CITS3002 Computer Networks

Project 2021

Tuguldur Gantumur (22677666)

University of Western Australia

June 5th 2021

How would you scale up your solution to handle many more clients?

Are there any other implementations outside the scope of this project you would like to mention?

My solution to the project is currently set to listen to 9 connections, where there will be up to 4 players and the remaining connections would be spectators. However for the server to handle more clients, it could be recommended to allow multiple separate games to commence at once. In such an event, I would implement a manager class that would handle many instances of one game at once. In addition, an implementation which would allow users to play with different players from each game after a round would be included, each game would then have to wait for the other to finish. The manager instance would have to receive a message instance from game instances when the game has reached a conclusion, and transfer each player instance to the shared player pool after their respective game has ended. Moreover, another possible implementation that would assist handling more clients would have been to implement a separate thread to handle messages.

How could you deal with identical messages arriving simultaneously on the same socket?

The server increments player objects stored in the global list in_game, and it will ignore any messages sent by a user if the user that sent the message is currently in the wait_list or is still awaiting for their turn. However, in such an event, I would assign each message with an id for each player instance, which would differentiate the time they were generated (for example, msg id: 0, msg id: 1). As a client can only send one message each round, if by any chance it has sent 2 messages, we would accept the message with the lowest id. In addition, in the case of the messages received simultaneously being duplicates a function could be implemented to parse the message and compare its content, and if a complete match does occur we could ignore the latter message. Another concern, would be the scenario if messages are merged or lost, in such case the server will not read or receive the message sent by the client, and we could send the client a message to resend the message. This implementation would require acknowledgements to be sent

by the server to the client. Taking into account the flow and structure of the game, I would have implemented a Stop n Wait Protocol for such a scenario.

With reference to your project, what are some of the key differences between designing network programs, and other programs you have developed?

The key differences between designing this network program and other programs I have developed, would be their structure. This project requires every client to receive and send messages to other clients on the server at least every turn . Its use of messaging protocols broadened my perspective of programming as I handled more edge cases in comparison to other programs. I had to take into consideration timeouts and a variety of setups at once and how it would impact the server and its efficiency. Another significant part of the project that greatly differed from other programs I have implemented in the past would be overseeing disconnections and eliminations, and making sure the client does not receive or send messages to disconnected or eliminated clients, as it would cause exceptions such as BrokenPipeError or ConnectionResetError.

What are the limitations of your current implementation (e.g. scale, performance, complexity)?

I believe that the most significant part of my implementation that could be improved are the complexity and performance of the server. I believe that implementing a data structure such as a Queue or PriorityQueue would have been more recommendable in regards to time complexity. In addition, I believe that if I had implemented a function to cache messages after each game. It would have improved efficiency and improved the latency. I was unable to send clients who joined during a commenced game the position of tiles and tokens from previous turns, I believe I could have achieved this if I implemented a Game instance to store every message sent by a player, as well as store updates which happened on the board. I believe I could have improved the servers' performance if I had achieved this feat.