Report of Assignment Two: Building A Game Server and A Client

Zexin Liao
zlia921@aucklanduni.ac.nz

THE PROBLEM

This assignment is mainly about building a C# console application named server as well as a web-based client application. The client is relatively easy, and there isn't a lot of requirements for the client. What I should be focused on is the construction of the server, to be more specific, the design of using socket and multi-threading to handle standard HTTP request. At the beginning of doing this assignment, I reckoned that it would be very complicated to handle HTTP request manually. But as soon as I realized that HTTP request is nothing else but a series of formatted string, the task has been very clear and straight forward to me.

ALTERNATIVES AND MY CHOICE

I think this part refers to what alternatives I have when choosing games that I would like to build for the client side. According to the instruction, the professor did not address importance on how good my client application is at restricting players with rules, so I chose the chess game. The rule of chess is a bit more complicated than other two-player game, especially when it comes to pawn promotion and en passant, etc. At the end, I did not choose to implement all the rules but those basic ones, such as pawn should be able to advance two squares for its first move.

I also have alternatives when choosing how to demonstrate that I have implement restrictions on players' move. The way I chose is to highlight the squares a piece can move to with colour green. I believe this is a way that is very much in line with human habits and intuition.

I don't think I have many alternatives when designing the server, because I am familiar with socket programming (thanks to assignment one) and multi-threading (which I have learnt from COMPSCI 718) and from the beginning I knew how this server should be designed, and in the end my idea proved to be a right one.

In general, I think my choice is good, and I will be able to make it more completed if I have more time.

DESIGN AND IMPLEMENTATION

I. Client

After observing the example of a chess client shown by the professor in class, I decided to improve on it. I wanted to make the interaction process as automated as possible, while still making it possible to send the requested HTTP requests. To this end, I made the following design:

- Combine the operation of registering a username with the operation of finding a match. The user will not need to do any copy and paste for the username. Once they clicked 'Find Match' button, the client will automatically retrieve a username first and display it on the page.
- Use interval to automatically find matches and automatically get opponent's moves. The client will start requesting '/pairme' endpoint every 1 second after it gets a username, and once the game start and this player has made his move, it will start requesting '/theirmove' endpoint every 1 second. This completely follow the instruction requirement and just make these actions all automatic, so the players can focus on the game.
- Highlight the squares a piece can move to with colour green, in addition, only at the correct moment and only the pieces that can be moved and highlighted squares will have click event listener function. This design can prevent the user from making illegal moves or make move in the wrong time, ensures that the game is played properly to the maximum extent possible.
- Use a signal box to display game status. The signal box is above the chess board, and it has four statuses. When it is this player's turn to move, the signal box will show 'Your Turn' with green background colour; when it is the opponent's turn, the signal box will show 'Opponent Turn' with red background colour; when this player or the opponent quit game, it will show 'You lose' with white background and 'You Win' with yellow background respectively. This design again enables players focus more on the game.

II. Server

According to the requirements, each client should be assigned a thread specifically to communicate with, so when considering how the server feeds information to the client, for example notifying the client when a pairing is successful, an intuition is that it may be necessary to implement a thread-to-thread communication. However, a closer look shows that in the current context, communication between client and server is always initiated by the client first, so communication between thread and thread is actually unnecessary. Based on this understanding, I made the following design on the server side:

- Use 'CurrentDictionary' to store players information and active games information. Player list and player moves will be recorded in such dictionaries so that when server receives requests from clients, it can always read from or

update these dictionaries, which ensure that all the responses clients could receive are correct and updated. So, we just need to carefully design the reaction of the client facing different responses.

- Use socket to send and receive data. Socket can send strings, and HTTP request is just formatted string, so there isn't any problem when applying this. I was not very familiar with what consist of a HTTP request, but after a little research I solved all the problems, such as CORS policy, sending JSON type content, etc.
- Manually handle incorrect request. Since socket has no built checking mechanism for HTTP request, so a challenge for me is to set my rules for checking requests that server received so that an incorrect request will not crash the server. I have implemented a lot of 'if' statements to achieve this.

Apart from the above, I don't think there's anything else worth noting about the design, it's all basic C# programming logic.

EVALUATION

I think there are so much more can be improved on both client side and server side, for example, the client should be able to reset the chess board when the game is over, and the server should be able to prevent one thread from pair with himself by register for two different username and request '/pairme' endpoint for each of them. For socket, I did not close it when the player quit, and for threads, I could have use thread pool to improve the performance. But I can possibly say that my applications have fulfil all the requirements to a high extend. Through this assignment, I have gained a deeper understanding of the nature of HTTP request and socket communication.

REFERENCE

[1] Microsoft. 2023. C# documentation. https://learn.microsoft.com/en-us/dotnet/csharp/. Web. Accessed: May 26, 2023.