Assignment 2 – Report: A Distributed Game

A Distributed Scrabble Game Created by Relax

Team Name: Relax

1. Introduction

This report is going to introduce a distributed system named scrabble game, which can support concurrent operations and communication between clients and the server based on TCP protocol. Furthermore, the architecture and detailed features of interactions will be demonstrated. At the same time, the UML charts of Client and Server, and the meaning of different Classes are introduced. Besides, this report will indicate different functions and error handling by screen shots. Finally, contributions of different team members will be demonstrated.

2. Description

2. 1 Requirement

According to the problem, there are several requirements for this program:

- Scrabble game
  It should have a complete game logic that can be played with precise rules
- Client-server
  The system should present a client-server architecture that each client can require this server for game by socket (TCP or UDP).
- Dealing with concurrency
  To be compatible with concurrent visits and operations, the server of this system should be designed as a multi-threaded system.
- Error handling.
  The errors which may happen during communications should be well managed.
- Structuring your application and handling the system state
  The structure of the application should be designed logically and meet game demand.
- GUI
  Client should be offered a game graphic interface to play this game alone or with other players
- JAVA Application
  The outcome of this assignment should be a JAVA application.

2.2 Game Architecture

The architecture of this system is Client-Server architecture. Shown as Fig 1.

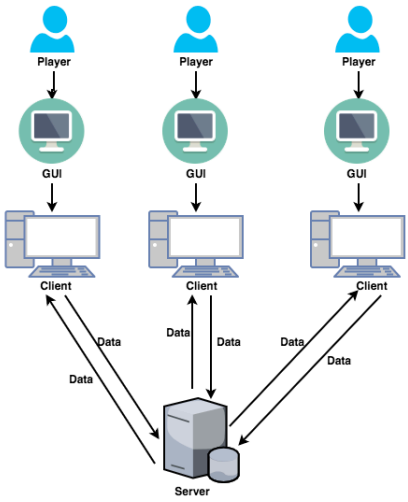


Figure 1: whole architecture of Client-Server System

3 Protocols

The communications between Client and Server are using socket. There are two kinds of sockets, TCP and UDP. Because the reliability of system is really important, the socket in this system is TCP. TCP is connection-oriented communication.

The message exchange protocol is Json. When a user register, the username will be sent to Server by Json. If there is no same username existed in server, server will approve this user's register. The first player clicks QuickGame, Client will send a command "Initialize a game" to Server, then Server returns a player list to this first player. This player can invite others through this player list. When this first player finished choose, the chosen player list is sent to Server, Server will broadcast this list to others. When others accept invitations, game start. This process is shown as Fig 2.

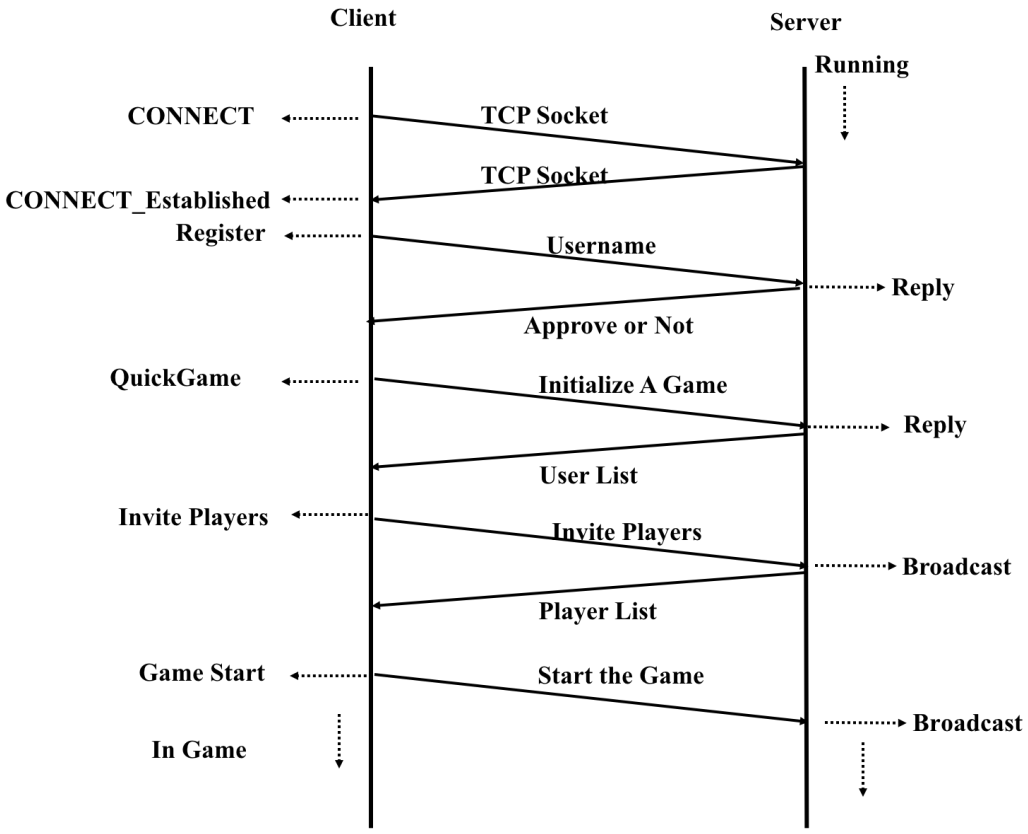


Figure 2: Pre-Game Data Exchange

When the game start, Server will send a command “This is your turn” to one user C, and broadcast who is turn now to other users. Then User C moves one letter, Client sends the coordinates of this letter to Server, then Server broadcast all coordinate data to all players. If User C click pass, Client sends command “pass” to Server rather than coordinates. Server will broadcast this user pass to others. If one player highlights one word, then Client sends the coordinates of this word to Server. The broadcast function of Server will make other users to vote this word. If all users pass this word, Server will broadcast the results of this game. This process is shown as Fig 3.

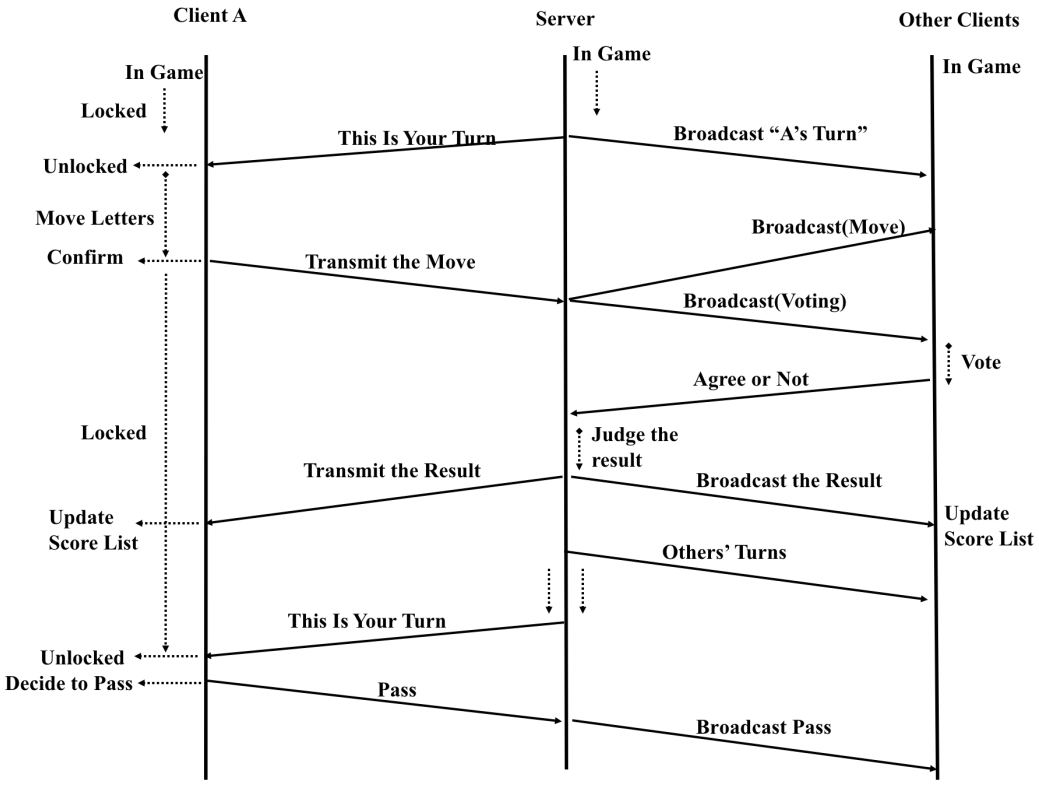


Figure 3: In-Game Data Exchange

4 Program design

4.1 Design diagrams

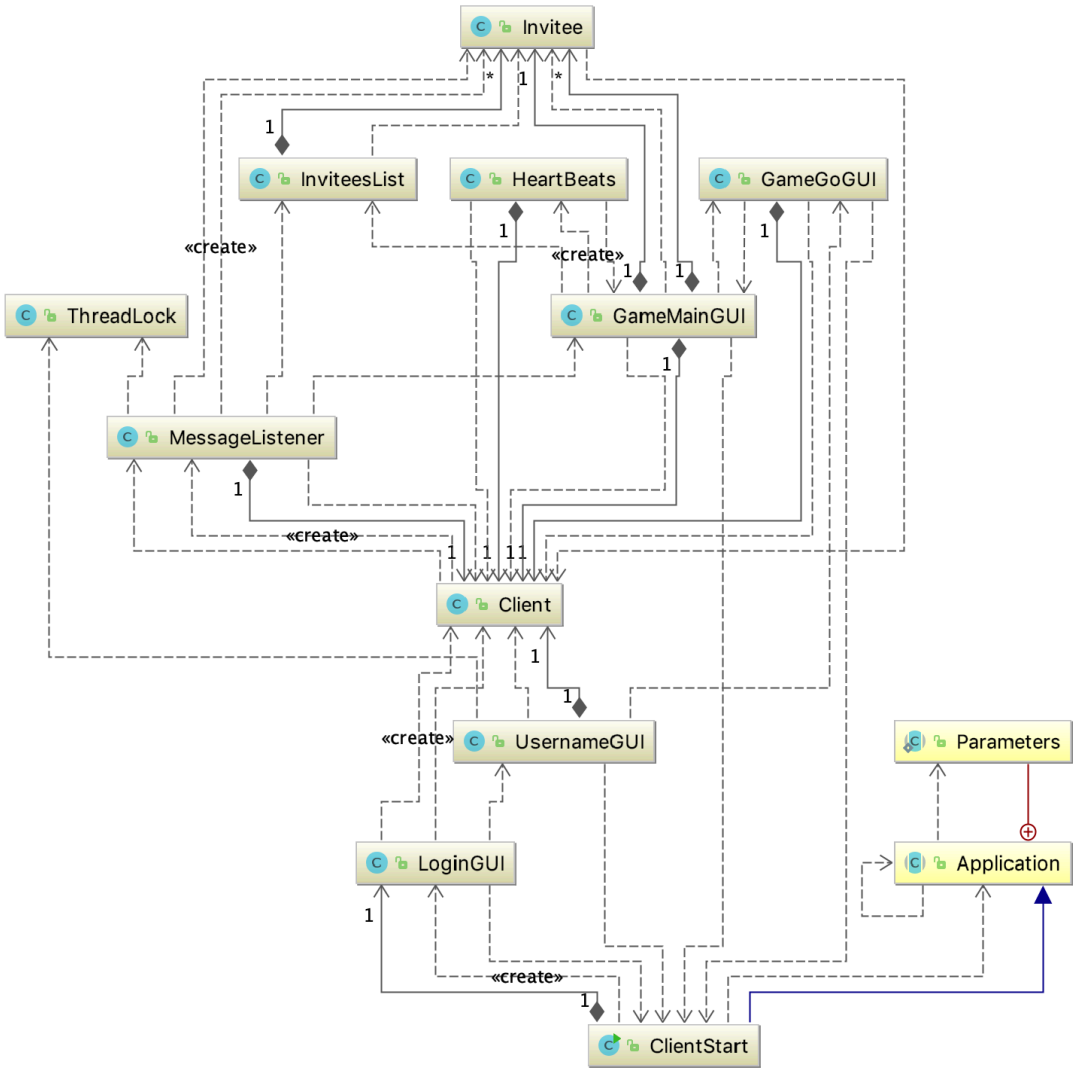


Figure 4: Client UML Class Diagram

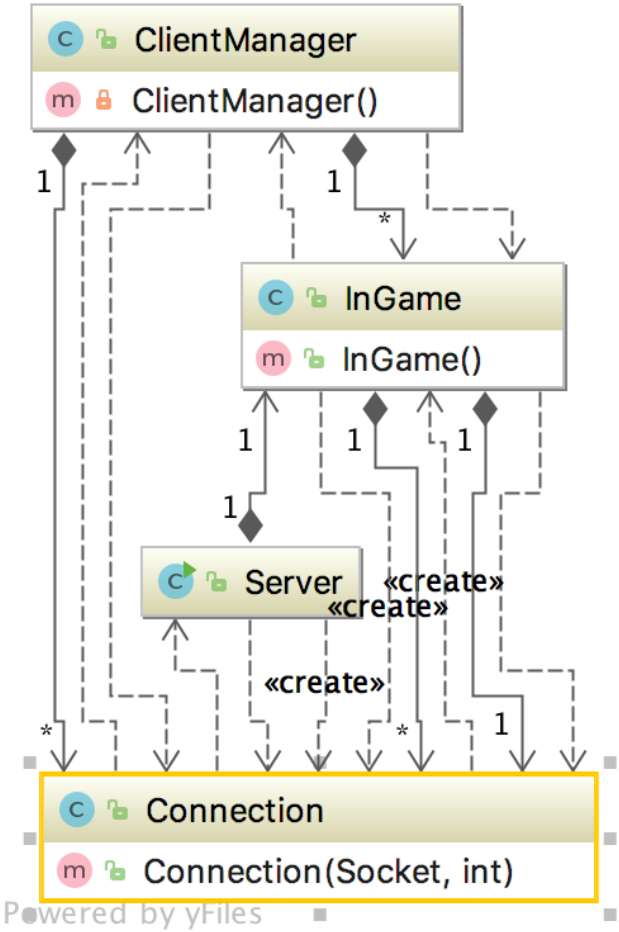


Figure 5: Server UML Class Diagram

4.2Document Explanation

4.2.1 Client Classes Explanation

There are 11 classes in the server part, where GUI classes include *ClientStart*, *LoginGUI*, *UsernameGUI*, *GameGoGUI*, and *GameMainGUI*. We start from *ClientStart* and enter in the login interface (*LoginGUI*). After we finished inputing the address and port number to establish a connection to the server, we will enter the user registration interface (*UsernameGUI*). And then, *GameGOGUI* is the interface of preparation of starting a game. The class *GameMainGUI* implements the main interface of the game process.

The class *Invitee* and *InviteesList* implement for the interaction with the *GameMainGUI*, which are in charge of updating the invitee list available for invitation for the host. The class Invitee implements the private attributes and methods for each invitee, where an invitee is defined as an instance of the class Invitee. The class *InviteesList* manages all the instances of the class Invitee, which can be deemed as an invitee manager. It has a list (invitees) to store all the instances of the class Invitee.

The class Client implements for defining the methods of sending requests to the server. We also implement two threads that are the class *MessageListener* and the class *HeartBeats*.

The class *MessageListener* is responsible for listening and receiving messages from the server. Furthermore, the class *ThreadLock* is implemented for interactive connection between the class *UsernameGUI* and the class *MessageListener*. More specifically, it's used for locking of jumping into the next interface until the server approved of the unique username.

And the other one *HeartBeats* is in charge of continuously sending heartbeat packages to the server to check whether it is still connected with the remote server.

4.2.2 Server Classes Explanation

There are four classes in the server part, *Server, Connection, ClientManager,* and *InGame*.

The class *Server* is the class with the main function. This class implements server running. Once the sever is running, it will listen and react to all the connections.

The class *Connection* is the client-server connection. Once a client connects to the server, a new *Connection* instance is created. Each connection will have an allocated number (*ClientNum*) as the ID. After setting the username, *Connection* will save this name (*ClientName*). The class *Conncetion* listens to its corresponding client and reacts to the command sent from the client.

The class *ClientManager* manages all the clients. It has a list (*connectedClients*) to store all the clients (*Connection*) that connected to the server, and another list (*inGameClients*) to store all the players (*Connection*) that have joined the game. *ClientManager* have several client-server interaction functions for different aims, including broadcasting JSON messages to all the clients (*broadcastToAllClients()*), broadcasting JSON messages to all the players (*broadcastToInGameClients()*) and clearing the list of players (*clearInGameList()*).

InGame is the class for the game status and the game logic. Every time a client initializes a game, a new *InGame* instance will be created. This class include some game logic functions, including deciding whose turn to move, judging whether the word submitted by a player is approved by all other players and judging whether all players pass their turn. *InGame* also has other operation functions of print all the players and their scores (*scoreList()*)and add some points to one player (*UpdatePlayerScore()*).

5 Demonstration and Analysis

5.1 Function

- **Login Interface**
  Clients execute the application and then will be shown a login interface to enter the server address and port number. (Demonstration can be found in Appendix A-1)
- **User Registration**
  Once clients have login they can create their own username. (Demonstration can be found in Appendix A-2)
- **Preparation Interface**
  Once clients have their username, they can choose to start or wait to watch a game. (Demonstration can be found in Appendix A-3)
- **Game Holder**
  The first client who click the "QUICK GAME" button will be the game holder, which means only he/she can invite other players and start the game. (Demonstration can be found in Appendix A-4)
- **Invitation**
  Players can see all other players in the left-top player list, and the game holder can invite them to join the game. Players who are invited will receive an invitation message and they can choose to accept or decline. When they accept, their name will be added into the score list. (Demonstration can be found in Appendix A-5)
- **Game Start**
  When the game holder clicks "PLAY" button, the game will start, all people in score list will join the game and other people can still watch the game or quit. (Demonstration can be found in Appendix A-6)
- **Move A Letter**
  Players can move the letter blocks into the grid turn by turn. When a player has moved a letter block, this player can click "UNDO" or "CONFIRM" button. When the player clicks "CONFIRM", this player can choose "SEND" or "VOTING". If the player clicks "SEND", Client sends the coordinates of this move to Server. (Demonstration can be found in Appendix A-8)
- **Vote A Letter**
  When the player clicks "VOTING" button, the words of the row and column of this letter will present on the right screen. This player has to choose one word to highlight. When this player chooses one word, Client sends the coordinates of this words to Server. Then other player can vote for this word. If all players choose "PASS" or the grid is filled or any player in game quits, the game will end, and the result and winner will be printed. (Demonstration can be found in Appendix A-9)
- **One Player End Game**
  If one player clicks "END GAME" during playing, game will end and other players' screen will show "Someone Quit Game. Game Over!". (Demonstration can be found in Appendix A-10)
- **All Plays Pass**
  If all players click "PASS" to pass their turn, the game will end. All players' screen will show "All players have passed the turn! Game Over!". (Demonstration can be

found in Appendix A-11)

5.2 Error handling

- **Port number or address entering error**
  In the login interface, if the client enters wrong port number or address, system will pop up a prompt. (Demonstration can be found in Appendix B-1)
- **User name entering error**
  In the username register interface, if the client enters none or existing username, system will decline this registration. (Demonstration can be found in Appendix B-2)
- **Frequent and invalid operations handling**
  In a scrabble game, when it is not the client's turn, he/she cannot click and button and do any operation except "END GAME". (Demonstration can be found in Appendix B-3)
- **Terminate server error handling**
  During the scrabble game, if the server is terminated for some reasons, the players will be noticed and cannot continue play anymore. (Demonstration can be found in Appendix B-4)
- **Terminate client error handling**
  During the scrabble game, if one client is terminated, there are two cases. If the client is currently playing the game, the game will be ended immediately and all other players will receive an end-game message. If the client is watching the game (not playing), no message will be sent to players and the game will carry on. (Demonstration can be found in Appendix B-5)

6 Contribution

6.1 Stage One

Site Huang, Zhentao Zhang and Qianying Dong wrote codes of Client for at least 100 hours.
Dinghao Yong and Zhaoqian Dai programmed Classes of Server for at least 80 hours.

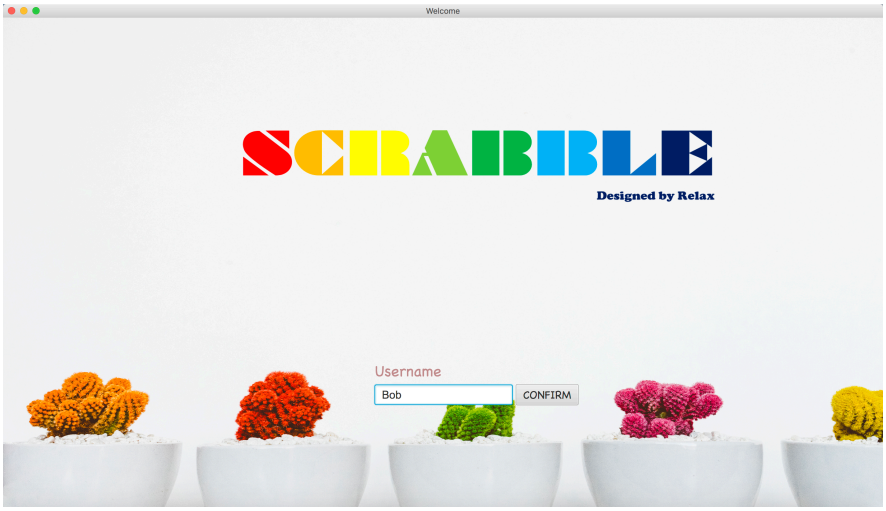
6.2 Stage Two

All team members link the codes of Client and Server and debug this system about 70 hours.

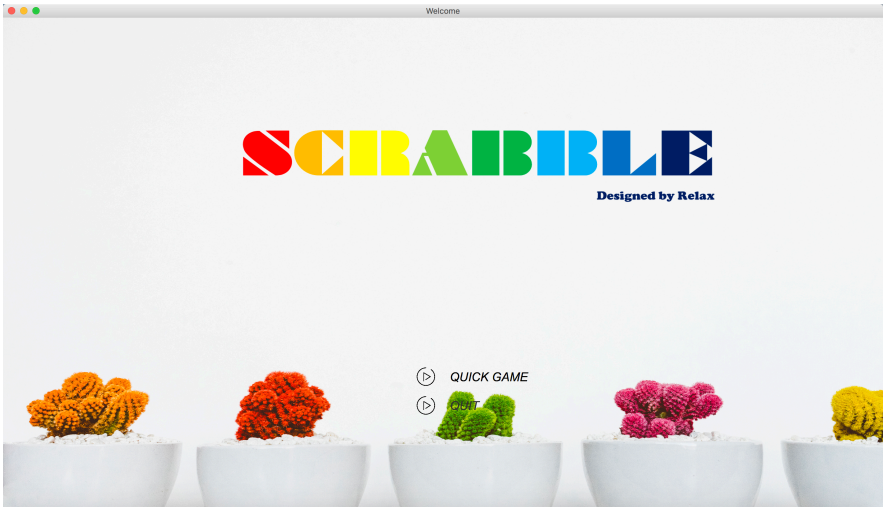
Appendix A Function Demonstration

1. Login Interface (ip address + port number):

2. User Registration:

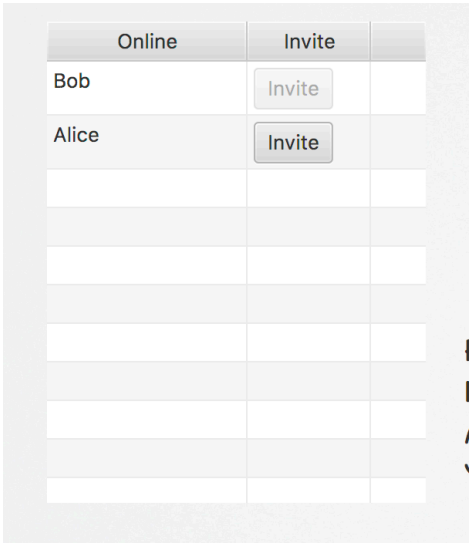


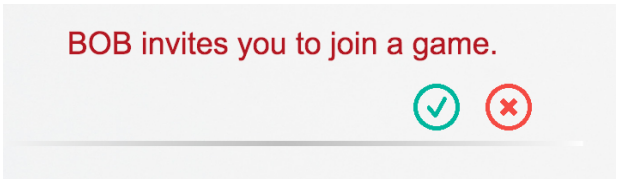
3. Preparation Interface:



4. Game Holder:

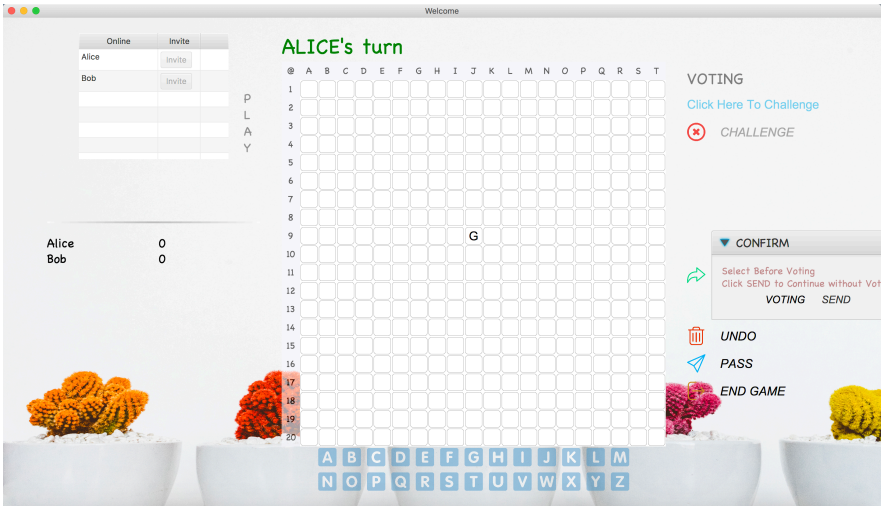
5. Invitation and Answer the Invitation:



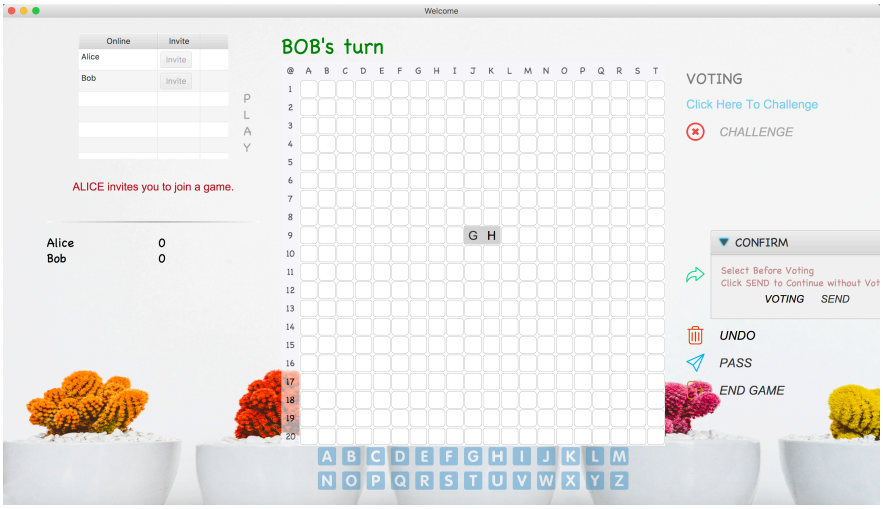


6. Game Start:

7. Move a letter:



8. Select a word and Ask for a vote:



9. Vote for the word

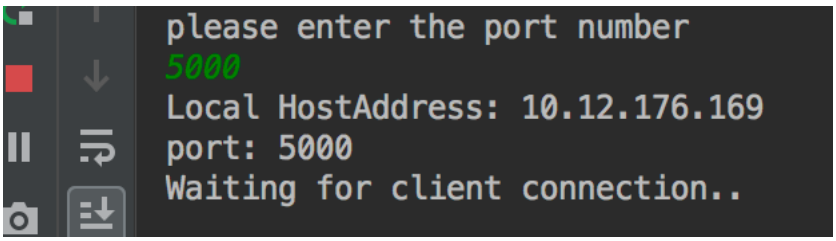
10. Someone decides to end the game:

11. All players have passed their turns:

Appendix B Error Handling Demonstration

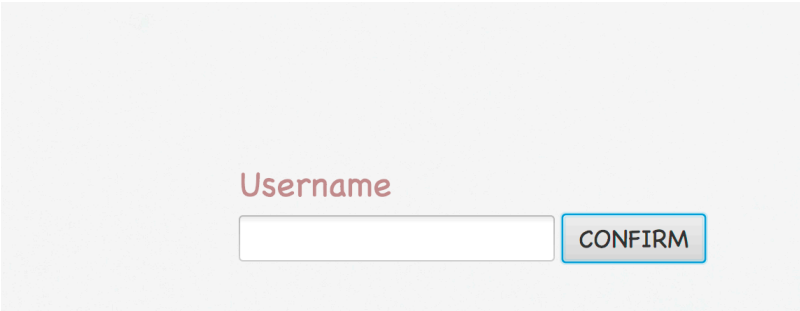
1. Port number or address entering error

   Server information:



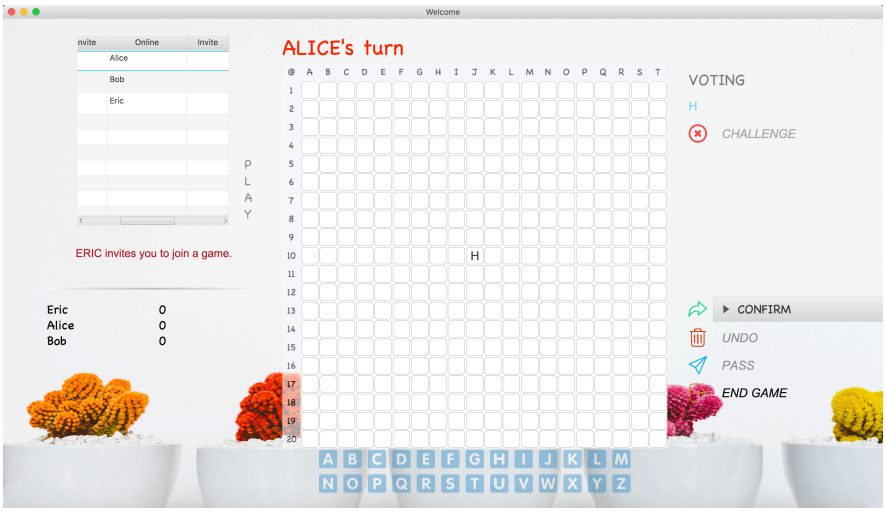
   Wrong entering information and prompt:

2. User name entering error
   Empty username will get no feedback:

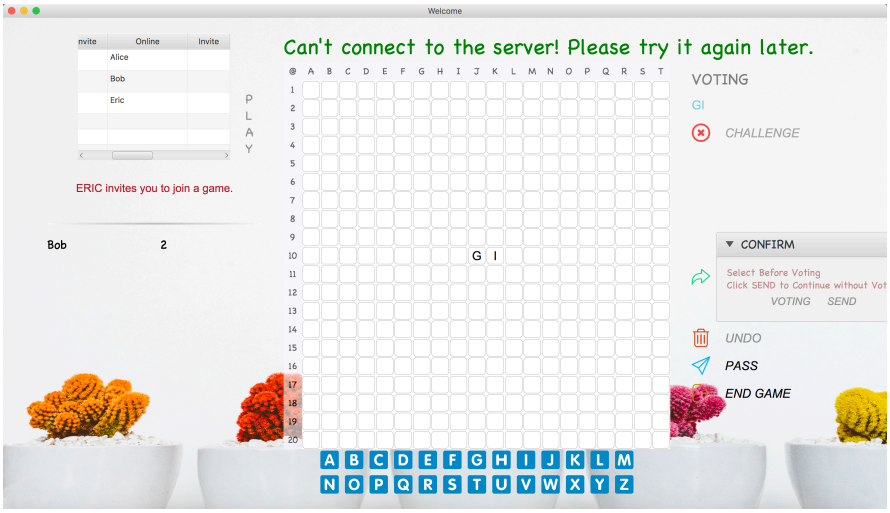


   Username already exists:

3. Frequent and invalid operations handling
   When this is Alice's turn, others cannot do anything (the grey buttons mean that they cannot reply any operation)



4. Server is terminated
   Players will find a prompt in their interface, and all operation will be locked except “END GAME”



5. One of the players is disconnected: