CITS3002 Project Report

Jono Jones (22479345)

2021

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

By this I assume you mean scale it so the server can handle multiple games with different clients.

To allow my server to allow for many more clients I would make game rooms. Each game room would have would have a subset of the socket connections, and its own GameMaster (which has its own unique board). The game would just work as it does in my own project but the GameMaster would only communicate with it room's subset of players. (essentially being it's own micro server).

I would have to add a process for the allocation of clients to available rooms but I do not think that this would be the end of the world.

A potential limitation of this would be that the input and output lists would get very large, and if there were too many clients then it would take a long time to send all of the messages out to clients, and there would be a large delay between the server deciding what to do and actually sending the information to the client.

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

To handle this I could add a sequence number to the messages so that I know whether or not the sender had intended to send the exact same message twice. However I don't this that this would happen in this project, and also if it did I would be able to just take one of the messages and throw the other away, because the server should not be receiving two identical messages anyway.

With references to this project, what are some key differences between designing network programs, and other programs you have developed

In this case we had to design the network to suit a given client, so that required some coding forensics to look into how the client worked and build that understanding. Other programs that I have written for uni have not not required this kind of investigation.

In terms of thinking about the code, using select not threads, I had to keep in mind what stage of the code I was in. For example all of my game logic is done immediately after reading from a socket, and before writing to the socket. This meant that I could not just have my GameMaster send something and expect it to happen immediately, I had to wait until the program had reached the send stage. This caused me some grief with implementing timers and countdowns.

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

On my computer I could run the came with absolutely no problem with 200 clients, which to me is pretty good. This could partly because the server is only sending and receiving from 4 of the clients, but that is how the clients are sent up.

I did not have time to do any stress or load testing.

Another limitation is that when updating a client that joined late it could well send more information to than is required to the client, potentially soaking up network resources. I chose this implementation for simplicity but if I was finding a bit lag spike when new spectators joined to watch then I would probably change this implementation.

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

I think that a big limitation is that only game game can be played at a time. If there were 200 clients connected then I think it would be better if the server was running 50 games.

The random ai to select the what a player will do if they are idle for too long is very dumb, and can really throw that player under the bus.

Any other notable things to discuss..

When I manually test my code it handles have to clients be spectators if they join after the game starts. The supplied tester.py says that this is not working, but I would really appreciate it if you play test the project rather than letting a tester with specific pass requirements test it.