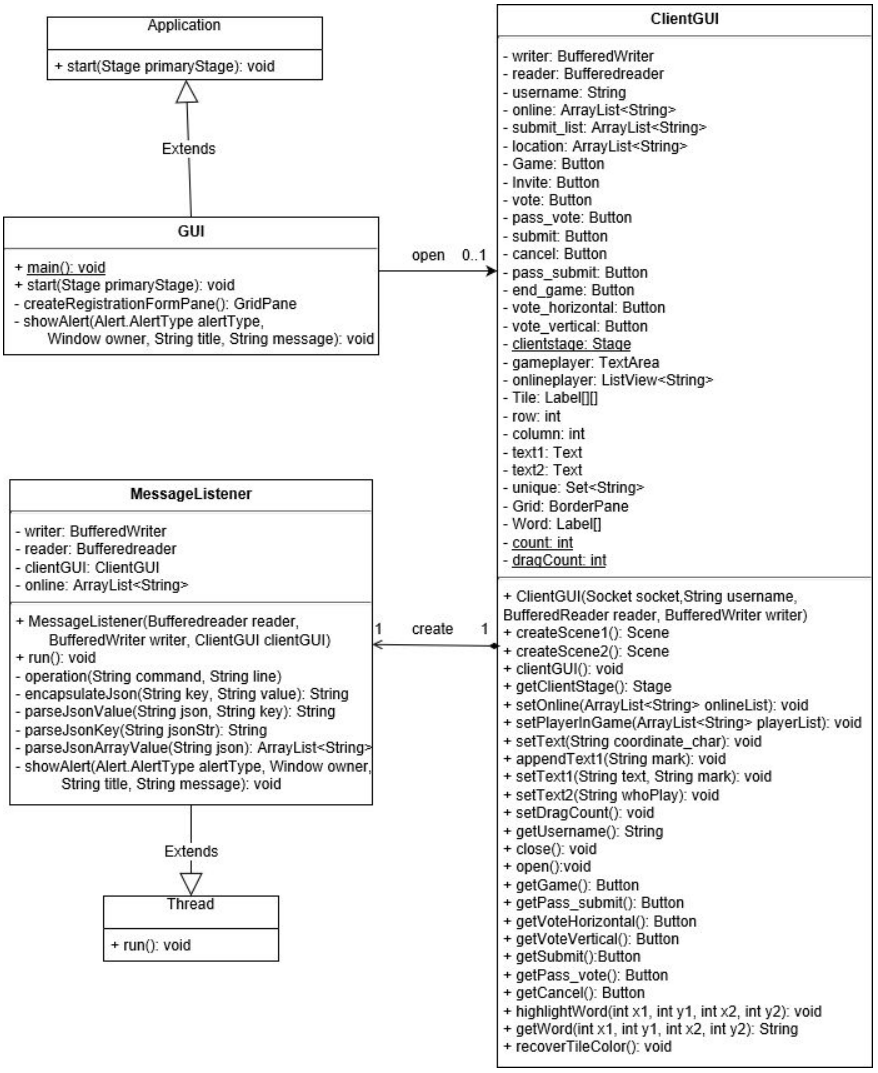

Figure 4. Client UML

In the Client UML:

- GUI is used for both the login GUI and login business. It also handles socket exceptions and other input exceptions.
- ClientGUI is used for user to present the game lobby GUI and the game GUI. It sends messages to server, especially through alters and main panel.
- MessageListener is used for user to receive messages from server and performs presentation control to the ClientGUI.

5.2 Interaction Diagrams

The interaction diagram is shown below.

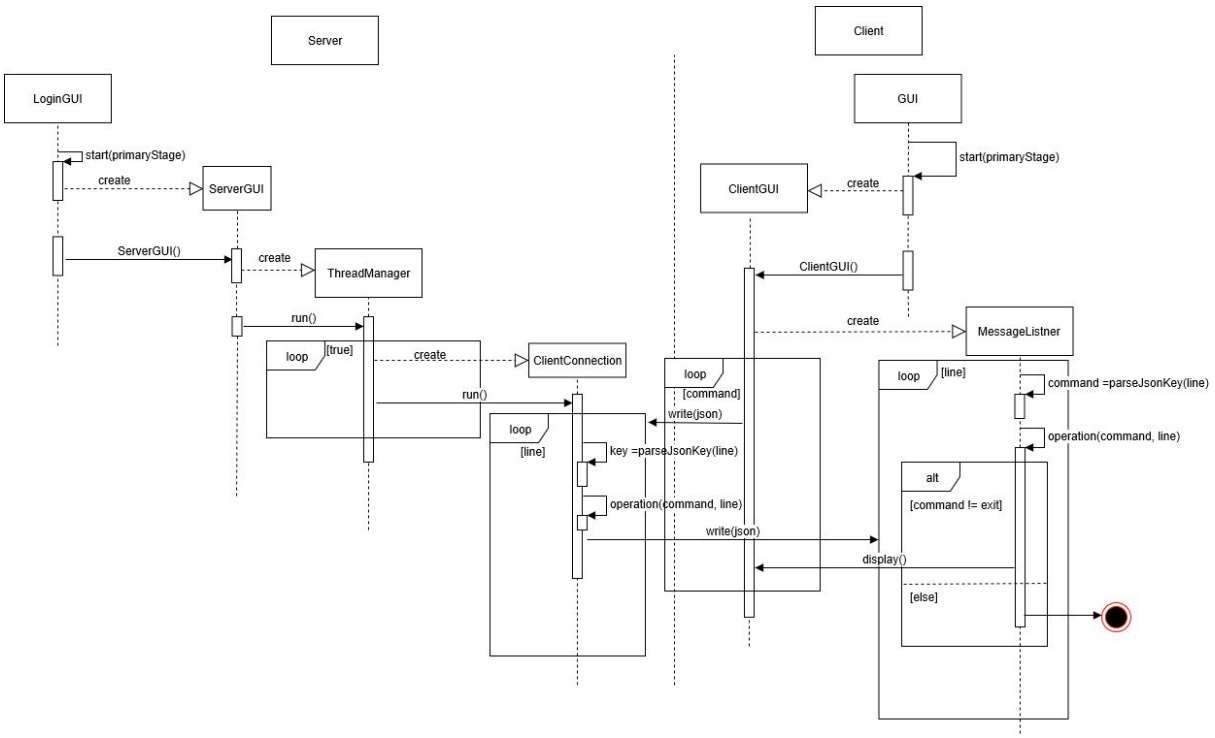

Figure 5. System Sequence Diagram

6 Other Implementation Details

6.1 Failure Handling

1. Preventing any illegal port number when creating the server.
2. Invalid port number and IP address in the client side will not be allowed.
3. When a person wants to log in to the game client, the system will check whether the username exists or not. The system will reject him/her if the username already exists.
4. If the game begins, no one can join the game until the game ends.
5. If the player is not in his/her turns, the operation interface will be invisible. Operation interface includes “Vote”, “Pass Vote”, “VoteVertical”,“Vote Horizontal”, “Pass Submit”, “Submit”. “End Game” etc.
6. Player can only press button under the control of game. Based on the rule of game, the button will freeze if the system do not allows he/she to press it or awaken if the system allows he/she to press it.
7. If one of the player log out the game or cannot connect to the server, the game will end and show the current game scores. The game score consists of all player’s name and their corresponding marks from highest to lowest.

8. If the server crashes or ends, client will return a message to inform users that the server is not online or working.

6.2 GUI Implementation

The Client GUI is a significant part of the system design. Importantly, the drag action is one of the core functions that provided by the game system. The system firstly provides the login interface that requires the port connection information. It is then followed by a game lobby that simultaneously reveals the online and in-game player. In fact, the lobby is an abstract membership pool, which allows players to invite other players to join the game. Afterwards when the game starts, the scrabble game GUI shows up which contains a broad panel of larger size, which is 20*20, and responsible for the players to play and enjoy the game. In the game, a voting mechanism is used to judge if the tile can make up a valid word put down by a client in each turn. The score is calculated based on the length of the word and will be updated as long as the word is approved by all the players. Finally, when the game ends, the players will be taken back to the lobby.

Additionally, to be more specific, all the GUI components are implemented under the support of javafx.

1. For client UI, the list of client who are not in the game and the list of client who are in the game are all real-time display.
2. For game UI, the list of player and their real-time mark are all real-time display.
3. If a player was invited by another, he/she can accept or unaccept. If he/she accept, he/she have the right to join the game. But if the game is begin, he/she can not join.
4. If the player put a character successful, this character will not only show on every player's game UI but also highlighted.
5. Each player can cancel the character his/her put before submit.
6. The game UI will highlight the character someone just put in the last turn and the word when they want to vote.
7. If all player pass without put any character in one turn or one of the player leave the game, the game will ends and show the game result. The game result is all player's name and their mark from highest to lowest.

6.3 Screenshots

This part illustrates some important screenshots from both GUI and exception handling.

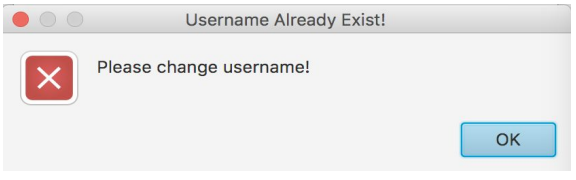

Figure 6. Duplicate Username Exception Handling

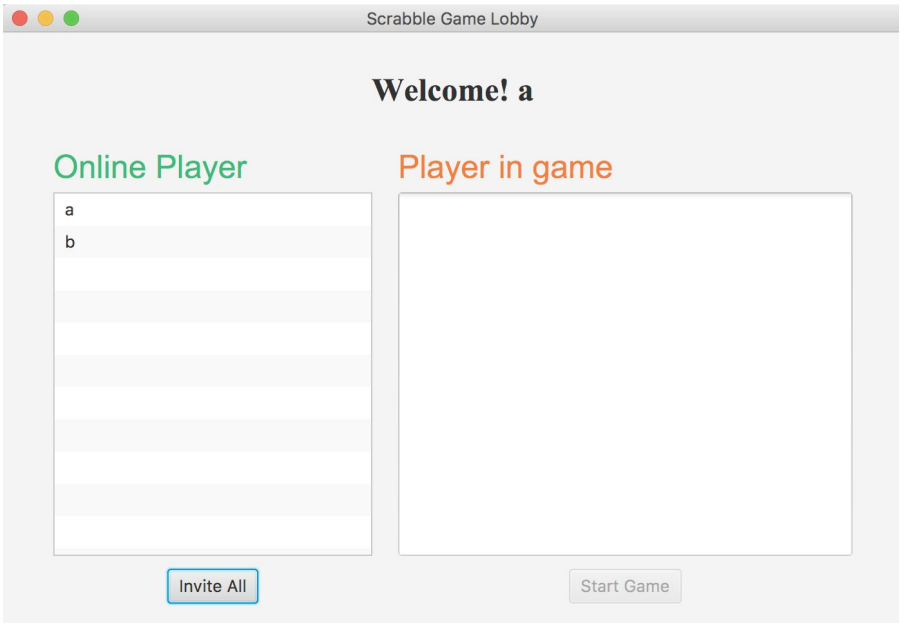

Figure 7. Game Lobby

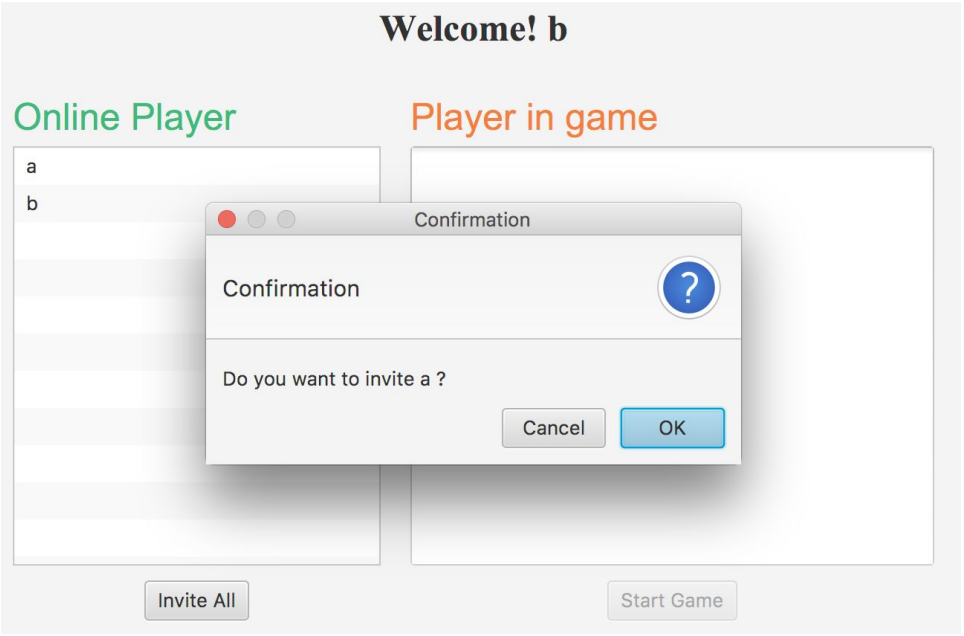

Figure 8. Invitation

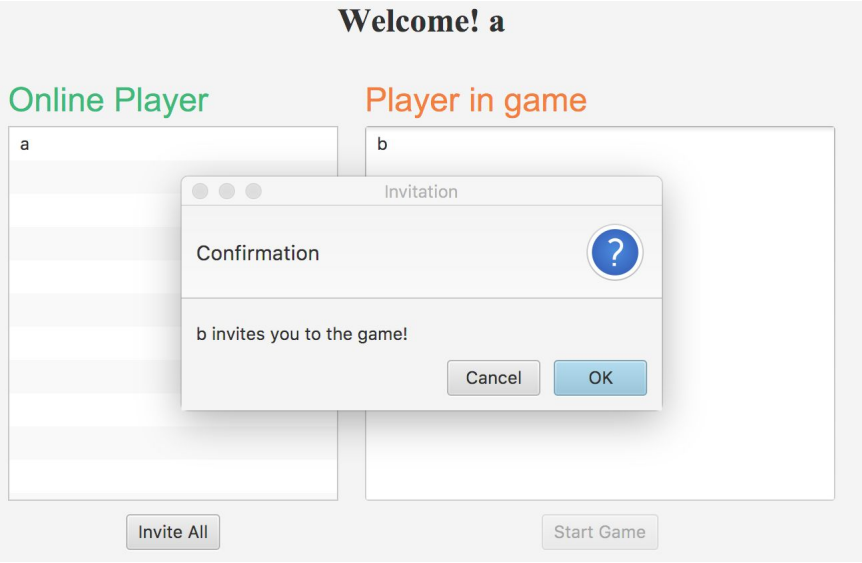

Figure 9. Confirmation

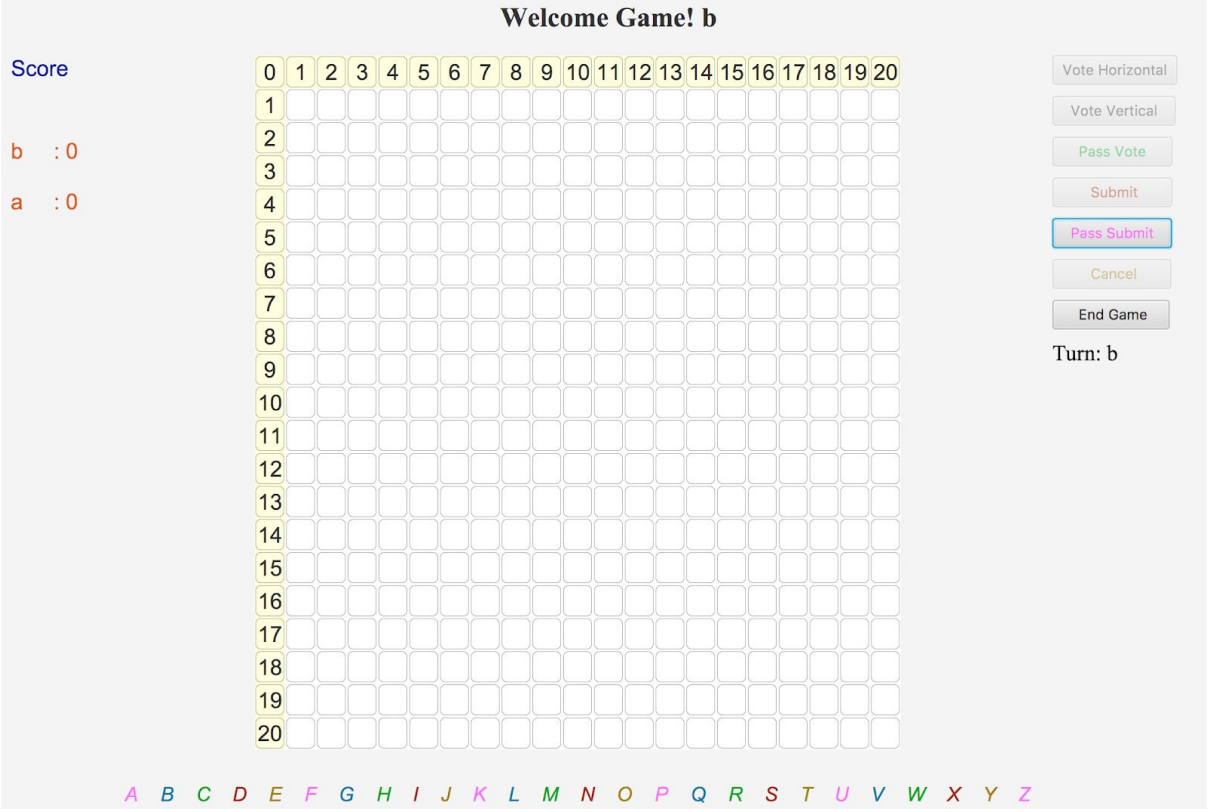

Figure 10. Scrabble Game GUI

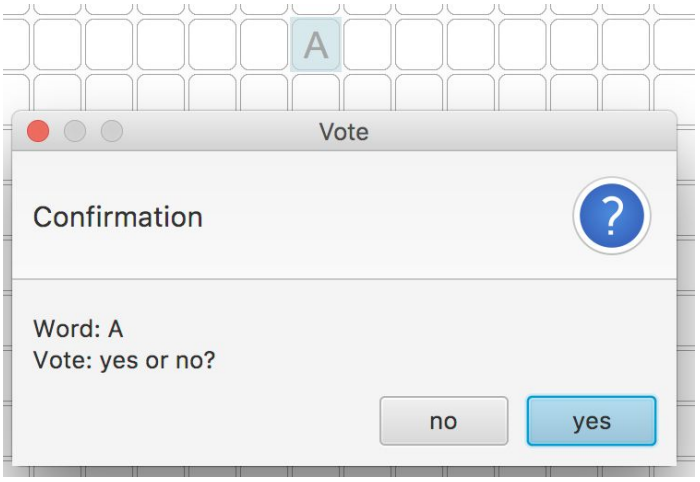

Figure 11. Vote GUI

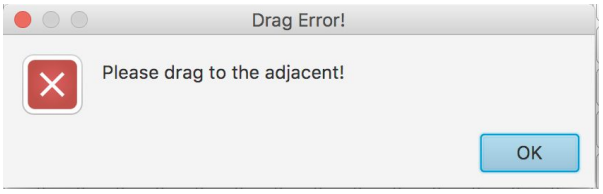

Figure 12. Invalid Drag Action

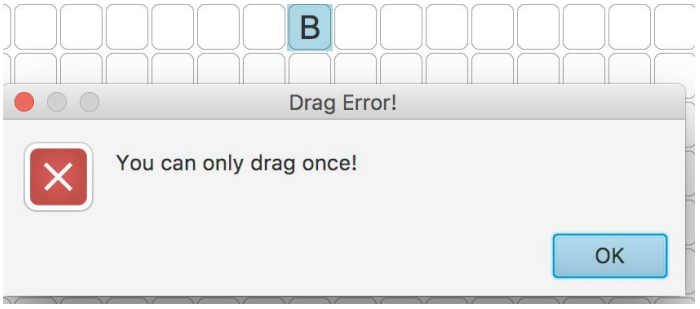

Figure 13. No more times drag action

Figure 14. Game Over Notification

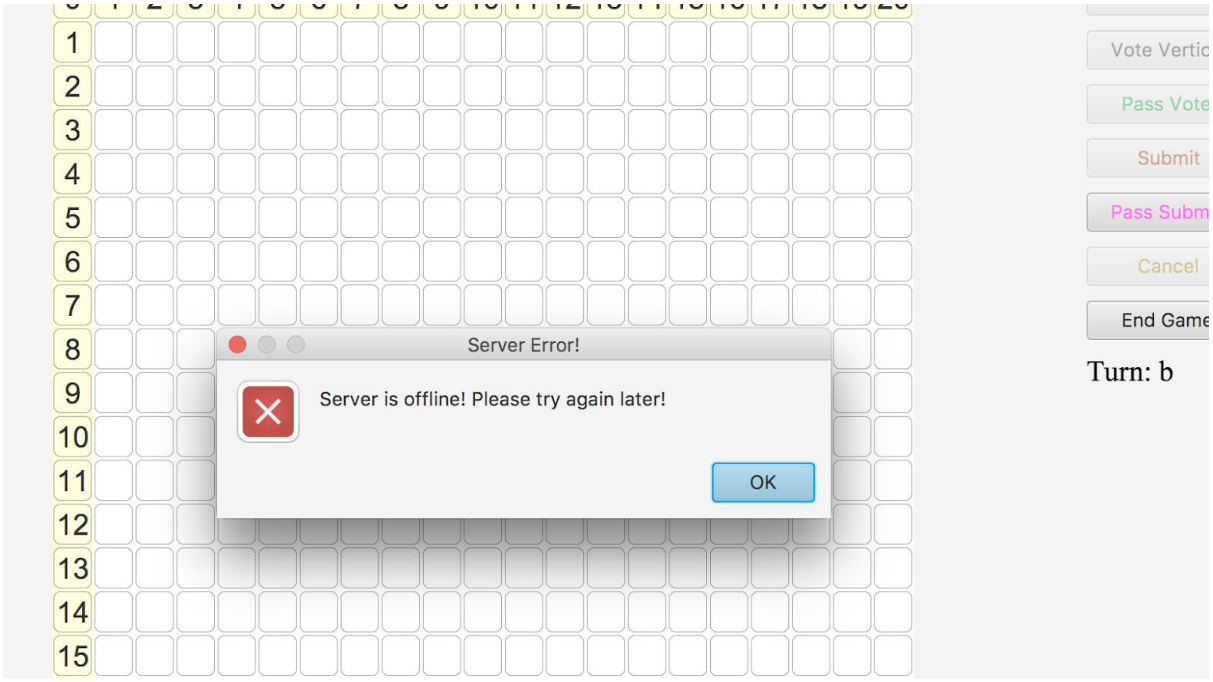

Figure 15. Server Status Detection

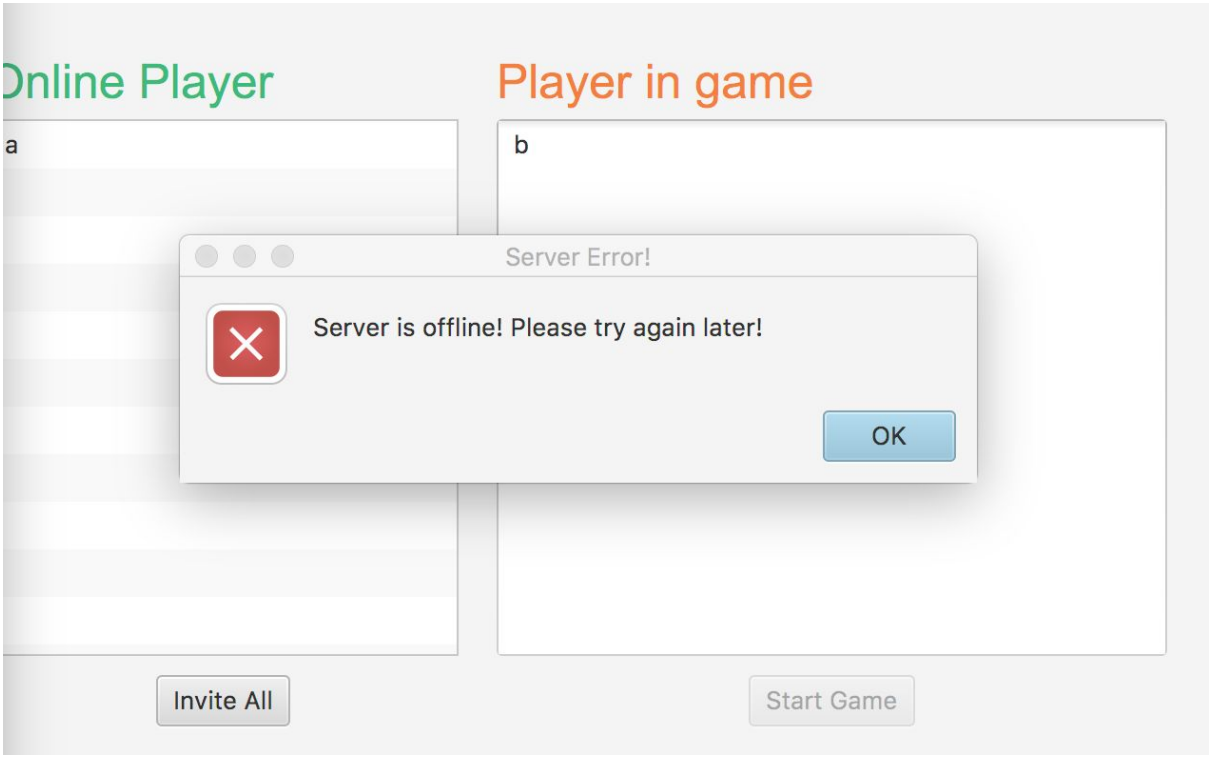

Figure 16. Server Offline Dectection

7 Conclusion

In this project, the distributed scrabble game system is designed and implemented. The system uses 2-tier client server architecture and uses TCP as the communication protocol. JSON is used as the message formats for message exchange. Class diagram and interaction diagram are provided to

illustrate the design and implementation of both front end and back end of the whole program. Apart from that, exception handling and GUI are stated at last to additionally demonstrate other design issues.

8 Member Contribution

Name	Responsibility	Percentage
Yizhou Wang	Mainly focus on the GUI construction	25%
Yuxin Guo	Mainly focus on the communication implementation	25%
Kaiyuan Wu	Mainly focus on the GUI construction	25%
Leewei Kuo	Mainly focus on the game design and backend	25%