CPMPSCI 711 A2 Report

Xuheng Duan
Computer Science
University Of Auckland
Auckland, New Zealand
xdua752@aucklanduni.ac.nz

This assignment design and implement a multithreaded game server suitable for two-player games (such as Backgammon, Checkers, Chess, Go, etc.). The server helps to pair up players and to coordinate the exchange of game moves. It does not know anything about the underlying game itself.

Keywords—game, multithreaded, socket, C#

I. Introduction

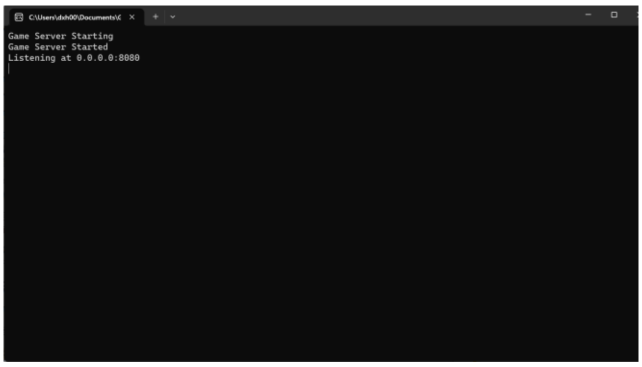
For this assignment, I use C# language, socket connection method, and multi-thread to write the backend. The front end uses basic html, css, JavaScript. For this set of front and back ends, it allows users to register, match, play games, and log out through the buttons in the stand-alone browser. When the user enters the matching operation, the front end will automatically detect whether another player has been matched. When entering the game stage, whenever the user operates a chess piece, the front-end will convert the layout of the chessboard into data and send it to the back-end for recording. At the same time, another player will also receive the operation of the opponent.

II. Ease of Use

A. Backend

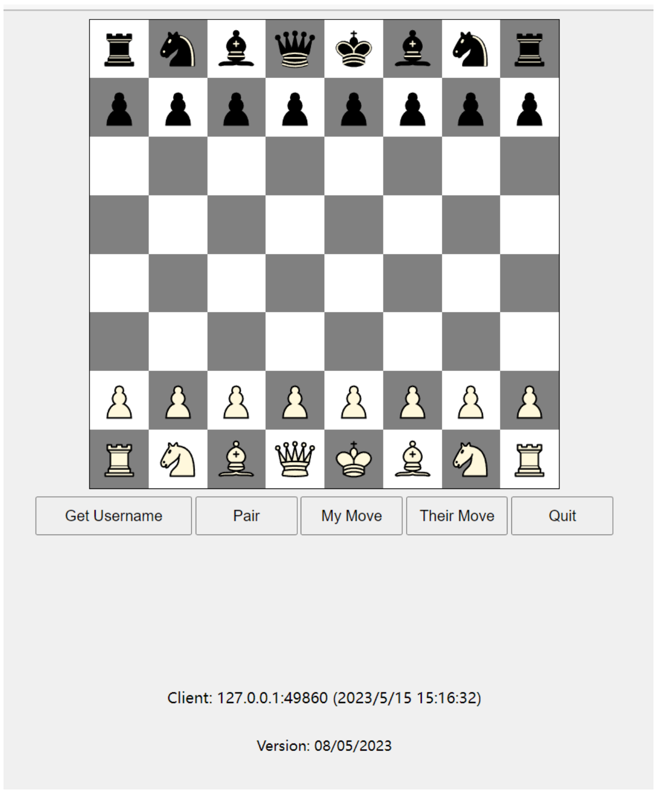
Use Visual Studio or any IDE open ./CS711-A2/ CS711 A2.sln and run it!

The console should log something like this.


```
C:\Users\dxh00\Documents\C
Game Server Starting
Game Server Started
Listening at 0.0.0.0:8080
```

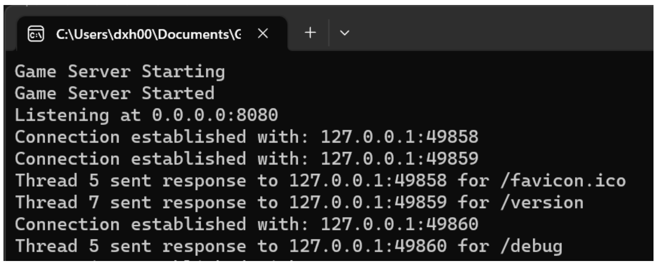
B. Frontend

After run backend use browser open ./CS711-A2/index,html.
Should see this page and some log from backend



Client: 127.0.0.1:49860 (2023/5/15 15:16:32)

Version: 08/05/2023


```
C:\Users\dxh00\Documents\C
Game Server Starting
Game Server Started
Listening at 0.0.0.0:8080
Connection established with: 127.0.0.1:49858
Connection established with: 127.0.0.1:49859
Thread 5 sent response to 127.0.0.1:49858 for /favicon.ico
Thread 7 sent response to 127.0.0.1:49859 for /version
Connection established with: 127.0.0.1:49860
Thread 5 sent response to 127.0.0.1:49860 for /debug
```

III. Functional

A. Backend

1) /register

This endpoint generates a random username for a player, registers this name, and returns to the user the registered name. The player is required to pass this username in all subsequent transactions for identification.

2) /pairme?player={username}

This endpoint attempts to pair the given player with another player. It returns a game record (or a suitable subset of it). The game record is a tuple containing a game ID, game state, username of the first player, user name of the second player, last move of the first player, and the last move of the second player.
When there is no other player waiting, the game record will contain a newly allocated game ID (which could be a GUID), a game state indicating "wait", and the username of the

requesting player as first player. The rest of the elements in the tuple are not defined. When there is a waiting player, the game record will be the game record first created for the waiting player, updated to add the second player and the state indicating "progress". The state "progress" tells both players that the game can now begin.
The endpoint can be invoked by both players as many times as they want before the commencement of the game. This helps the first player to "poll" the state to see if a second player has been paired up.

3) /mymove?player={username}&id={gameId}&move={move}

This endpoint, when used during the game's "progress", will supply the user's move to the server. The "last move" of the player in the corresponding game record will be updated with this supplied move.

4) /theirmove?player={username}&id={gameId}

This endpoint, when used during the game's "progress", will collect the other player's move from the server. The game server will supply the "last move" of the other player from the game record corresponding to the given game ID.

5) /quit?player={username}&id={gameId}

This endpoint notes to the server the intention of the player to quit the game. The server will remove the game record corresponding to the given game ID. Attempt by the players to access the game record (for example, to get a move) will fail after the record is removed.

6) /debug

This endpoint return connection information (Clientt IP, Time for connection)

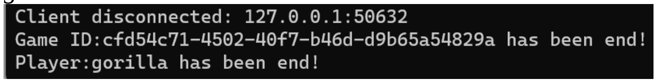
7) /version

This endpoint return Version for backed

8) / favicon.ico

This endpoint return favicon for game (UOA)

9) When the user closes the browser, the server will automatically disconnect and clean up the corresponding game and user data.



```
Client disconnected: 127.0.0.1:50632
Game ID:cfd54c71-4502-40f7-b46d-d9b65a54829a has been end!
Player:gorilla has been end!
```

Note: When a player in the game quits the game, another player will automatically join the waiting list until the next player is matched.
This function takes effect in handle client close and quit

B. *FrontEnd*

The front end contains a chessboard design and several buttons for the play function. When a player moves a piece, the front end automatically sends the entire board to the back end. At the same time, another player will send a check to the backend according to the set cycle time, and if a board change is found, it will be displayed on the frontend.

IV. TECHNIQUES

A. *Backend*

Socket
Volatile
Lock

B. *FrontEnd*

setInterval
fetch